CS3401-ALGORITHMS

UNIT 1

TimeandSpaceComplexity

Time complexity is a measure of how long an algorithm takes to run as a function of the size of the input. It is typically expressed using big O notation, which describes the upper bound on the growth of the time required by the algorithm. For example, an algorithm with a time complexity of O(n) takes longer to run as the input size (n) increases.

Therearedifferenttypesoftimecomplexities:

- O(1) or constant time: the algorithm takes the same amount of time to run regardless of the size of the input.
- O(log n) or logarithmic time: the algorithm's running time increases logarithmically with the size of the input.
- O(n)orlineartime:thealgorithm'srunningtimeincreaseslinearlywiththesizeoftheinput.
- O(n log n) or linear logarithmictime: the algorithm's running time increases linearly with the size of the input and logarithmically with the size of the input.
- O(n^2) or quadratic time: the algorithm's running time increases quadratically with the size of the input.

- O(2^n) or exponential time: the algorithm's running time increases exponentially with the size of the input.

Space complexity, on the other hand, is a measure of how much memory an algorithm uses as a function of the size of the input. Like time complexity, it is typically expressed using big O notation. For example, an algorithm with a space complexity of O(n) uses more memory as the input size (n) increases. Space complexities are generally categorized as:

- O(1) or constant space: the algorithm uses the same amount of memory regardless of the size of the input.
- O(n) or linear space: the algorithm's memory usage increases linearly with the size of the input.
- O(n^2) or quadratic space: the algorithm's memory usage increases quadratically with the size of the input.
- O(2^n)orexponentialspace:thealgorithm'smemoryusageincreasesexponentiallywith the

- Big O notation (O(f(n))) provides an upper bound on the growth of a function. It describesthe worst-case scenario for the time or space complexity of an algorithm. For example, an algorithm with a time complexity of O(n^2) means that the running time of the algorithm is at most n^2, where n is the size of the input.
- Big Ω notation (Ω(f(n))) provides a lower bound on the growth of a function. It describes the best-case scenario for the time or space complexity of an algorithm. For example, an algorithm with a space complexity of Ω(n) means that the memory usage of the algorithm is at least n, where n is the size of the input.
- Big Θ notation (Θ(f(n))) provides a tight bound on the growth of a function. It describes the average-case scenario for the time or space complexity of an algorithm. For example, an algorithm with a time complexity of Θ(n log n) means that the running time of the algorithm is both O(n log n) and Ω(n log n), where n is the size of the input.

It's important to note that the asymptotic notation only describes the behavior of the function for large values of n, and does not provide information about the exact behavior of the function for small values of n. Also, for some cases, the best, worst and average cases can be the same, in that case the notation will be simplified to O(f(n)) = Ω(f(n)) = Θ(f(n))

Additionally, these notations can be used to compare the efficiency of different algorithms, where a lower order of the function is considered more efficient. For example, an algorithm with a time complexity of O(n) is more efficient than an algorithm with a time complexity of O(n^2).

It's also worth mentioning that asymptotic notation is not only limited to time and space complexity but can be used to express the behavior of any function, not just algorithms.

Thereare three asymptoticnotations that areused torepresent the time complexityof analgorithm. They are:

- **Input:**Hereourinputisanintegerarrayofsize"n"andwehaveoneinteger"k"thatwe need to search for in that array.
- **Output:**Iftheelement"k"isfoundinthearray,thenwehavereturn1,otherwisewehave

```
//for-looptoiteratewitheachelementinthe array
for (inti = 0;i <n;++i)
{
  //checkifithelement isequalto"k"ornot
  if(arr[i]==k)
    return1;//return1,ifyoufind"k"
```

```
  }
  return0;//return0,ifyoudidn'tfind"k"
}
```

- If the input array is [1, 2, 3, 4, 5] and you want to find if "1" is present in the array or not, thenthe if-condition ofthe code willbe executed 1 time andit willfind that the element 1 is there in the array. So, the if-condition will take 1 second here.
- If the input array is [1, 2, 3, 4, 5] and you want to find if "3" is present in the array or not, then the if-condition of the code will be executed 3 times and it will find that the element 3is there in the array. So, the if-condition will take 3 seconds here.
- If the input array is [1, 2, 3, 4, 5] and you want to find if "6" is present in the array or not, then the if-condition of the code will be executed 5 times and it will find that the element 6is not there in the array and the algorithm will return 0 in this case. So, the if-condition will take 5 seconds here.

As we can see that for the same input array, we have different time for different values of "k".
So,this can be divided into three cases:

- **Best case:** This is the lower bound on running time of an algorithm. We must know the case that causes the minimum number of operations to be executed. In the above example, our array was [1, 2, 3, 4, 5] and we are finding if "1" is present in the array or not. So here, after only one comparison, we will get that ddelement is present in the array. So, this is the best case of our algorithm.

- **Average case:** We calculate the running time for all possible inputs, sum all the calculated values and divide the sum by the total number of inputs. We must know (or predict) distribution of cases.
- **Worst case:** This is the upper bound on running time of an algorithm. We must know the case that causes the maximum number of operations to be executed. In our example, the worst case can be if the given array is [1, 2, 3, 4, 5] and we try to find if element "6" is present in the array or not. Here, the if-condition of our loop will be executed 5 times and then the algorithm will give "0" as output.

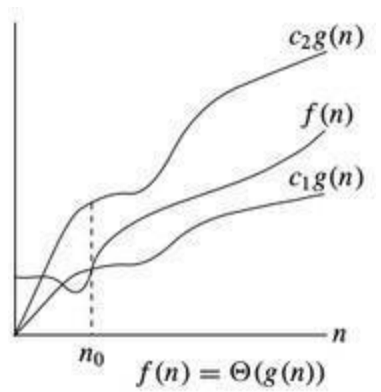
So, we learned about the best, average, and worst case of an algorithm. Now, let's get back to the asymptotic notation where we saw that we use three asymptotic notation to represent the complexity of an algorithm i.e. Θ Notation (theta), Ω Notation, Big O Notation.

NOTE: Intheasymptoticanalysis,wegenerallydealwithlargeinput size.

ΘNotation(theta)

The Θ Notation is used to find the average bound of an algorithm i.e. it defines an upper bound anda lower bound, and your algorithm will lie in between these levels. So, if a function is g(n), then the theta representation is shown as Θ(g(n)) and the relation is shown as:

Θ(g(n))={f(n):thereexistpositiveconstantsc1,c2andn0



ΩNotation

The Ω notation denotes the lower bound of an algorithm i.e. the time taken by the algorithm can'tbe lower thanthis.Inotherwords, thisisthefastesttimeinwhichthealgorithmwillreturn aresult.

Its the time taken by the algorithm when provided with its best-case input. So, if a function is g(n), then the omega representation is shown as Ω(g(n)) and the relation is shown as:

Ω(g(n))={f(n):thereexistpositiveconstantscandn0 such

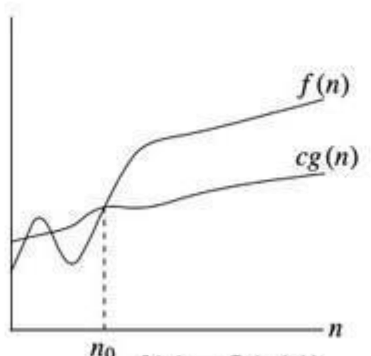
that $0 \le cg(n) \le f(n)$ **for** all $n \ge n0$ }

Theaboveexpressioncan bereadas omegaofg(n)isdefinedassetofallthe functionsf(n)forwhich there exist some constants c and n0 such that c*g(n) is less than or equal to f(n), for all n greaterthan or equal to n0.

if$f(n)=2n^2+3n+1$ and

g$(n) = n^2$

then**for** c=2 andn0=1,wecansaythat**f**$(n)=\Omega(n^2)$



$f(n) = \Omega(g(n))$

BigONotation

The Big Onotation definesthe upper bound ofany algorithm i.e.you algorithm can't take more time than this time. In other words, we can say that the big O notation denotes the maximum time taken by an algorithm or the worst-case time complexity of an algorithm. So, big O notation is the most used notation for the time complexity of an algorithm. So, if a function is g(n), then the big O representation of g(n) is shown as O(g(n)) and the relation is shown as:

O(g(n))={f(n):thereexistpositiveconstantscandn0 such

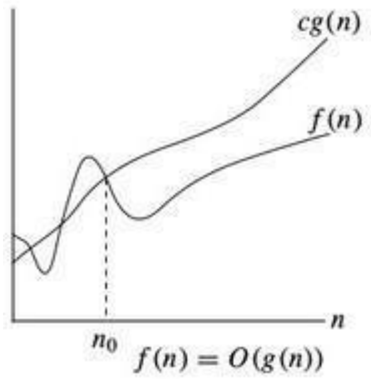
that $0 \le f(n) \le cg(n)$ **for** all $n \ge n0$ }

Theabove expression can be read as Big O of g(n) is defined as a set offunctions f(n) for which there exist some constants c and n0 such that f(n) is greater than or equal to 0 and f(n) is smaller than or equal to c*g(n) for all n greater than or equal to n0.

if$f(n)=2n^2+3n+1$ and

g$(n) = n^2$

then**for** c=6 andn0=1,wecansaythat**f**$(n)=O(n^2)$



BigOnotationexampleofAlgorithms

Big O notation is the most used notation to express the time complexity of an algorithm. In this section of the blog, we will find the big O notation of various algorithms.

Example1:Findingthesumofthefirstn numbers.

In this example, we have to find the sum of first n numbers. For example, if n = 4, then our output should be 1 + 2 + 3 + 4 = 10. If n = 5, then the ouput should be 1 + 2 + 3 + 4 + 5 = 15. Let's try various solutions to this code and try to compare all those codes.

O(1)solution

```
//functiontakinginput"n"

intfindSum(intn)

{

  returnn*(n+1)/2;//thiswilltakesomeconstanttimec1

}
```

In the above code, there is only one statement and we know that a statement takes constant time for its execution. The basic idea is that if the statement is taking constant time, then it will take the same amount of time for all the input size and we denote this as ***O(1)*** .

O(n)solution

In this solution, we will run a loop from 1 to n and we will add these values to a variable named "sum".

```
//functiontakinginput"n"

intfindSum(intn)

{

  intsum=0;//------------------->ittakessomeconstanttime"c1"

  for(inti= 1;i <=n; ++i)//--> herethecomparisionand increment willtakeplace ntimes(c2*n)and the
creation of i takes place with some constant time

      sum=sum+i;//-------------->thisstatementwillbeexecutedntimesi.e. c3*n
```

```
   returnsum;// ------------------>ittakessomeconstanttime"c4"
}
/*
* Totaltimetaken=timetakenbyallthestatmentstoexecute
* here in our example we have 3 constant time taking statements i.e. "sum = 0", "i = 0", and "return sum", so we can add all the constatnts and replacce with some new constant "c"
* apart fromthis, we havetwo statementsrunning n-timesi.e. "i< n(in realn+1)"and "sum= sum+ i" i.e. c2*n + c3*n = c0*n
* Totaltimetaken=c0*n+c
*/
```

The big O notation of the above code is O(c0*n) + O(c), where c and c0 are constants. So,the overall time complexity can be written as ***O(n)*** .

$O(n^2)$solution

In this solution, we will increment the value of sum variable "i" times i.e. for i = 1, the sum variable will be incremented once i.e. sum = 1. For i = 2, the sum variable will be incremented twice. So, let's see the solution.

```
//functiontakinginput"n"
intfindSum(intn)
{
   intsum=0;//---------------------- >constanttime
   for(inti= 1;i<=n;++i)
      for(intj=1;j<=i;++j)
         sum++;//------------------ >itwillrun[n*(n+1)/2]
   returnsum;// --------------------- >constant time
}
/*
* Totaltimetaken=timetakenbyallthestatmentstoexecute
* thestatement thatisbeingexecutedmostofthetime is"sum++"i.e.n*(n+1)/2
* So, total complexity will be: c1*n² + c2*n + c3 [c1 is for the constant terms of n², c2 is for the constant terms of n, and c3 is for rest of the constant time]
*/
```

The big O notation of the above algorithm is $O(c1*n^2)$ +O(c2*n) + O(c3). Since we take the higher order of growth in big O. So, our expression will be reduced to ***$O(n^2)$*** .

So,until now,we saw 3 solutions for the same problem. Now, whichalgorithm will you prefer to use whenyouarefindingthesumoffirst "n"numbers?If youranswerisO(1)solution,thenwehaveone bonus section for you at the end of this blog. We would prefer the O(1) solution because the time taken by the algorithm will be constant irrespective of the input size.

RecurrenceRelation

A recurrence relation is a mathematical equation that describes the relation between the input size and the running time ofa recursive algorithm.It expressesthe running time of aproblem intermsof the running time of smaller instances of the same problem.

ArecurrencerelationtypicallyhastheformT(n)=aT(n/b)+f(n)where:

- T(n)istherunningtimeofthealgorithmonaninputofsizen
- aisthenumberofrecursivecallsmadebythealgorithm
- bisthesizeoftheinputpassedtoeachrecursivecall
- f(n)isthetimerequiredtoperformanynon-recursiveoperations

The recurrence relation can be used to determine the time complexity of the algorithm using techniques such as the Master Theorem or Substitution Method.

For example, let's consider the problem of computing the nth Fibonacci number. A simple recursive algorithm for solving this problem is as follows:

```
Fibonacci(n)

if n <= 1

return nelse

returnFibonacci(n-1)+Fibonacci(n-2)
```

The recurrencerelationforthisalgorithmisT(n)=T(n-1)+T(n-2)+ O(1),whichdescribesthe running time of the algorithm in terms of the running time of the two smaller instances of the problem with input sizes n-1 and n-2. Using the Master Theorem, it can be shown that the time complexity of this algorithm is O(2^n) which is very inefficient for large input sizes.

Searching

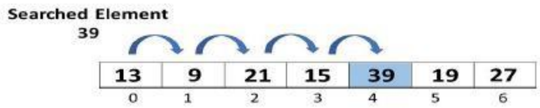
Searching is the process of fetching a specific element in a collection of elements. The collection can be an array or a linked list. If you find the element in the list, the process is considered successful, and it returns the location of that element.
Two prominent search strategies are extensively used to find a specific item on a list. However, the algorithm chosen is determined by the list's organization.

1. LinearSearch
2. BinarySearch
3. Interpolationsearch

LinearSearch

Linear search, often known as sequential search, is the most basic search technique. In this type of search,wegothroughtheentirelistandtrytofetchamatchforasingleelement.Ifwe find a match, then the address of the matching target element is returned.

On the other hand, if the element is not found, then it returns a NULL value. Followingisastep-by-stepapproachemployedtoperformLinearSearchAlgorithm.



Theproceduresforimplementinglinearsearchareasfollows:

Step1:First,readthesearchelement(Targetelement)inthearray.

Step2:Inthesecondstepcomparethesearchelementwiththefirstelementinthearray.

Step3:Ifbotharematched,display"Targetelementisfound"andterminatetheLinearSearch function.

Step 4: If both are not matched, compare the search element with the next element in the array.

Step 5: In this step, repeat steps 3 and 4 until the search (Target) element is compared with the last element of the array.

Step 6 - If the last element in the list does not match, the Linear Search Function will be terminated, and the message "Element is not found" will be displayed.

AlgorithmandPseudocodeofLinearSearchAlgorithm Algorithm of the Linear Search Algorithm

```
LinearSearch(ArrayArr,Value a)//Arristhenameofthe array,andaisthesearchedelement. Step 1: Set i
to 0 // i is the index of an array which starts from 0
Step2:ifi>nthengotostep7//nisthe numberofelementsinarray Step 3: if
Arr[i] = a then go to step 6
Step4:Setitoi+1
Step5:Gotostep2
Step6:Printelementafoundatindexiandgotostep8 Step 7:
Print element not found
Step8:Exit
```

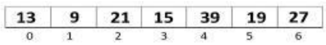
PseudocodeofLinearSearchAlgorithm

```
Start
linear_search(Array,value)
```

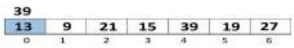
```
Foreachelementinthearray
   If(searchedelement==value)
      Return'sthesearchedelementlocation end
   if
endfor
end
```

ExampleofLinearSearchAlgorithm

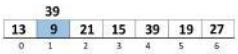
Consider anarrayofsize7withelements13,9,21,15,39,19,and27thatstartswith0andends with size minus one, 6.
Searchelement=39



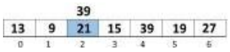
Step1:Thesearchedelement39iscomparedtothefirstelementofanarray,whichis13.



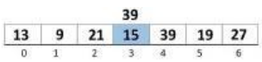
Thematchisnotfound,younowmoveontothenextelementandtrytoimplement acomparison. Step 2: Now, search element 39 is compared to the second element of an array, 9.



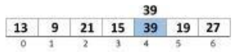
Step3:Now,searchelement39iscomparedwiththethirdelement,whichis21.



Again,boththeelementsarenotmatching,youmoveontothenextfollowingelement. Step 4;
Next, search element 39 is compared with the fourth element, which is 15.



Step5:Next,searchelement39iscomparedwiththefifthelement39.



Aperfectmatchisfound,displaytheelementfoundatlocation4.

TheComplexityofLinearSearchAlgorithm

Three different complexities faced while performing Linear Search Algorithm, they are mentioned as follows.

1. BestCase
2. WorstCase
3. AverageCase

BestCase Complexity

- Theelementbeingsearchedcouldbefoundinthefirstposition.
- Inthiscase,thesearchendswithasinglesuccessful comparison.
- Thus,inthebest-casescenario,thelinearsearchalgorithmperformsO(1)operations.

WorstCaseComplexity

- Theelementbeingsearchedmaybeatthelastpositioninthearrayornotat all.
- Inthefirstcase,thesearchsucceedsin‘n’comparisons.
- Inthenextcase,thesearchfailsafter‘n’ comparisons.
- Thus,intheworst-casescenario,thelinearsearchalgorithmperformsO(n)operations.

AverageCaseComplexity

Whentheelementto be searchedisinthe middleofthe array,the averagecase ofthe LinearSearch Algorithm is O(n).

SpaceComplexityofLinearSearchAlgorithm

Thelinearsearchalgorithmtakesupnoextraspace;itsspacecomplexityisO(n)foranarrayofn elements.

ApplicationofLinearSearchAlgorithm

Thelinearsearchalgorithmhasthefollowingapplications:

- Linearsearchcanbeappliedtobothsingle-dimensionalandmulti-dimensionalarrays.
- Linearsearchiseasytoimplementandeffectivewhenthearraycontainsonlyafewelements.
- LinearSearchisalsoefficientwhenthesearchisperformedtofetchasinglesearchinan unordered-List.

CodeImplementationofLinearSearchAlgorithm

```
#include<stdio.h>
#include<stdlib.h>
#include<conio.h>
int main()
{
  intarray[50],i,target,num;
```

```
    printf("Howmanyelementsdoyouwantinthearray"); scanf("%d",&num);
    printf("Enterarrayelements:");
    for(i=0;i<num;++i)
       scanf("%d",&array[i]);
    printf("Enterelementtosearch:");
    scanf("%d",&target);
    for(i=0;i<num;++i)
       if(array[i]==target)
          break;
    if(i<num)
       printf("Targetelementfoundatlocation%d",i); else
       printf("Targetelementnotfoundinanarray"); return
    0;
}
```

BinarySearch

Binary search is the search technique that works efficiently on sorted lists. Hence, to search an element into some list using the binary search technique, we must ensure that the list is sorted.
Binary search follows the divide and conquer approach in which the list is divided into two halves, and the item is compared with the middle element of the list. If the match is found then, thelocationofthe middle elementisreturned.Otherwise,wesearchintoeitherofthehalvesdepending upon the result produced through the match
NOTE: Binary search can be implemented on sorted array elements. If the list elements are not arranged in a sorted manner, we have first to sort them.

Algorithm

1. Binary_Search(a,lower_bound, upper_bound, val) //'a' is the given array,'lower_bound' is t he index ofthe first array element, 'upper_bound'is the indexof the last array element, 'val' is the value to search
2. Step1:setbeg=lower_bound,end=upper_bound,pos=-1
3. Step2:repeatsteps3 and4 whilebeg**<**=end
4. Step3:setmid=(beg+ end)/2
5. Step4:ifa[mid]=val
6. setpos =mid
7. printpos
8. gotostep6
9. elseifa[mid]**>**val
10. setend= mid-1
11. else
12. setbeg= mid+1
13. [endofif]
14. [endof loop]
15. Step5:if pos=-1

16. print"valueisnotpresentinthearray"
17. [endofif]
18. Step6:exit

Procedurebinary_search

```
  A←sortedarray
  n←sizeof array
  x←valuetobesearched Set
  lowerBound = 1
  SetupperBound=n
  while x not found
    ifupperBound<lowerBound EXIT:
      x does not exists.
      setmidPoint=lowerBound+(upperBound-lowerBound)/2 if
    A[midPoint] < x
      setlowerBound=midPoint+1 if
    A[midPoint] > x
      setupperBound=midPoint-1 if
    A[midPoint] = x
      EXIT:xfoundatlocationmidPoint end
  while
 end procedure
```

WorkingofBinarysearch

To understand the working of the Binary search algorithm, let's take a sorted array. It will be easy to understand the working of Binary search with an example.

Therearetwomethodstoimplementthebinarysearchalgorithm-

- Iterativemethod
- Recursivemethod

Therecursivemethodofbinarysearchfollowsthedivideandconquerapproach. Let the elements of array are -

0	1	2	3	4	5	6	7	8
10	12	24	29	39	40	51	56	69

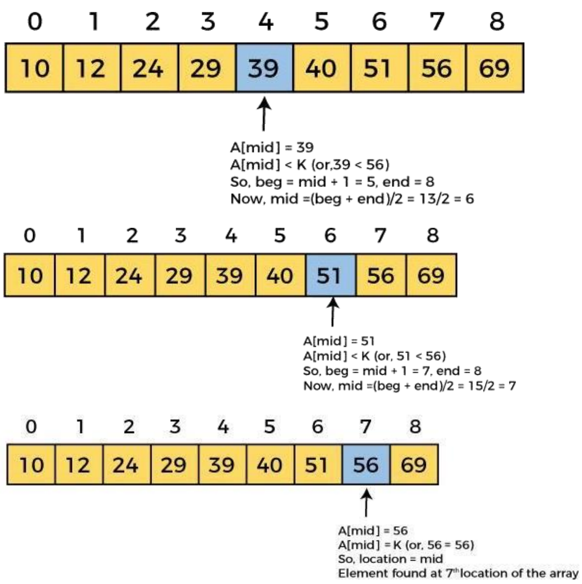
Lettheelementtosearchis,**K= 56**

Wehavetousethebelowformulatocalculatethe**mid**ofthearray-

1. mid=(beg+end)/2

So, in the given array -

beg= 0
end=8
mid=(0+ 8)/2= 4.So,4is themidofthe array.



Now,the elementtosearchisfound.Soalgorithmwillreturntheindexoftheelementmatched. Binary Search complexity

Now, let's see the time complexity of Binary search in the best case, average case, and worst case.We will also see the space complexity of Binary search.

1. TimeComplexity

Case	TimeComplexity
BestCase	O(1)
AverageCase	O(logn)
WorstCase	O(logn)

- **Best Case Complexity -** In Binary search, best case occurs when the element to search is found in first comparison, i.e., when the first middle element itself is the element to be searched. The best-case time complexity of Binary search is **O(1).**
- **AverageCaseComplexity-**TheaveragecasetimecomplexityofBinarysearchis**O(logn).**
- **Worst Case Complexity -** In Binary search, the worst case occurs, when we have to keep reducing the search space till it has only one element. The worst-case time complexity of Binary search is **O(logn).**

2. Space Complexity

SpaceComplexity	O(1)

- ThespacecomplexityofbinarysearchisO(1).

ImplementationofBinarySearch

Program: WriteaprogramtoimplementBinarysearchinClanguage.

```
#include<stdio.h>
intbinarySearch(inta[],intbeg,intend,intval)
{
  intmid;
  if(end>=beg)
```

```
    {      mid=(beg+end)/2;
/*ifthe itemtobe searchedispresentatmiddle*/
        if(a[mid]== val)
        {
            returnmid+1;
        }
            /* if the item to be searched is smaller than middle, thenit can onlybe in left subarra y
*/
        elseif(a[mid]<val)
        {
            returnbinarySearch(a,mid+1,end,val);
        }
            /*if the itemto be searchedis greater than middle,thenit can onlybe in right subarr ay
*/
        else
        {
            returnbinarySearch(a,beg,mid-1,val);
        }
    }
    return-1;
}
intmain(){
 inta[]={11,14,25,30,40,41,52,57,70};//givenarray
 intval= 40;//valuetobesearched
 intn=sizeof(a)/sizeof(a[0]);//sizeofarray
 intres=binarySearch(a,0,n-1,val);//Storeresult
 printf("Theelementsofthearrayare-");
 for(inti =0;i<n;i++)
 printf("%d",a[i]);
 printf("\nElementtobesearchedis-%d",val);
 if(res==-1)
 printf("\nElementisnotpresentinthearray");
 else
 printf("\nElementispresentat%dpositionofarray",res);
 return0;
}
```

Output

```
The elements of the array are - 11 14 25 30 40 41 52 57 70
Element to be searched is - 40
Element is present at 5 position of array
```

InterpolationSearch

Interpolation search is an improved variant of binary search. This search algorithm works on the probing position of the required value. For this algorithm to work properly, the data collectionshould be in a sorted form and equally distributed.

Binary search has a huge advantage of time complexity over linear search. Linear search has worst-case complexity of O(n) whereas binary search has O(log n).

There are cases where the location of target data may be known in advance. For example, in case of a telephone directory, if we want to search the telephone number of Morphius. Here, linear search and even binary search will seem slow as we can directly jump to memory space where the names start from 'M' are stored.

PositionProbinginInterpolationSearch

Interpolation search finds a particular item by computing the probe position. Initially, the probe position is the position of the middle most item of the collection.

If a match occurs, then the index of the item is returned. To split the list into two parts, we use the following method −

mid=Lo+((Hi-Lo)/(A[Hi]-A[Lo]))* (X-A[Lo])

where

−A=list

Lo=Lowestindexofthelist Hi=

Highestindexofthe list

A[n]=Valuestoredatindexninthelist

If the middle item is greater than the item, then the probe position is again calculated in the sub-array to the right of the middle item. Otherwise, the item is searched in the subarray to the left of the middle item. Thisprocess continueson the sub-array as welluntil the size ofsubarray reducesto zero.

Runtime complexity of interpolation search algorithm is **O(log (log n))** as compared to **O(log n)** ofBST in favorable situations.

Algorithm

AsitisanimprovisationoftheexistingBSTalgorithm,wearementioningthestepstosearchthe 'target' data value index, using position probing −

Step1−Startsearching**data**frommiddleofthelist.

Step2−Ifitisamatch,returntheindexoftheitem,andexit. Step 3 −

If it is not a match, probe position.

Step4−Dividethelistusingprobingformulaandfind thenewmidle. Step 5 −

If data is greater than middle, search in higher sub-list.

Step6−Ifdataissmallerthanmiddle,searchinlowersub-list. Step 7

− Repeat until match.

PseudocodeA

→Arraylist

N→Size ofA

X→TargetValue

ProcedureInterpolation_Search()

Set Lo→0

```
  Set Mid → -1
  SetHi→N-1

  WhileXdoesnotmatch

    ifLoequalstoHiORA[Lo]equalsto A[Hi]
      EXIT:Failure,Targetnotfound
    end if

    SetMid=Lo+ ((Hi-Lo)/ (A[Hi]-A[Lo]))*(X- A[Lo])

    ifA[Mid]=X
      EXIT:Success,TargetfoundatMid else
      ifA[Mid]<X
          SetLotoMid+1
       else if A[Mid] > X
          Set Hi to Mid-1
      endif
    end if
  End While

EndProcedure
```

ImplementationofinterpolationinC

```
#include<stdio.h>#defi
ne MAX 10
//arrayofitemsonwhichlinearsearchwillbeconducted. int
list[MAX] = { 10, 14, 19, 26, 27, 31, 33, 35, 42, 44 };
intfind(intdata){ int
  lo = 0;
  inthi=MAX-1; int
  mid = -1;
  intcomparisons=1;
  int index = -1;
  while(lo <= hi) {
    printf("\nComparison%d\n",comparisons);
    printf("lo:%d,list[%d]=%d\n",lo,lo,list[lo]);
    printf("hi:%d,list[%d]=%d\n",hi,hi, list[hi]);

    comparisons++;
    //probethemidpoint
    mid=lo+(((double)(hi-lo)/(list[hi]-list[lo]))*(data-list[lo]));
    printf("mid = %d\n",mid);
    // data found
    if(list[mid]==data){
      index=mid;
      break;
```

```
    }else{
      if(list[mid]<data){
        //ifdataislarger,dataisinupperhalf lo =
        mid + 1;
      }else{
        //ifdataissmaller,dataisinlowerhalf hi =
        mid - 1;
      }
    }
  }

  printf("\nTotalcomparisonsmade:%d",--comparisons); return
  index;
}
intmain(){
  //find location of 33
  intlocation=find(33);

  //ifelementwasfound
  if(location != -1)
    printf("\nElementfoundatlocation:%d",(location+1)); else
    printf("Elementnotfound.");
  return 0;
}
```

Ifwecompileandruntheabove program,itwillproducethefollowingresult− Output

```
Comparison1
lo:0,list[0]= 10
hi:9,list[9]=44
mid=6

Total comparisons made: 1
Elementfoundatlocation:7
```

TimeComplexity

- **Bestcase-O(1)**
  The best-case occurs when the target is found exactly as the first expected position computed using the formula. As we only perform one comparison, the time complexity is O(1).

- **Worst-case-O(n)**
  Theworstcaseoccurswhenthegivendatasetisexponentiallydistributed.

- **Averagecase-O(log(log(n)))**
  If the data set is sorted and uniformly distributed, then it takes O(log(log(n))) time as on an average (log(log(n))) comparisons are made.

SpaceComplexity
O(1)asnoextraspaceisrequired.

PatternSearch

Pattern Searching algorithms are used to find a pattern or substring from another bigger string.There are different algorithms. The main goal to design these type of algorithms to reduce the time complexity. The traditional approach may take lots of time to complete the pattern searching taskfor a longer text.
Herewewillseedifferentalgorithmstoget abetterperformanceofpatternmatching. In this Section We are going to cover.

- Aho-CorasickAlgorithm
- AnagramPatternSearch
- BadCharacterHeuristic
- BoyerMooreAlgorithm
- EfficientConstructionofFiniteAutomata
- kasai'sAlgorithm
- Knuth-Morris-PrattAlgorithm
- Manacher'sAlgorithm
- NaivePatternSearching
- Rabin-KarpAlgorithm
- SuffixArray
- TrieofallSuffixes
- ZAlgorithm

Naïve pattern searching is the simplest method among other pattern searching algorithms. It checks for all character of the main string to the pattern. This algorithm is helpful for smaller texts. It does not need any pre-processing phases. We can find substring by checking once for the string. It also does not occupy extra space to perform the operation.
The time complexity of Naïve Pattern Search method is O(m*n). The m is the size of pattern and n is the size of the main string.

InputandOutput
Input:
MainString:"ABAAABCDBBABCDDEBCABC",pattern:"ABC"
Output:
Pattern found at position: 4
Patternfoundatposition:10
Patternfoundatposition:18

Algorithm

naive_algorithm(pattern,text)
Input−Thetextandthepattern
Output−locations,wherethepatternispresentinthetext
Start
pa _len:=patternSize

```
  str_len:=string size
  fori:=0to(str_len-pat_len),do for j
    := 0 to pat_len, do
      iftext[i+j]≠pattern[j],then
      break
  ifj==patLen,then
  displaythepositioni,astherepatternfound
End
```

ImplementationinC

```
#include  <stdio.h>
#include<string.h>
int main (){
  chartxt[]="tutorialsPointisthebestplatformforprogrammers"; char
  pat[] = "a";
  intM=strlen(pat); int
  N = strlen (txt);
  for(inti=0;i<=N-M;i++){ int j;
    for (j = 0; j < M;
      j++)if(txt[i+j]!=pat[j
      ])
    break;
    if(j==M)
                                                    printf      ("Pattern      matches      at      index      %d
",i);
  }
   return0;
}
```

Output

Pattern matches at 6

Patternmatchesat25

Patternmatchesat 39

Rabin-Karpmatchingpattern

Rabin-Karp is another pattern searching algorithm. It is the string matching algorithm that was proposed by Rabin and Karp to find the pattern in a more efficient way. Like the Naive Algorithm, it alsochecksthe pattern bymoving the window oneby one,but withoutchecking allcharactersforall cases, it finds the hash value. When the hash value is matched, then only it proceeds to check each character. In this way, there is only one comparison per text subsequence making it a more efficient algorithm for pattern searching.

Preprocessingtime-O(m)

ThetimecomplexityoftheRabin-KarpAlgorithmis**O(m+n)**,butfortheworstcase,itis**O(mn)**.

Algorithm

rabinkarp_algo(text,pattern,prime)

Input−Themaintextandthepattern.Anotherprimenumberoffindhash location

Output–locations,wherethepatternisfound
Start
  pat_len:=patternLength
  str_len := string Length
  patHash:=0 andstrHash:=0,h:=1
  maxChar:=totalnumberofcharactersincharacterset for
index i of all character in the pattern, do
  h:=(h*maxChar)modprime
forallcharacterindexiofpattern,do
  patHash:=(maxChar*patHash+pattern[i])modprime strHash
  := (maxChar*strHash + text[i]) mod prime
fori:=0to(str_len-pat_len),do if
  patHash = strHash, then
    forcharIndex:=0 topat_len-1,do
      iftext[i+charIndex]≠pattern[charIndex],then
      break
ifcharIndex=pat_len, then
  printthelocationiaspatternfoundatiposition. if i <
(str_len - pat_len), then
  strHash:=(maxChar*(strHash–text[i]*h)+text[i+patLen])modprime,then if
  strHash < 0, then
  strHash:=strHash+prime
End

ImplementationInC

```
#include<stdio.h>
#include<string.h>
int main (){
  chartxt[80],pat[80];
  int q;
 printf("Enterthecontainerstring");
 scanf ("%s", &txt);
 printf("Enterthepatterntobesearched");
  scanf ("%s", &pat);
  int d = 256;
 printf("Enteraprimenumber");
 scanf ("%d", &q);
  intM=strlen(pat);
  int N = strlen (txt);
  int i, j;
  intp=0;
  int t = 0;
  inth=1;
  for(i=0;i<M-1;i++)  h  =
    (h * d) % q;
  for(i=0;i<M;i++){
    p= (d*p+ pat[i])%q;
```

```
    t=(d*t+txt[i])%q;
  }
  for(i=0;i<=N-M;i++){ if (p
    == t){
      for (j = 0; j < M; j++){
        if(txt[i+j]!=pat[j])
        break;
      }
      if (j == M)
printf("Patternfoundatindex%d",i);
    }
    if(i<N-M){
      t=(d*(t-txt[i]*h)+txt[i+M])%q; if (t < 0)
        t=(t+q);
    }
  }
  return0;
}
```

Output
Enter the container string
tutorialspointisthebestprogrammingwebsite
Enter the pattern to be searched
p
Enteraprimenumber 3
Pattern found at index 8
Patternfoundatindex21

nthisproblem,wearegiventwostringsatextandapattern.Ourtaskistocreateaprogramfor KMP algorithm for pattern search, it will find all the occurrences of pattern in text string.

Here,wehavetofindalltheoccurrencesofpatternsinthetext.

Let’stakeanexampletounderstandtheproblem,

Input
text=“xyztrwqxyzfg”pattern=“xyz” Output
Foundatindex0
Foundatindex7

Here, we will discuss the solution to the problem using KMP (*Knuth Morris Pratt*) pattern searching algorithm, it will use a preprocessing string ofthe pattern whichwill be usedfor matching inthe text. And help’s in processing or finding pattern matches in the case where matching characters are followed by the character of the string that does not match the pattern.

We will preprocess the pattern wand to create an array that contains the proper prefix and suffix from the pattern that will help in finding the mismatch patterns.

ProgramforKMPAlgorithmforPatternSearching

```
//CProgramforKMPAlgorithmforPatternSearching Example
#include<iostream>
```

```
#include<string.h>usin
gnamespacestd;
voidprefixSuffixArray(char*pat,intM,int*pps){ int
  length = 0;
  pps[0] = 0;int
  i =
  1;while(i<M){
    if(pat[i]==pat[length]){
      length++;
      pps[i]=length;
      i++;
    }else{
      if(length!=0)
      length=pps[length-1];
      else {
        pps[i]=0;
        i++;
      }
    }
  }
}
voidKMPAlgorithm(char*text,char*pattern){
  int M = strlen(pattern);
  intN=strlen(text); int
  pps[M];
  prefixSuffixArray(pattern,M,pps); int
  i = 0;
  int j = 0;
  while(i<N){
    if(pattern[j]==text[i]){ j++;
      i++;
    }
    if(j==M)
 {
 printf("Foundpatternatindex%d",i-j); j
      = pps[j - 1];
    }
    elseif(i<N&&pattern[j]!=text[i]){ if (j
      != 0)
      j=pps[j-1];
      else
      i =i+1;
    }
  }
}
intmain(){
  chartext[]="xyztrwqxyzfg";
```

```
    char pattern[] = "xyz";
   printf("Thepatternisfoundinthetextatthefollowingindex:");
   KMPAlgorithm(text, pattern);
    return0;
}
```

Output

Thepatternisfoundinthetextatthefollowingindex− Found
pattern at index 0
Foundpatternatindex7

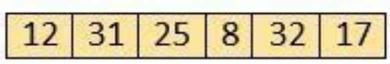
Sorting:Insertionsort

Insertionsort workssimilarto thesorting ofplayingcardsinhands. It isassumedthatthe first cardis already sorted in the card game, and then we select an unsorted card. If the selected unsorted cardis greater than the first card, it will be placed at the right side; otherwise, it will be placed at the left side. Similarly, all unsorted cards are taken and put in their exact place.

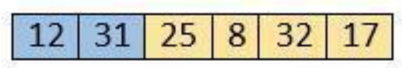
The same approach is applied in insertion sort. The idea behind the insertion sort is that first take one element,iterate it through the sortedarray.Although it issimple to use,it is not appropriatefor large data sets as the time complexity of insertion sort in the average case and worst case is **$O(n^2)$**, where n is the number of items. Insertion sort is less efficient than the other sorting algorithms like heap sort, quick sort, merge sort, etc.

Algorithm

Thesimplestepsofachievingtheinsertionsortarelistedasfollows-
Step1-Iftheelementisthefirstelement,assumethatitisalreadysorted.Return 1.
Step2 - Pick the next element, and store it separately in a **key.**
Step3-Now,comparethe**key**withallelementsinthesortedarray.
Step4 -Iftheelement inthesortedarrayissmallerthanthecurrent element,thenmove tothenext element. Else, shift greater elements in the array towards the right.
Step5-Insertthevalue.
Step6-Repeatuntilthearrayissorted. Working
of Insertion sort Algorithm
Now,let'sseetheworkingoftheinsertionsortAlgorithm.
Tounderstandtheworkingoftheinsertionsortalgorithm,let'stakeanunsortedarray.Itwillbe easier to understand the insertion sort via an example.
Lettheelementsofarrayare-

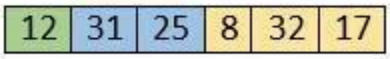


Initially,thefirsttwoelementsarecomparedininsertionsort.

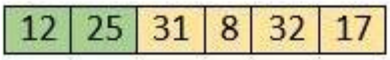


Here, 31 is greater than 12. That means both elements are already in ascending order. So, for now, 12 is stored in a sorted sub-array.

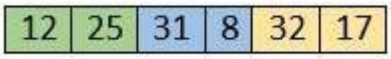
Now,movetothenexttwoelementsandcompare them.



Here,25issmallerthan31.So,31isnotatcorrectposition.Now,swap31with25.Alongwith swapping, insertion sort will also check it with all elements in the sorted array.
For now, the sorted array has only one element, i.e. 12. So, 25 is greater than 12. Hence, the sorted array remains sorted after swapping.

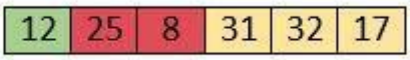


Now, two elements in the sorted array are 12 and 25. Move forward to the next elements that are31 and 8.

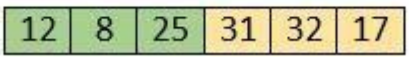


Both31and8are notsorted.So,swap them.

Afterswapping,elements25and8areunsorted.

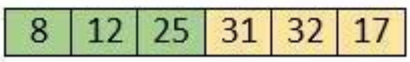


So,swapthem.

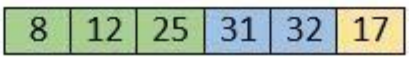


Now,elements12and8areunsorted.

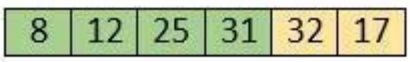
So,swapthem too.



Now, the sorted array has three items that are 8, 12 and 25. Move to the next items that are 31 and 32.



Hence,theyarealreadysorted.Now,thesortedarrayincludes8,12,25and31.



Movetothenextelementsthatare32and17.

17issmallerthan32.So,swap them.

Swappingmakes31and17unsorted.So,swapthemtoo.

Now,swappingmakes25and17unsorted.So,performswappingagain.

Now,thearrayiscompletelysorted.

Insertion sort complexity

1. TimeComplexity

Case	**TimeComplexity**
BestCase	O(n)
AverageCase	$O(n^2)$
WorstCase	$O(n^2)$

- **Best Case Complexity -** It occurs when there is no sorting required, i.e. the array is already sorted. The best-case time complexity of insertion sort is **O(n)**.
- **Average Case Complexity -** It occurs when the array elements are in jumbled order that is not properly ascending and not properly descending. The average case time complexity of insertion sort is **$O(n^2)$**.
- **Worst Case Complexity -** It occurs when the array elements are required to be sorted in reverse order. That means suppose you have to sort the array elements in ascending order, butitselementsareindescendingorder.Theworst-casetimecomplexityofinsertionsort is **$O(n^2)$**.

2. Space Complexity

SpaceComplexity	O(1)
Stable	YES

- ThespacecomplexityofinsertionsortisO(1).Itisbecause,ininsertionsort,anextra variable is required for swapping.

Implementationofinsertionsort

Program: WriteaprogramtoimplementinsertionsortinClanguage.

```
#include<stdio.h>

voidinsert(inta[],intn)/*functiontosortanaaywithinsertionsort*/
{
    inti,j, temp;
```

```
    for (i=1;i<n;i++){
       temp= a[i];
       j =i- 1;

       while(j>=0 && temp<= a[j])/* Move the elements greater than temp to one position a head from their current position*/
       {
          a[j+1]= a[j];
          j=j-1;
       }
       a[j+1]= temp;
    }
}

voidprintArr(inta[],intn)/*functiontoprintthearray*/
{
    inti;
    for(i=0;i <n;i++)
       printf("%d", a[i]);
}

intmain()
{
    inta[]={12,31,25,8,32,17 };
    intn=sizeof(a)/sizeof(a[0]);
    printf("Beforesortingarrayelementsare- \n");
    printArr(a,n);
    insert(a,n);
    printf("\nAftersortingarrayelementsare-\n");
    printArr(a,n);

    return0;
}
```

Output:

```
Before sorting array elements are -
12 31 25 8 32 17
After sorting array elements are -
8 12 17 25 31 32
```

HeapSort

HeapSortAlgorithm

Heap sort processes the elements by creating the min-heap or max-heap using the elements of the given array. Min-heap or max-heap represents the ordering of array in which the root element represents the minimum or maximum element of the array.

Heapsortbasicallyrecursivelyperformstwomainoperations-

- BuildaheapH,usingtheelementsof array.
- Repeatedlydeletetherootelementoftheheapformedin1st phase.

Aheapisacompletebinary tree,andthe binary treeisatreeinwhichthe nodecanhave theutmost two children. A complete binary tree is a binary tree in which all the levels except the last level, i.e., leaf node, should be completely filled, and all the nodes should be left-justified.

Heapsort is a popular and efficient sorting algorithm. The concept of heap sort is to eliminate the elements one by one from the heap part of the list, and then insert them into the sorted part of the list.

Algorithm

1. HeapSort(arr)
2. BuildMaxHeap(arr)
3. fori=length(arr)to2
4. swaparr[1]witharr[i]
5. heap_size[arr]=heap_size[arr]?1
6. MaxHeapify(arr,1)
7. End

BuildMaxHeap(arr)

1. BuildMaxHeap(arr)
2. heap_size(arr)=length(arr)
3. fori=length(arr)/2to1
4. MaxHeapify(arr,i)
5. End

MaxHeapify(arr,i)

1. MaxHeapify(arr,i)
2. L= left(i)
3. R=right(i)
4. ifL?heap_size[arr]andarr[L]**>**arr[i]
5. largest=L
6. else
7. largest=i
8. ifR?heap_size[arr]andarr[R]**>**arr[largest]

9. largest=R
10. iflargest!=i
11. swaparr[i]witharr[largest]
12. MaxHeapify(arr,largest)
13. End

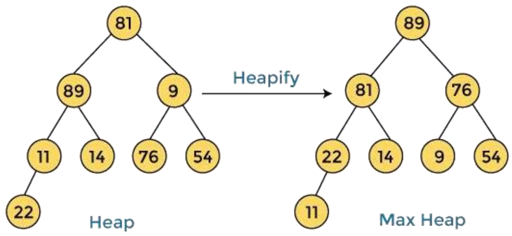
WorkingofHeapsortAlgorithm

In heap sort, basically, there are two phases involved in the sorting of elements. By using the heap sort algorithm, they are as follows -

- Thefirststepincludesthecreationofaheapbyadjustingtheelementsofthearray.
- After the creation of heap, now remove the root element of the heap repeatedly by shifting it to the end of the array, and then store the heap structure with the remaining elements.

81	89	9	11	14	76	54	22

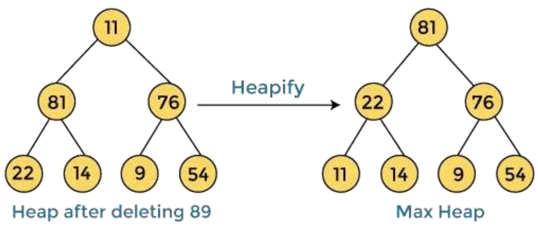
First,wehavetoconstructaheapfromthegivenarrayandconvertitintomaxheap.



Afterconvertingthegivenheapintomaxheap,thearrayelementsare-

89	81	76	22	14	9	54	11

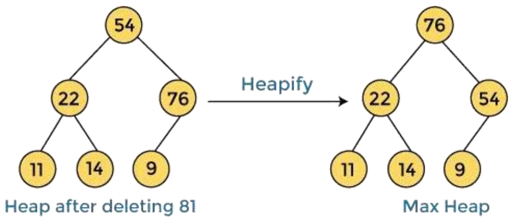
Next, we have to delete the root element **(89)** from the max heap. To delete this node, we have to swap it with the last node, i.e. **(11).** After deleting the root element, we again have to heapify it to convert it into max heap.



After swapping the array element **89** with **11,** and converting the heap into max-heap, the elements of array are -

81	22	76	11	14	9	54	89

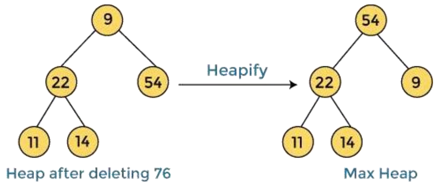
In the next step, again, we have to delete the root element **(81)** from the max heap. To delete this node, wehave to swapit with thelast node, i.e. **(54).** After deletingthe rootelement, we again have to heapify it to convert it into max heap.



After swapping the array element **81** with **54** and converting the heap into max-heap, the elements of array are -

76	22	54	11	14	9	81	89

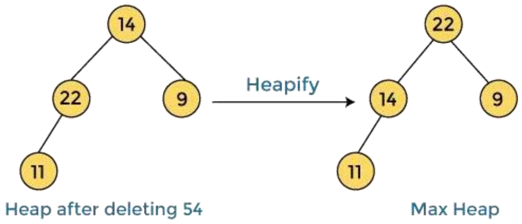
In the next step, we have to delete the root element **(76)** from the max heap again. To delete this node, we have to swap it with the last node, i.e. **(9).** After deleting the root element, we again have to heapify it to convert it into max heap.



Afterswapping the array element **76**with **9** and converting the heap into max-heap,the elementsof array are -

54	22	9	11	14	76	81	89

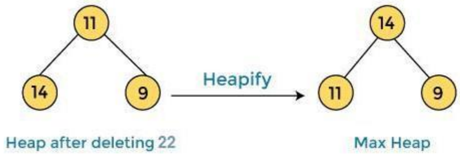
In the next step, again we have to delete the root element **(54)** from the max heap. To delete this node, wehave to swapit with thelast node, i.e. **(14).** After deletingthe rootelement, we again have to heapify it to convert it into max heap.



After swapping the array element **54** with **14** and converting the heap into max-heap, the elements of array are -

22	14	9	11	54	76	81	89

In the next step, again we have to delete the root element **(22)** from the max heap. To delete this node, wehave to swapit with thelast node, i.e. **(11).** After deletingthe rootelement, we again have to heapify it to convert it into max heap.



After swapping the array element **22** with **11** and converting the heap into max-heap, the elements of array are -

14	11	9	22	54	76	81	89

In the next step, again we have to delete the root element **(14)** from the max heap. To delete this node, we have to swap it with the last node, i.e. **(9).** After deleting the root element, we again have to heapify it to convert it into max heap.

Afterswapping the array element **14**with **9** and converting the heap into max-heap,the elementsof array are -

11	9	14	22	54	76	81	89

In the next step, again we have to delete the root element **(11)** from the max heap. To delete this node, we have to swap it with the last node, i.e. **(9).** After deleting the root element, we again have to heapify it to convert it into max heap.

Afterswappingthearrayelement**11**with**9,**theelementsofarrayare-

9	11	14	22	54	76	81	89

Now,heaphasonlyoneelementleft.Afterdeletingit,heapwillbeempty.

Aftercompletionofsorting,thearrayelementsare-

9	11	14	22	54	76	81	89

TimecomplexityofHeapsortinthebestcase,averagecase,andworst case

1. TimeComplexity

Case	**TimeComplexity**

BestCase	O(nlogn)
AverageCase	O(nlogn)
WorstCase	O(nlogn)

- **Best Case Complexity -** It occurs when there is no sorting required, i.e. the array is already sorted. The best-case time complexity of heap sort is **O(n logn).**
- **Average Case Complexity -** It occurs when the array elements are in jumbled order that is not properly ascending and not properly descending. The average case time complexity of heap sort is **O(n log n).**
- **Worst Case Complexity -** It occurs when the array elements are required to be sorted in reverse order. That means suppose you have to sort the array elements in ascending order, but itselements are in descending order. Theworst-case time complexityofheap sortis **O(n log n).**

The time complexity of heap sort is **O(n logn)** in all three cases (best case, average case, and worstcase). The height of a complete binary tree having n elements is **logn.**

2. Space Complexity

SpaceComplexity	O(1)
Stable	N0

- ThespacecomplexityofHeapsortisO(1).

Implementation of Heapsort

Program: WriteaprogramtoimplementheapsortinC language.

```
#include<stdio.h>
/*functiontoheapifyasubtree.Here'i'isthe
indexofrootnodeinarraya[],and'n'isthesizeofheap.*/
voidheapify(inta[],intn,inti)
{
  intlargest=i;//Initializelargestas root
  int left= 2*i+1;//leftchild
  int right =2* i+2;//rightchild
  //Ifleftchildislargerthan root
  if(left<n&&a[left]>a[largest])
    largest=left;
```

```
    //Ifrightchildislargerthanroot
    if(right<n&&a[right]>a[largest])
        largest=right;
    //Ifrootisnot largest
    if(largest!=i){
        //swapa[i]witha[largest]
        inttemp=a[i];
        a[i]= a[largest];
        a[largest]=temp;
       heapify(a,n,largest);
    }
}
/*Functiontoimplementtheheapsort*/
voidheapSort(inta[],intn)
{
    for(inti=n/2-1;i>=0;i--)
        heapify(a,n,i);
    //Onebyoneextract anelementfromheap
    for(inti=n-1;i>=0;i--) {
        /*Movecurrentrootelementtoend*/
        //swapa[0]witha[i]
        inttemp=a[0];
        a[0]= a[i];
        a[i]=temp;

        heapify(a,i,0);
    }
}
/*functiontoprintthearrayelements*/
voidprintArr(intarr[],intn)
{
```

```
    for(inti=0;i<n;++i)
    {
        printf("%d",arr[i]);
        printf("");
    }

}
int main()
{
    inta[]={48,10,23,43,28,26,1};
    intn=sizeof(a)/ sizeof(a[0]);
    printf("Beforesortingarrayelementsare- \n");
    printArr(a,n);
    heapSort(a,n);
    printf("\nAftersortingarrayelementsare-\n");
    printArr(a,n);
    return0;
}
```

Output

```
Before sorting array elements are -
48 10 23 43 28 26 1
After sorting array elements are -
1 10 23 26 28 43 48
```

UNIT2-GRAPHS:basics,representation, traversals, and application

Basicconcepts

Definition

AgraphG(V,E) isanon-lineardatastructurethat consistsofnode andedge pairsofobjectsconnectedby links.

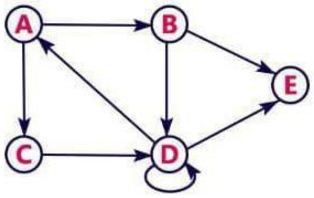
Thereare2typesofgraphs:

- Directed
- Undirected

Directedgraph

A graph with only directed edgesissaid tobe adirected graph. Example

The following directed graph has5 verticesand8 edges. This graphG canbedefinedasG=(V,E),whereV={A,B,C,D,E}andE={(A,B),(A,C)(B,E),(B,D),(D,A),(D,E),(C,D),(D,D)}.

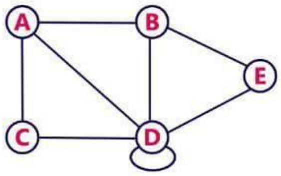


DirectedGraph

Undirectedgraph

Agraphwithonlyundirectededgesissaidtobeanundirectedgraph. Example

Thefollowingisanundirectedgraph.



UndirectedGraph

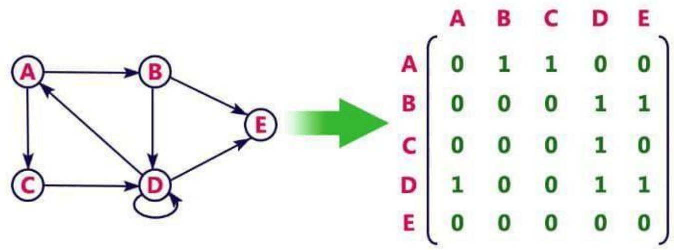
RepresentationofGraphs

Graph data structure is represented using the following representations.

1. AdjacencyMatrix
2. AdjacencyList

AdjacencyMatrix

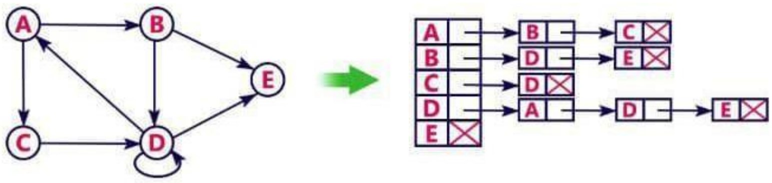
- Inthisrepresentation,the graph canbe representedusing a matrix of size n x n, where nisthe number of vertices.
- Thismatrixisfilledwitheither1’sor0’s.
- Here,1representsthatthere isanedgefromrowvertexto columnvertex,and0representsthatthereisnoedgefromrow vertextocolumnvertex.



Directedgraphrepresentation

Adjacencylist

- In this representation, every vertex of the graph contains a listofitsadjacent vertices.
- Ifthegraphisnotdense,i.e.,thenumberofedgesisless, thenit isefficient to represent thegraphthrough the adjacency list.



AdjacencyList

Graphtraversals

- Graph traversalisa technique used to search for a vertexina graph.It isalso used to decide the order of vertices to be visited inthe search process.
- A graph traversal finds the edges tobe usedinthe search process without creating loops. Thismeans that, with graph traversal,we canvisit allthe vertices of the graph without getting into a looping path. There are two graph traversal techniques:

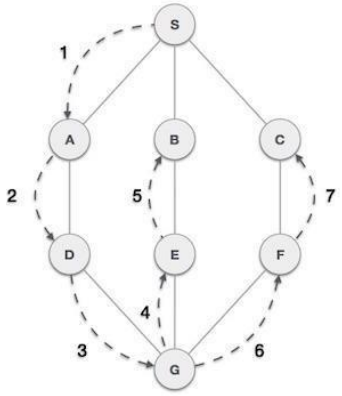
1. DFS(DepthFirstSearch)
2. BFS(Breadth-FirstSearch)

Applicationsofgraphs

1. Social network graphs:To tweet or not to tweet. Graphs that representwhoknowswhom,whocommunicateswithwhom,who influenceswhom,orotherrelationshipsinsocialstructures.An exampleisthe twitter graph ofwho followswhom.
2. Graphs in epidemiology: Vertices represent individuals and directededgestoviewthetransferofaninfectiousdisease fromoneindividualtoanother.Analyzingsuchgraphshasbecome animportantcomponentinunderstandingandcontrollingthe spread of diseases.
3. Protein-protein interactions graphs: Vertices represent proteins andedges represent interactionsbetweenthem that carry out some biological function in the cell.These graphscanbeused to,forexample,studymolecularpathway—chainsofmolecular interactions ina cellular process.
4. Network packet traffic graphs: Vertices are IP (Internet protocol)addressesandedgesarethepacketsthatflowbetween them.Such graphs are used for analyzingnetwork security, studying the spread of worms,and trackingcriminalor non-criminal activity.
5. Neuralnetworks:Verticesrepresentneuronsandedgesarethe synapsesbetweenthem.Neuralnetworksareusedtounderstand howourbrainworksandhowconnectionschangewhenwelearn. Thehumanbrainhasabout1011neuronsandcloseto1015 synapses.

DFS–DepthFirstSearch

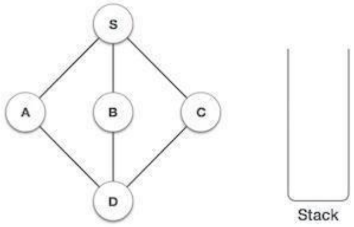
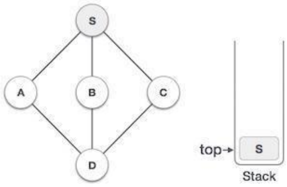
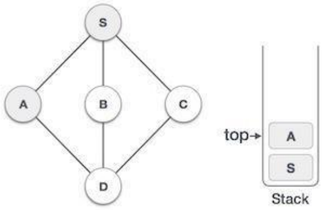
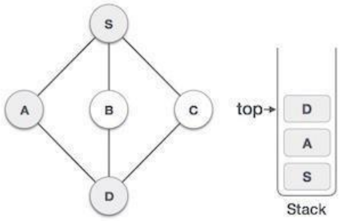
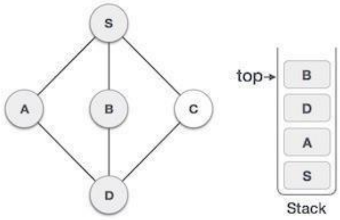
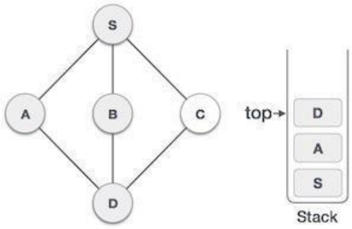
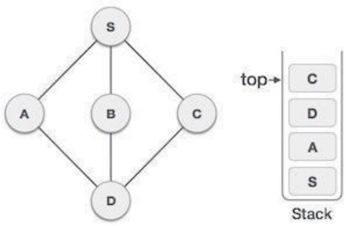
DepthFirstSearch(DFS)algorithmtraversesagraph inadepthwardmotionandusesastackto remember to get the next vertex to start a search, when a dead end occurs in any iteration.



As inthe examplegivenabove, DFSalgorithmtraversesfromStoA toDtoG toE toBfirst,thentoF and lastly to C. It employs the following rules.

- **Rule1**–Visittheadjacentunvisitedvertex.Markitasvisited.Displayit.Pushitinastack.
- **Rule2**–Ifnoadjacent vertexisfound,popup avertexfromthestack.(It willpopupallthe vertices from the stack, which do not have adjacent vertices.)
- **Rule3**–RepeatRule1andRule2untilthestackisempty.

Step	Traversal	Description

<table>
<tr><td>1</td><td>
</td><td>Initializethestack.</td></tr>
<tr><td>2</td><td>
</td><td>Mark S as visited and put it onto the stack. Explore any unvisited adjacent nodefromS.Wehavethreenodesand we can pick any of them. For this example, we shall take the node in an alphabetical order.</td></tr>
<tr><td>3</td><td>
</td><td>MarkAas visitedandput itontothe stack.Exploreanyunvisitedadjacent node from A. Both S and D are adjacent to A but we are concerned for unvisited nodes only.</td></tr>
<tr><td>4</td><td>
</td><td>VisitDandmarkitasvisitedandput onto the stack. Here, we have B and C nodes, which are adjacenttoDandbothareunvisited. However,weshallagainchooseinan alphabetical order.</td></tr>
<tr><td>5</td><td>
</td><td>We choose B, mark it as visited and put onto the stack. Here B does not haveanyunvisitedadjacentnode.So, we pop B from the stack.</td></tr>
<tr><td>6</td><td>
</td><td>Wecheckthestacktopforreturnto thepreviousnodeandcheckifithas any unvisited nodes. Here, we findDtobe onthetopofthestack.</td></tr>
<tr><td>7</td><td>
</td><td>Onlyunvisitedadjacentnode is fromDisCnow.SowevisitC,markit as visited and put it onto the stack.</td></tr>
</table>

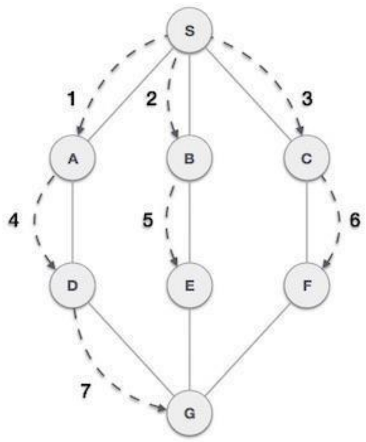
```
DFS(G, u)
   u.visited=true
   foreachv∈G.Adj[u]
      ifv.visited==false
         DFS(G,v)

init(){
   For each u ∈ G
      u.visited=false
    Foreachu∈G DFS(G,
      u)
}
```

ApplicationofDFSAlgorithm

1. Forfindingthepath
2. Totestifthegraphisbipartite
3. Forfindingthestronglyconnectedcomponentsofa graph
4. Fordetectingcyclesinagraph

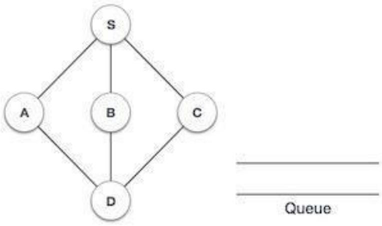
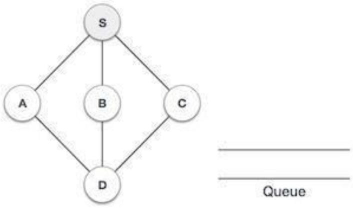
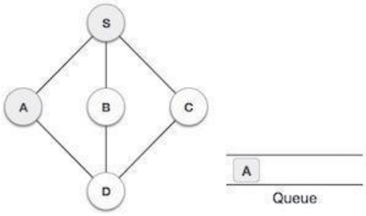
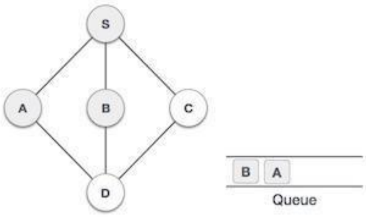
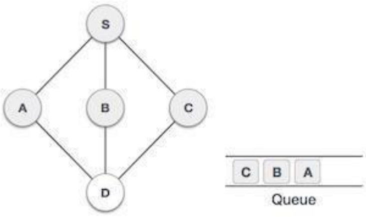
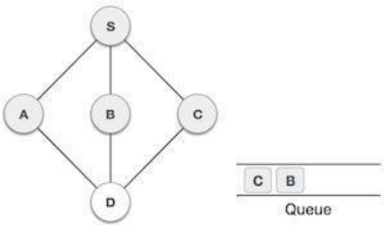
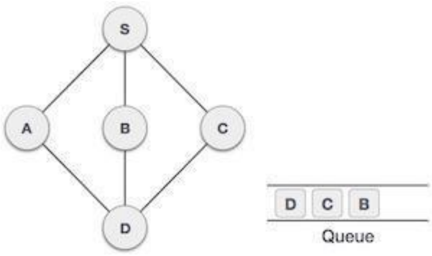
BreadthFirstSearch

BreadthFirstSearch(BFS)algorithmtraversesagraph inabreadthwardmotionandusesaqueue to remember to get the next vertex to start a search, when a dead end occurs in any iteration.



Asinthe examplegivenabove,BFSalgorithmtraversesfromAtoBtoEtoFfirst thentoCandG lastly to D. It employs the following rules.

- Rule1−Visittheadjacentunvisitedvertex.Markitasvisited.Displayit.Insertitinaqueue.
- Rule2−Ifnoadjacentvertexisfound,removethefirstvertexfromthequeue.
- Rule3− RepeatRule1andRule2untilthequeueisempty.

Step	Traversal	Description
1		Initializethequeue.
2		WestartfromvisitingS(starting node), and mark it as visited.
3		We then see an unvisited adjacent nodefromS.Inthisexample,wehave three nodes but alphabetically we choose A, mark it as visited and enqueue it.
4		Next, the unvisited adjacent node fromSisB.Wemarkitasvisitedand enqueue it.
5		Next, the unvisited adjacent node fromSisC.Wemarkitasvisitedand enqueue it.
6		Now, S is left with no unvisited adjacentnodes.So,wedequeueand find A.
7		**From A we have D as unvisitedadjacentnode.We mark it as visited and enqueue it.**

BFSpseudocode

createaqueueQ

markvasvisitedandputvintoQ while

Q is non-empty

  removetheheaduofQ

  markandenqueueall(unvisited)neighboursofu

BFSAlgorithmComplexity

ThetimecomplexityoftheBFSalgorithmis representedintheformofO(V +E),whereVis the number of nodes and E is the number of edges.

ThespacecomplexityofthealgorithmisO(V).

BFSAlgorithmApplications

1. Tobuildindexbysearchindex
2. ForGPSnavigation
3. Pathfindingalgorithms
4. InFord-Fulkersonalgorithmtofindmaximumflowinanetwork
5. Cycledetectioninanundirectedgraph
6. Inminimumspanningtree

Connectedgraph,StronglyconnectedandBi-Connectivity

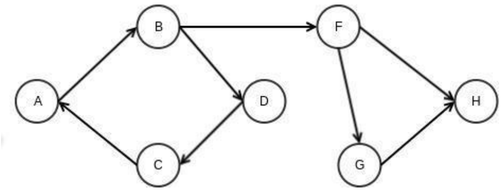
Connected Graph Component

Aconnectedcomponentorsimplycomponent ofanundirectedgraphisasubgraphinwhicheach pair of nodes is connected with each other via a path.

```
Component_Count = 0;
for each vertex k ∈ V do
|  Visited[k] = False;
end
for each vertex k ∈ V do
|  if Visited[k] == False then
|  |  DFS(V,k);
|  |  Component_Count = Component_Count + 1;
|  end
end
Print Component_Count;
Procedure DFS(V,k)
Visited[k] = True;
for each vertex p ∈ V.Adj[k] do
|  if Visited[p] == False then
|  |  DFS(V,p);
|  end
end
```

StronglyConnectedGraph

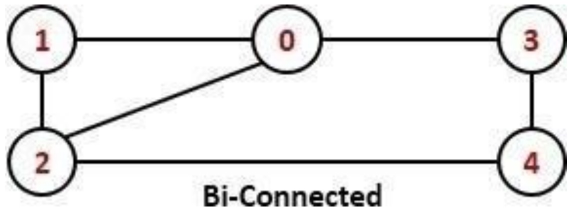
The **Kosaraju algorithm** is a DFS based algorithm used to find Strongly Connected Components**(SCC)**inagraph.It isbasedontheideathatifoneisabletoreachavertexvstarting fromvertex*u*, thenoneshouldbe abletoreachvertex*u*startingfromvertex*v*and ifsuchis thecase, one can say that vertices *u* and *v* are **strongly connected** - they are in a strongly connected sub- graph.




```
stackSTACK
voidDFS(intsource){
   visited[s]=true
   forallneighboursXofsourcethatarenotvisited:
      DFS(X)
   STACK.push(source)
}

CLEARADJACENCY_LIST
foralledgese:
   first = one end point of e
   second=otherendpointofe
   ADJACENCY_LIST[second].push(first)

whileSTACKisnotempty:
   source=STACK.top()
   STACK.pop()
   ifsourceisvisited:
      continue
   else :
      DFS(source)
```

BiConnectivityGraph

An undirected graph is said to be a biconnected graph, if there are two vertex-disjoint paths betweenanytwoverticesarepresent.Inotherwords,wecansay thatthereisacyclebetweenany two vertices.



WecansaythatagraphGisabi-connectedgraphifitisconnected,andthereare noarticulation points or cut vertex are present in the graph.

Tosolvethisproblem,wewillusetheDFStraversal.UsingDFS,wewilltrytofindifthereisany articulationpointispresentornot.WealsocheckwhetherallverticesarevisitedbytheDFSornot, if not we can say that the graph is not connected.

PseudocodeforBi connectivity

```
isArticulation(start,visited,disc,low,parent)
Begin
  time := 0      //thevalueoftimewillnotbeinitializedfornextfunctioncalls
  dfsChild := 0
  markstartasvisited
  setdisc[start]:=time+1andlow[start]:=time+1 time
  := time + 1
  forallvertexvinthegraph G,do
    ifthereisanedgebetween(start,v),then if v
      is visited, then
        increasedfsChild
        parent[v]:=start
        ifisArticulation(v,visited,disc,low,parent)istrue,then
          return ture
        low[start]:=minimumoflow[start]andlow[v] if
        parent[start] is ϕ AND dfsChild > 1, then
          returntrue
        ifparent[start]isϕANDlow[v]>=disc[start],then return
          true
      else if v is not the parent of start,
        thenlow[start]:=minimumoflow[start]anddisc[
        v]
  donereturn
  false
End
isBiconnected(graph)
Begin
  initiallysetallverticesareunvisitedandparentofeachverticesareϕ if
  isArticulation(0, visited, disc, low, parent) = true, then
    returnfalse
  foreachnodeiofthegraph,do if i
    is not visited, then
      returnfalse
  done
  returntrue
End
```

MinimumSpanningTree

A Spanning Tree is a tree which have V vertices and V-1 edges. All nodes in a spanning tree are reachable from each other.

A Minimum Spanning Tree(MST) or minimum weight spanning tree for a weighted, connected, undirected graph is a spanning tree having a weight less than or equal to the weight of every other possible spanning tree. The weight of a spanning tree is the sum of weights given to each edge of the spanning tree. In short out of all spanning trees of a given graph, the spanning tree having minimum weight is MST.

AlgorithmsforfindingMinimumSpanning Tree(MST):-

1. Prim'sAlgorithm

2. Kruskal’sAlgorithm

Prim’sAlgorithm

Prim'salgorithmisa[minimumspanningtree]algorithmthattakesagraphasinputandfindsthe subset of the edges of that graph which

- formatreethatincludeseveryvertex
- hastheminimumsumofweightsamongallthetreesthatcanbeformedfromthegraph

---

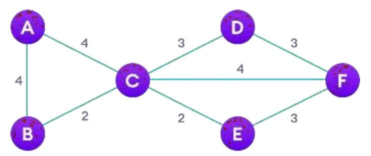
HowPrim'salgorithmworks

It falls under a class of algorithms called [greedy algorithms]that find the local optimum in the hopes of finding a global optimum.

Westart fromonevertexandkeepaddingedgeswiththelowestweight untilwereachourgoal. The steps for implementing Prim's algorithm are as follows:

1. Initializetheminimumspanningtreewithavertexchosenat random.
2. Find all the edges that connect the tree to new vertices, find the minimum and add it to the tree
3. Keeprepeatingstep2untilwegetaminimumspanningtree

ExampleofPrim'salgorithm

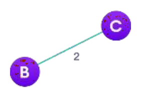


Step: 1

Startwithaweightedgraph

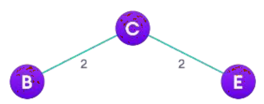
Step: 2

Chooseavertex



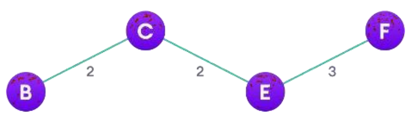
Step: 3

Choosetheshortestedgefromthisvertexandaddit



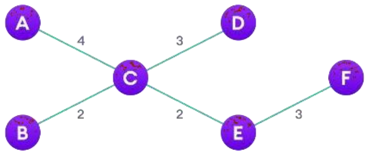
Step: 4

Choosethenearestvertexnotyetinthesolution



Step: 5

Choosethenearestedgenotyetinthesolution,iftherearemultiplechoices,chooseoneatrandom



Step: 6

Prim'sAlgorithm pseudocode

The pseudocode for prim's algorithm shows how we create two sets of vertices U and V-U. U contains the list of vertices that have been visited and V-U the list of vertices that haven't. One by one, we move vertices from set V-U to set U by connecting the least weight edge.

T=∅;
U={1};
while(U≠V)
   let (u,v)be thelowestcostedgesuchthatu∈ Uandv∈ V- U;
   T=T∪ {(u,v)}
   U =U∪ {v}

Prim'sAlgorithmComplexity

ThetimecomplexityofPrim'salgorithmisO(ElogV).

KruskalAlgorithm

Kruskal's algorithm is a minimum spanning treealgorithm that takes a graph as input and finds the subset of the edges of that graph which

- formatreethatincludeseveryvertex
- hastheminimumsumofweightsamongallthetreesthatcanbeformedfromthegraph

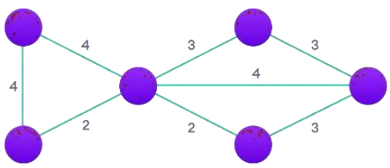
HowKruskal'salgorithmworks

It falls under a class of algorithms called greedy algorithmsthat find the local optimum in the hopes of finding a global optimum.

Westart fromtheedgeswiththe lowestweightandkeepaddingedgesuntilwereachourgoal. The steps for implementing Kruskal's algorithm are as follows:

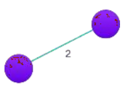
1. Sortalltheedgesfromlowweighttohigh
2. Taketheedgewiththelowestweightandaddittothespanningtree.Ifaddingtheedge created a cycle, then reject this edge.
3. Keepaddingedgesuntilwereachallvertices.

ExampleofKruskal'salgorithm



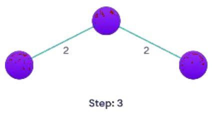
Step: 1

Startwithaweightedgraph

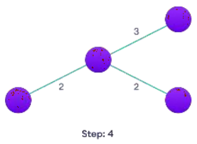


Step: 2

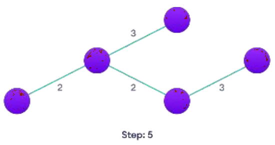
Choosetheedgewiththeleastweight,iftherearemorethan1,chooseanyone



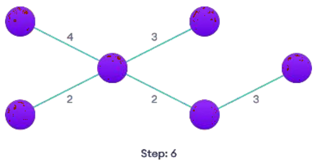
Choosethenextshortestedgeandaddit



Choosethenextshortestedgethatdoesn'tcreateacycleandaddit



Choosethenextshortestedgethatdoesn'tcreateacycleandaddit



Repeatuntilyouhaveaspanning tree

KruskalAlgorithmPseudocode

```
KRUSKAL(G):
A =∅
Foreachvertexv∈G.V:
   MAKE-SET(v)
Foreachedge(u,v)∈G.Eorderedbyincreasingorderbyweight(u,v):
   ifFIND-SET(u)≠FIND-SET(v):
   A=AU{(u,v)}
   UNION(u, v)
returnA
```

ShortestPathAlgorithm

The shortest path problem is about **finding a path between vertices in a graph such that the totalsum of the edges weights is minimum**.

AlgorithmforShortestPath

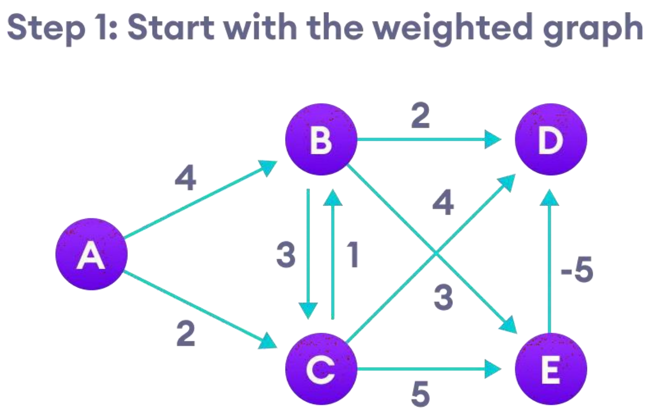
1. BellmanAlgorithm
2. DijkstraAlgorithm
3. FloydWarshallAlgorithm

BellmanAlgorithm

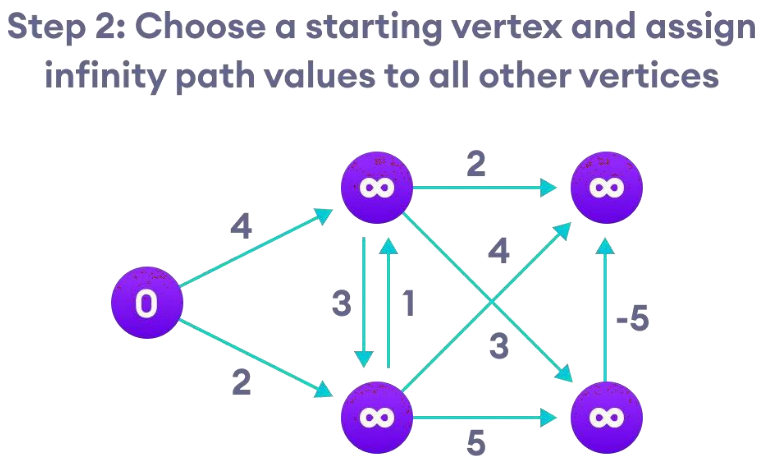
BellmanFordalgorithmhelpsusfindtheshortestpathfromavertextoallotherverticesofa weighted graph. ItissimilartoDijkstra'salgorithmbutitcanworkwithgraphsinwhichedgescanhavenegative weights.

HowBellmanFord'salgorithmworks

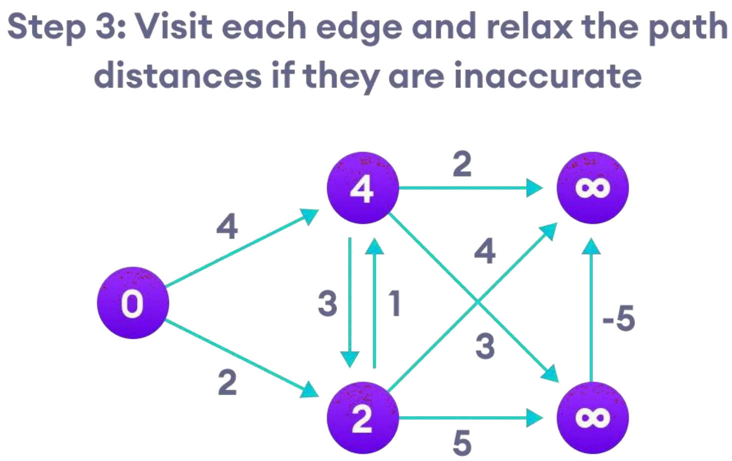
Bellman Ford algorithm works by overestimating the length of the path from the starting vertex toall other vertices. Then it iteratively relaxes those estimates by finding new paths that are shorter than the previously overestimated paths.
Bydoingthisrepeatedlyforallvertices,wecanguaranteethattheresultisoptimized.



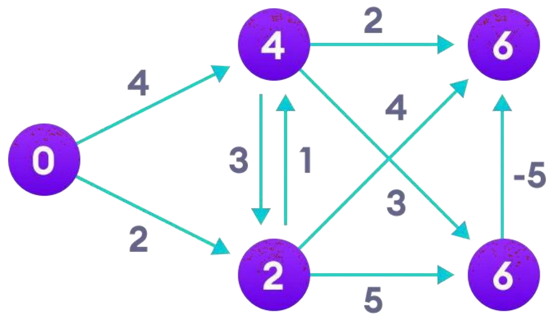
Step-1forBellmanFord'salgorithm



Step-2forBellmanFord'salgorithm

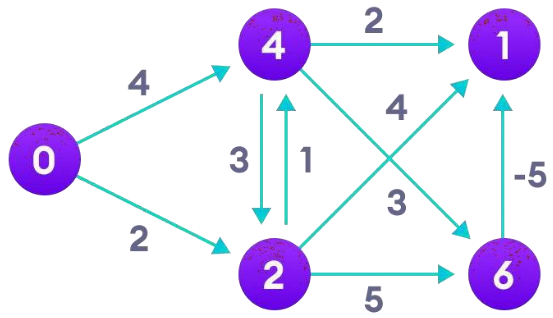


Step 4: We need to do this V times because in the worst case, a vertex's path length might need to be readjusted V times



Step-4forBellmanFord'salgorithm

Step 5: Notice how the vertex at the top right corner had its path length adjusted



Step-5forBellmanFord'salgorithm

Step 6: After all the vertices have their path lengths, we check if a negative cycle is present

	B	C	D	E
0	∞	∞	∞	∞
0	4	2	∞	∞
0	3	2	6	6
0	3	2	1	6
0	3	2	1	6

Step-6forBellmanFord'salgorithm

BellmanFordPseudocode

Weneedtomaintainthepathdistanceofeveryvertex.Wecanstorethatinanarrayofsizev, where v is the number of vertices.

We also want to be able to get the shortest path, not only know the length of the shortest path. For this, we map each vertex to the vertex that last updated its path length.

Oncethe algorithmisover,wecanbacktrack fromthe destinationvertextothesourcevertextofind the path.

```
functionbellmanFord(G,S) for
 each vertex V in G
 distance[V] <- infinite
   previous[V]<-NULL
 distance[S] <- 0

 for each vertex V in
  Gforeachedge(U,V)inG
   tempDistance<-distance[U]+edge_weight(U,V) if
   tempDistance < distance[V]
    distance[V]<-tempDistance
    previous[V] <- U

 foreachedge (U,V)inG
  Ifdistance[U]+edge_weight(U,V)<distance[V}
   Error:NegativeCycleExists

return distance[], previous[]
```

Bellman Ford's Complexity

Time Complexity

BestCaseComplexity	O(E)
AverageCaseComplexity	O(VE)
WorstCaseComplexity	O(VE)

DijkstraAlgorithm

Dijkstra'salgorithmallowsustofindtheshortestpathbetweenanytwoverticesofa graph. Itdiffersfromtheminimumspanningtreebecausetheshortestdistancebetweentwovertices might not include all the vertices of the graph.

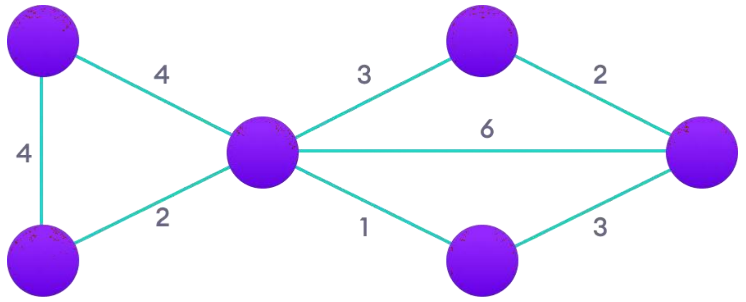
HowDijkstra'sAlgorithmworks

Dijkstra's Algorithm works on the basis that any subpath B -> D of the shortest path A -> D between vertices A and D is also the shortest path between vertices B and D.

Eachsubpathistheshortest path

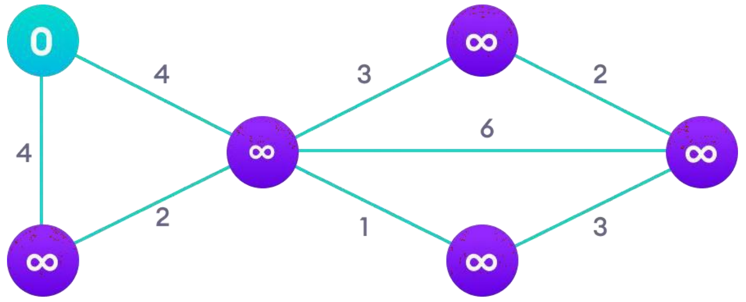
Djikstra used this property in the opposite direction i.e we overestimate the distance of each vertex from the starting vertex. Then we visit each node and its neighbors to find the shortest subpath to those neighbors.
The algorithm uses a greedy approach in the sense that we find the next best solution hoping that the end result is the best solution for the whole problem.

ExampleofDijkstra'salgorithm
Itiseasiertostartwithanexampleandthenthinkaboutthealgorithm.



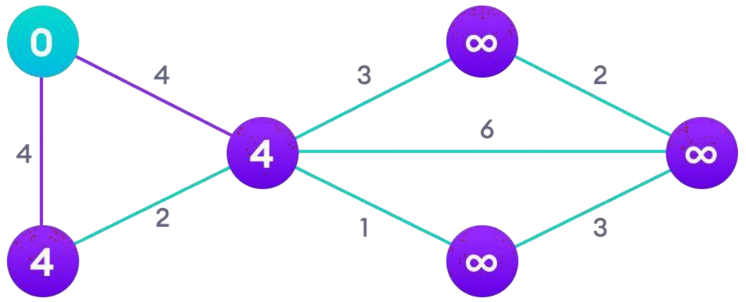
Step: 1

Startwithaweightedgraph



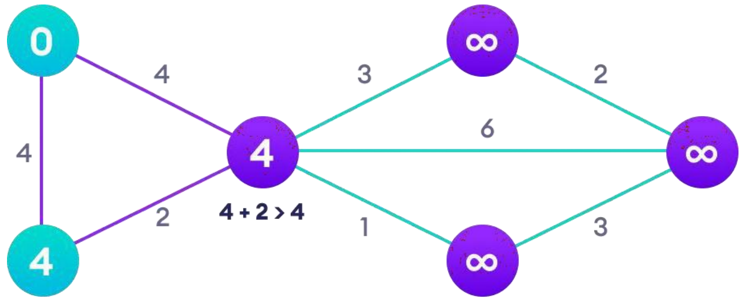
Step: 2

Chooseastartingvertexandassigninfinitypathvaluestoallotherdevices



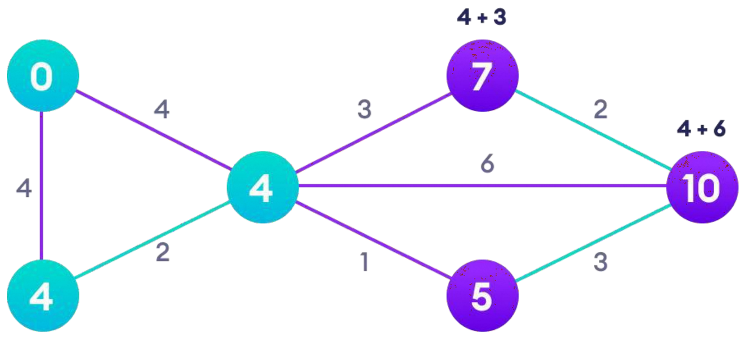
Step: 3

Gotoeachvertexandupdateitspath length



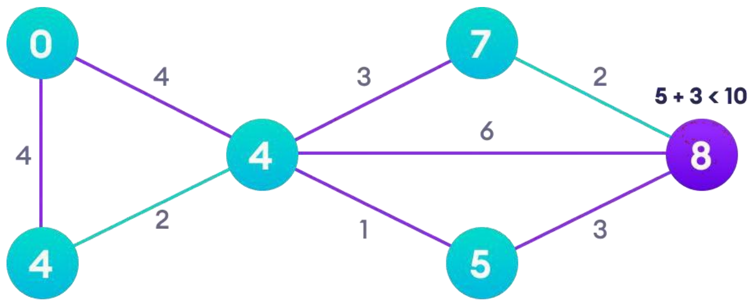
Step: 4

Ifthepathlengthoftheadjacentvertexislesserthannewpathlength,don'tupdateit



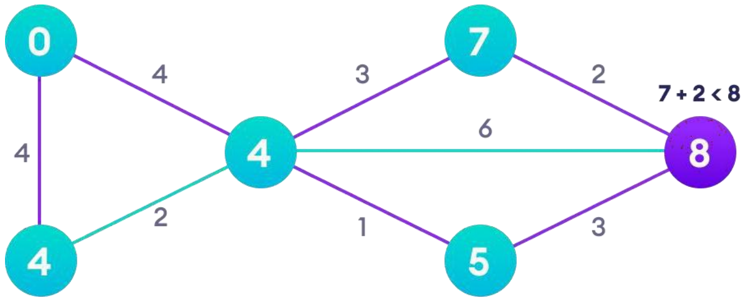
Step: 5

Avoidupdatingpathlengthsofalreadyvisitedvertices



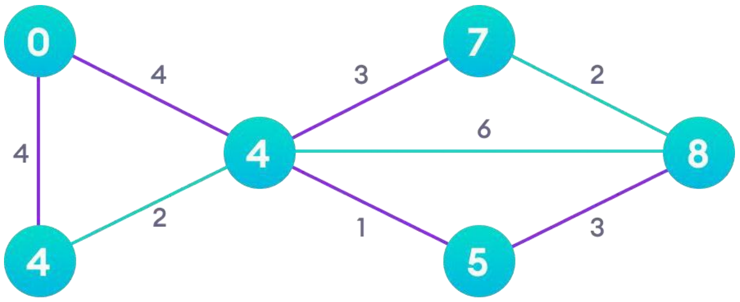
Step: 6

Aftereachiteration,wepicktheunvisitedvertexwiththeleastpathlength.Sowechoose5before7



Step: 7

Noticehowtherightmostvertexhasitspathlengthupdatedtwice



Step: 8

Repeatuntilalltheverticeshavebeenvisited

Djikstra'salgorithmpseudocode

Weneedtomaintainthepathdistanceofeveryvertex.Wecanstorethatinanarrayofsizev, where v is the number of vertices.

We also want to be able to get the shortest path, not only know the length of the shortest path. For this, we map each vertex to the vertex that last updated its path length.

Oncethe algorithmisover,wecanbacktrack fromthe destinationvertextothesourcevertextofind the path.

Aminimumpriorityqueuecanbeusedtoefficiently receivethe vertexwithleastpathdistance. function dijkstra(G, S)

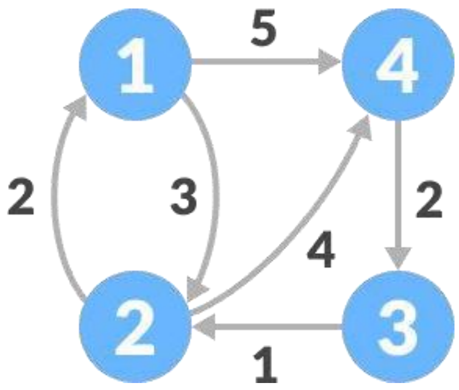
```
  for each vertex V in G
    distance[V]<-infinite
    previous[V] <- NULL
    IfV!=S,addVtoPriorityQueueQ
  distance[S] <- 0

  whileQISNOTEMPTY
    U<-ExtractMINfromQ
    foreachunvisitedneighbourVofU
      tempDistance<-distance[U]+edge_weight(U,V) if
      tempDistance < distance[V]
        distance[V]<-tempDistance
        previous[V] <- U
  returndistance[],previous[]
```

Dijkstra'sAlgorithmComplexity
TimeComplexity:O(ELogV)
where,EisthenumberofedgesandVisthenumberofvertices. Space
Complexity: O(V)

FloydWarshallAlgorithm

Floyd-Warshall Algorithm is an algorithm for finding the shortest path between all the pairs of vertices in a weighted graph. This algorithm works for both the directed and undirected weighted graphs. But, it does not work for the graphs with negative cycles (where the sum of the edges in a cycle is negative).
Aweightedgraphisagraph inwhicheachedgehasanumericalvalueassociatedwith it.
Floyd-Warhshall algorithm is also called as Floyd's algorithm, Roy-Floyd algorithm, Roy-Warshall algorithm, or WFI algorithm.
Thisalgorithmfollowsthedynamicprogrammingapproachtofindtheshortestpaths.

HowFloyd-WarshallAlgorithmWorks?
Letthegivengraphbe:



Initialgraph
Followthestepsbelowtofindtheshortestpathbetweenallthepairsof vertices.

1. CreateamatrixA^{0}ofdimensionn*nwherenisthenumberofvertices.Therowandthe column are indexed as i and j respectively. i and j are the vertices of the graph. EachcellA[i][j]isfilledwiththedistancefromtheithvertextothejthvertex.Ifthereisno path from ith vertex to jth vertex, the cell is left as infinity.

$$A^0 = \begin{array}{c c} & \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \end{array} & \begin{bmatrix} 0 & 3 & \infty & 5 \\ 2 & 0 & \infty & 4 \\ \infty & 1 & 0 & \infty \\ \infty & \infty & 2 & 0 \end{bmatrix} \end{array}$$

   Filleachcellwiththedistancebetweenithandjthvertex

2. Now, create a matrix A^1 using matrix A^0. The elements in the first column and the first roware left as they are. The remaining cells are filled in the following way. Letkbetheintermediatevertexintheshortestpathfromsourcetodestination.Inthis step, k is the first vertex. A[i][j] is filled with (A[i][k] + A[k][j]) if (A[i][j] > A[i][k] + A[k][j]). Thatis,ifthedirectdistancefromthesourcetothedestinationisgreaterthanthepath h the vertex k, then the cell is filled with A[i][k] + A[k][j].

Inthisstep,k isvertex1.Wecalculatethedistancefromsourcevertextodestination vertex through this vertex

$$A^1 = \begin{array}{cc} & \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \end{array} & \begin{bmatrix} 0 & 3 & \infty & 5 \\ 2 & 0 & & \\ \infty & & 0 & \\ \infty & & & 0 \end{bmatrix} \end{array} \longrightarrow \begin{array}{cc} & \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \end{array} & \begin{bmatrix} 0 & 3 & \infty & 5 \\ 2 & 0 & 9 & 4 \\ \infty & 1 & 0 & 8 \\ \infty & \infty & 2 & 0 \end{bmatrix} \end{array}$$

k. Calcula tethedistancefromthesourcevertextodestinationvertexthroughthisvertexk

Forexample:ForA^1[2,4],thedirectdistancefromvertex2to4is4andthesumofthe distancefromvertex2to4throughvertex(ie.fromvertex2 to1andfromvertex1to4)is7. Since4<7,A^0[2,4]isfilledwith4.

3. Similarly, A^2 is created using A^1. The elements in the second column and the second row are left as they are.
   Inthisstep,kisthesecond vertex(i.e.vertex2).Theremainingstepsarethesameasin **step**

$$A^2 = \begin{array}{cc} & \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \end{array} & \begin{bmatrix} 0 & 3 & & \\ 2 & 0 & 9 & 4 \\ & 1 & 0 & \\ & \infty & & 0 \end{bmatrix} \end{array} \longrightarrow \begin{array}{cc} & \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \end{array} & \begin{bmatrix} 0 & 3 & 9 & 5 \\ 2 & 0 & 9 & 4 \\ 3 & 1 & 0 & 5 \\ \infty & \infty & 2 & 0 \end{bmatrix} \end{array}$$

2. Calcula tethedistancefromthesourcevertextodestinationvertexthroughthisvertex2

4. Similarly,A^3andA^4isalsocreated.

$$A^3 = \begin{array}{cc} & \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \end{array} & \begin{bmatrix} 0 & & \infty & \\ & 0 & 9 & \\ \infty & 1 & 0 & 8 \\ & & 2 & 0 \end{bmatrix} \end{array} \longrightarrow \begin{array}{cc} & \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \end{array} & \begin{bmatrix} 0 & 3 & 9 & 5 \\ 2 & 0 & 9 & 4 \\ 3 & 1 & 0 & 5 \\ 5 & 3 & 2 & 0 \end{bmatrix} \end{array}$$

Calculat e the distance from the source vertex to destination vertex through this

$$A^4 = \begin{array}{c c} & \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \end{array} & \begin{bmatrix} 0 & & & 5 \\ & 0 & & 4 \\ & & 0 & 5 \\ 5 & 3 & 2 & 0 \end{bmatrix} \end{array} \rightarrow \begin{array}{c c} & \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \end{array} & \begin{bmatrix} 0 & 3 & 7 & 5 \\ 2 & 0 & 6 & 4 \\ 3 & 1 & 0 & 5 \\ 5 & 3 & 2 & 0 \end{bmatrix} \end{array}$$

vertex C
alculatethedistancefromthesourcevertextodestinationvertexthroughthisvertex4

5. A^4givestheshortestpathbetweeneachpairofvertices.

Floyd-WarshallAlgorithm

```
n=noof vertices
A=matrixofdimensionn*n for
k = 1 to n
  for i = 1 to n
    forj=1ton
      A^k[i,j]=min(A^{k-1}[i,j],A^{k-1}[i,k]+A^{k-1}[k,j])
return A
```

TimeComplexity

There are three loops. Each loop has constant complexities. So, the time complexity of the Floyd-Warshall algorithm is $O(n^3)$.

NetworkFlow

Flow Network is a directed graph that is used for modeling material Flow. There are two different vertices; one is a**source** whichproducesmaterialat some steady rate,and anotherone issink which consumes the content at the same constant speed. The flow of the material at any mark in the system is the rate at which the element moves.

Somereal-life problemslikethe flowofliquids throughpipes, the currentthroughwiresanddelivery of goods can be modelled using flow networks.

Definition:AFlowNetworkisadirectedgraphG=(V,E)suchthat

1. For each edge (u, v) ∈ E, we associate a nonnegative weight capacity c (u, v) ≥ 0.If (u, v) ∉ E, we assume that c (u, v) = 0.
2. Therearetwodistinguishingpoints,thesources,andthesink t;
3. Foreveryvertexv∈ V,thereisapathfromstotcontainingv.

Let G = (V, E) be a flow network. Let s be the source of the network, and let t be the sink. A flow in G is a real-valued function f: V x V→R such that the following properties hold:

PlayVideo

- **CapacityConstraint:**Forallu,v∈ V,weneedf(u,v)≤c(u,v).
- **SkewSymmetry:**Forallu,v∈ V,weneedf(u,v)=-f(u,v).
- **FlowConservation:**Forallu∈V-{s,t},we need

$$\boxed{\sum_{v \in V} f(u,v) = \sum_{u \in V} f(u,v) = 0}$$

Thequantityf(u,v),whichcanbepositiveornegative,isknownasthenetflowfromvertexuto vertexv.Inthe**maximum-flowproblem**,wearegivenaflownetworkGwithsourcesandsinkt,and aflowofmaximumvaluefromstot.

Ford-FulkersonAlgorithm

Initially,theflowofvalueis 0.Find someaugmentingPathpandincreaseflowf oneachedge of pby residual Capacity c_f (p). When no augmenting path exists, flow f is a maximum flow.

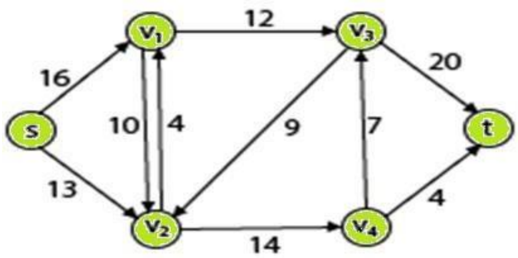
FORD-FULKERSONMETHOD(G,s,t)

1. Initializeflowfto0
2. whilethereexistsanaugmentingpathp
3. doargumentflowfalongp
4. Returnf

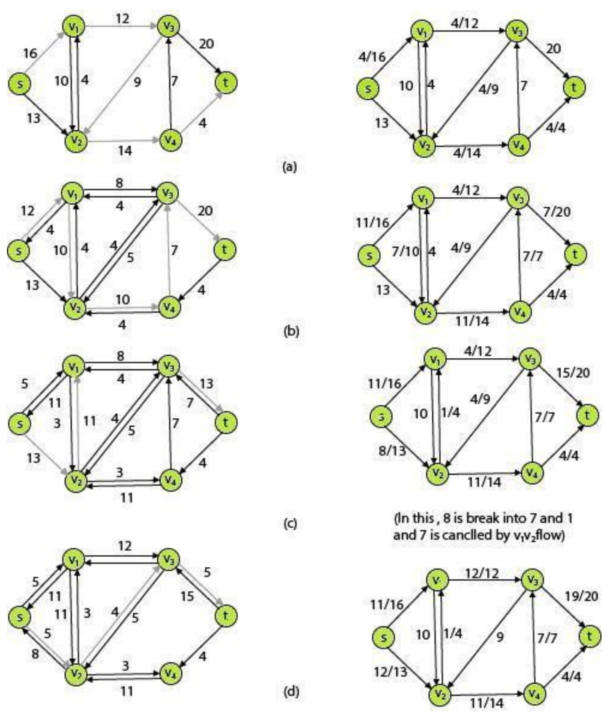
FORD-FULKERSON(G,s,t)

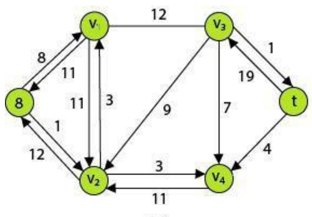
1. foreachedge(u,v)∈E [G]
2. dof[u, v]←0
3. f[u,v]←0
4. whilethereexistsapathpfromstotintheresidualnetworkG_f.
5. doc_f(p)←min?{C_f(u,v):(u,v)isonp}
6. foreachedge(u,v)inp
7. dof [u,v]←f[u, v]+ c_f(p)
8. f[u,v]←-f[u,v]

Example: Each Directed Edge is labeled with capacity. Use the Ford-Fulkerson algorithm to find the maximum flow.



Solution: The left side of each part shows the residual network $G_f$with a shaded augmenting pathp,and the right side of each part shows the net flow f.



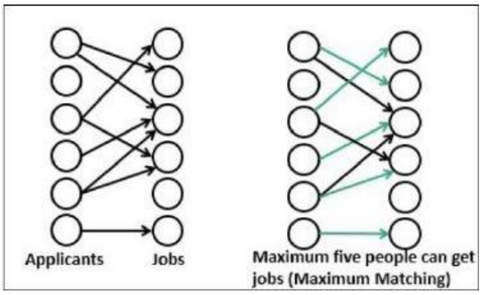


(e)

Now , it has no augmenting paths .So, the maximum flow shown in (d) is 23 is a maximum flow .

MaximumBipartiteMatching

The bipartite matching is a set of edges in a graph is chosen in such a way, that no two edges in that set will share an endpoint. The maximum matching is matching the maximum number of edges.



When the maximum match is found, we cannot add another edge. If one edge is added to the maximum matched graph, it is no longer a matching. For a bipartite graph, there can be more than one maximum matching is possible.

Algorithm

bipartiteMatch(u,visited,assign)
Input: Startingnode,visitedlisttokeeptrack,assignthelisttoassignnodewithanothernode.
Output−Returnstruewhenamatchingforvertexuispossible.

```
Begin
  forallvertexv,whichareadjacentwithu,do if v is
    not visited, then
      markvas visited
      ifvisnotassigned,orbipartiteMatch(assign[v],visited,assign)istrue,then assign[v] := u
        returntrue
  done
  returnfalse
End
```

maxMatch(graph)Input
−Thegivengraph.
Output−Themaximumnumberofthematch.

```
Begin
  initiallynovertexisassigned
  count := 0
  for all applicant u in M, do
    makeallnodeasunvisited
    ifbipartiteMatch(u,visited,assign),then
      increase count by 1
  done
End
```

Unit3

DivideandConquerAlgorithm

Adivideandconqueralgorithmis astrategy ofsolvingalargeproblemby

1. breakingtheproblemintosmallersub-problems
2. solvingthesub-problems,and
3. combiningthemtogetthedesiredoutput.

Tousethedivideandconqueralgorithm,recursionis used.

HowDivideandConquerAlgorithmsWork?

Herearethesteps involved:

1. Divide:Dividethegivenproblemintosub-problemsusing recursion.
2. Conquer:Solvethesmallersub-problemsrecursively.Ifthesubproblemissmall enough, then solve it directly.
3. Combine:Combinethesolutionsofthesub-problemsthatarepartoftherecursive process to solve the actual problem.

FindingMaximumand Minimum

To find the maximum and minimum numbers in a given array *numbers[]* of size n, the followingalgorithmcan beused.Firstwearerepresentingthenaivemethodandthen we will present divide and conquer approach.

NaïveMethod

Naïve method is a basic method to solve any problem. In this method, the maximum and minimumnumbercanbefoundseparately.Tofindthemaximumandminimumnumbers, the following straightforward algorithm can be used.

Algorithm:Max-Min-Element(numbers[])

```
max := numbers[1]
min:=numbers[1]
for i = 2 to n do
  ifnumbers[i]>maxthen
    max := numbers[i]
  ifnumbers[i]<minthen
    min := numbers[i]
return(max,min)
```

Analysis

ThenumberofcomparisoninNaivemethodis2n-2.

Thenumberofcomparisonscan bereducedusingthedivideandconquerapproach. Following is the technique.

DivideandConquer Approach

In this approach, the array is divided into two halves. Then using recursive approach maximum and minimum numbers in each halves are found. Later, return the maximum of two maxima of each half and the minimum of two minima of each half.

Inthisgivenproblem,thenumberofelementsin anarrayisy−x+1, whereyisgreaterthan or equal to x.

Max−Min(x,y)will returnthemaximumandminimum valuesofanarraynumbers[x...y].

Algorithm:Max-Min(x,y)

ify –x ≤1then
  return(max(numbers[x],numbers[y]),min((numbers[x],numbers[y]))
else
  (max1,min1):=maxmin(x,⌊((x+ y)/2)⌋)
  (max2,min2):=maxmin(⌊((x+y)/2)+1)⌋,y)
return(max(max1, max2),min(min1,min2))
Analysis
Let*T(n)* bethenumberofcomparisonsmadebyMax−Min(x,y), wherethenumberof elements n=y−x+1.

If*T(n)*representsthenumbers,thentherecurrencerelationcanberepresentedas

$$T(n) = \begin{cases} T\left(\lfloor \frac{n}{2} \rfloor\right) + T\left(\lceil \frac{n}{2} \rceil\right) + 2 & for\, n > 2 \\ 1 & for\, n = 2 \\ 0 & for\, n = 1 \end{cases}$$

Letusassumethat*n*isintheformofpowerof 2.Hence,n= 2^kwherekisheightofthe recursion tree.

So,

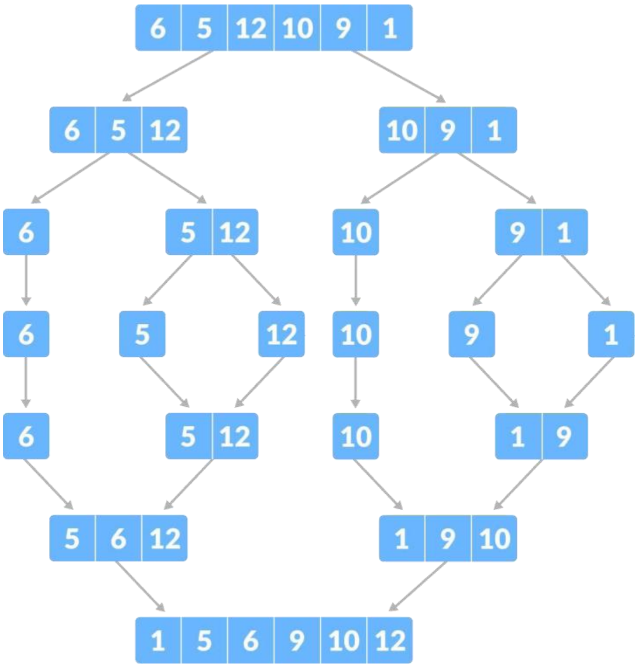
$$T(n) = 2.T(\frac{n}{2}) + 2 = 2.\left(2.T(\frac{n}{4}) + 2\right) + 2\ldots.. = \frac{3n}{2} - 2$$

ComparedtoNaïvemethod,individeandconquerapproach,thenumberofcomparisonsis less. However, using the asymptotic notation both of the approaches are represented by O(n).

MergeSort

MergeSortisoneofthemostpopularsortingalgorithmsthat isbasedontheprinciple of Divide and Conquer Algorithm.

Here,aproblemisdividedintomultiplesub-problems.Eachsub-problemissolved individually. Finally, sub-problems are combined to form the final solution.



MergeSort example

DivideandConquer Strategy

Usingthe**DivideandConquer**technique,wedivideaproblemintosubproblems.Whenthe solution to each subproblem is ready, we 'combine' the results from the subproblems to solve the main problem.

Supposewe hadtosortanarrayA.Asubproblemwouldbetosortasub-sectionofthis array starting at index p and ending at index r, denoted as A[p..r].

Divide

Ifqisthehalf-waypointbetweenpandr,thenwecansplitthesubarrayA[p..r]intotwo arrays A[p..q] and A[q+1, r].

Conquer

Intheconquerstep,wetrytosortboth thesubarraysA[p..q]andA[q+1,r].Ifwehaven'tyet reached the base case, we again divide both these subarrays and try to sort them.

Combine

Whentheconquer stepreachesthebasestepandwegettwosorted subarraysA[p..q]andA[q+1,r]forarrayA[p..r],wecombinetheresultsbycreatingasorted array A[p..r] from two sorted subarrays A[p..q] and A[q+1, r].

MergeSort Algorithm

TheMergeSortfunctionrepeatedlydividesthearrayintotwo halvesuntilwe reachastage where we try to perform MergeSort on a subarray of size 1 i.e. p == r.

Afterthat,themergefunctioncomesintoplayandcombinesthesortedarraysinto larger arrays until the whole array is merged.

```
MergeSort(A,p,r): if
  p > r
    return
  q = (p+r)/2
  mergeSort(A, p, q)
  mergeSort(A,q+1,r)
  merge(A, p, q, r)


voidmerge(intarr[],intp,intq,intr)

{

  //CreateL←A[p..q]andM←A[q+1..r] int
  n1 = q - p + 1;
  intn2=r-q;
  intL[n1],M[n2];
  for(inti=0;i<n1;i++) L[i] =
    arr[p + i];
  for(intj=0;j<n2;j++) M[j]
    = arr[q + 1 + j];

  //Maintaincurrentindexofsub-arraysandmainarray int i,
  j, k;
  i=0;
  j=0;
  k=p;

  //Untilwereacheither endofeitherLorM,picklarger among
  //elementsLandMandplacetheminthecorrectpositionatA[p..r] while (i <
  n1 && j < n2)
      {
              if(L[i] <=M[j])
                      {
                              arr[k]= L[i];
```

```
                        i++;
                    }
            else
                    {
                        arr[k]=M[j]; j++;

                    }
            k++;
        }

    //WhenwerunoutofelementsineitherL orM,
    //pickuptheremainingelementsandputinA[p..r] while (i
            < n1)
                    {
                        arr[k]=L[i];
                        i++;
                        k++;
                    }

            while(j <n2)
                    {
                        arr[k]=M[j]; j++;
                        k++;

                    }
        }
```

Time Complexity

Best Case Complexity: O(n*log n)

Worst Case Complexity: O(n*log n)

AverageCaseComplexity:O(n*logn)

Dynamic Programming

MatrixChainMultiplication

Dynamicprogrammingisamethodforsolvingoptimization problems. Itisalgorithmtechniquetosolve acomplexandoverlappingsub-problems.Computethe solutionsto thesub-problemsonce andstorethesolutionsinatable, sothattheycanbe reused (repeatedly) later.

Dynamicprogrammingismoreefficientthenotheralgorithmmethodslikeas Greedy method, Divide and Conquer method, Recursion method, etc….

The real time many of problems are not solve using simple and traditional approach methods. like as coin change problem , knapsack problem, Fibonacci sequence generating , complexmatrixmultiplication….TosolveusingIterativeformula,tediousmethod,repetition again and again it become a more time consuming and foolish. some of the problem it should be necessary to divide a sub problems and compute its again and again to solve a suchkindofproblemsandgivetheoptimalsolution,effectivesolutiontheDynamic programming is needed…

BasicFeaturesofDynamicprogramming:-

- Getallthepossiblesolutionandpickupbestandoptimal solution.
- Workonprincipalofoptimality.
- Definesub-partsandsolvethem usingrecursively.
- Lessspace complexityButmoreTimecomplexity.
- Dynamicprogrammingsavesusfromhavingtorecomputepreviouslycalculatedsub-solutions.
- Difficultto understanding.

We are covered a many of the real world problems.In our day to day life when we do making coin change, robotics world, aircraft, mathematical problems like Fibonacci sequence,simplematrixmultiplicationofmorethentwomatricesanditsmultiplication possibility is many more so in that get the best and optimal solution. NOW we can look about one problem that is ***MATRIX CHAIN MULTIPLICATION PROBLEM.***

Suppose,Wearegivenasequence(chain)***(A1,A2……An)***ofnmatricestobemultiplied,and we wish to compute the product ***(A1A2…..An)***.We can evaluate the above expression using the standard algorithm for multiplying pairs of matrices as a subroutine once we have parenthesized it to resolve all ambiguities in how the matrices are multiplied together. Matrixmultiplicationisassociative,andsoallparenthesizationsyield thesameproduct.For example, if the chain of matrices is ***(A1, A2, A3, A4)*** then we can fully parenthesize the product (***A1A2A3A4)*** in five distinct ways:

1:-(A1(A2(A3A4))),

2:-(A1((A2A3)A4)),

3:-((A1A2)(A3A4)),

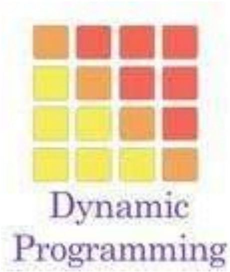
4:-((A1(A2A3))A4),

5:-(((A1A2)A3)A4).

WecanmultiplytwomatricesAandBonlyiftheyarecompatible.thenumberofcolumnsof A must equal the number of rows of B. If A is a p x q matrix and B is a q x r matrix,the resulting matrix C is a p x r matrix. The time to compute C is dominated by the number of scalar multiplications is pqr. we shall express costs in terms of the number of scalar multiplications.For example, if we have three matrices (A1,A2,A3) and its cost is (10x100),(100x5),(5x500)respectively. so we can calculate thecost of scalarmultiplication is 10*100*5=5000 if ((A1A2)A3), 10*5*500=25000 if (A1(A2A3)), and so on cost calculation. ***Note that in the matrix-chain multiplication problem, we are not actually multiplyingmatrices.Ourgoalisonlytodetermineanorderformultiplyingmatricesthat has the lowest cost.***that is here is minimum cost is 5000 for above example .So problem is we can perform a many time of cost multiplication and repeatedly the calculation is performing.sothisgeneralmethodisverytimeconsumingandtedious.Sowecan apply ***dynamic programming*** for solve this kind of problem.

whenweusedtheDynamicprogrammingtechniqueweshallfollowsomesteps.

1. Characterizethestructureofanoptimal solution.
2. Recursivelydefinethevalueofanoptimalsolution.
3. Computethevalueofanoptimal solution.
4. Constructanoptimalsolutionfromcomputedinformation.



wehavematricesofanyoforder.ourgoalisfindoptimalcostmultiplicationof matrices.when we solve the this kind of problem using DP step 2 we can get

m[i ,j]=min {m[i , k]+m[i+k, j]+ pi-1*pk*pj}ifi <j….wherep isdimensionofmatrix,i≤ k < j …..

Thebasicalgorithmofmatrixchainmultiplication:-

```
//MatrixA[i]hasdimensiondims[i-1]xdims[i]fori =1..n
MatrixChainMultiplication(intdims[])
{
//length[dims]=n+1
n=dims.length -1;
//m[i,j]=Minimumnumberofscalarmultiplications(i.e.,cost)
//neededtocomputethematrixA[i]A[i+1]...A[j]= A[i..j]
//Thecostiszerowhenmultiplyingonematrix
for(i=1;i<=n;i++)
m[i, i] = 0;

for(len=2;len<=n;len++){
//Subsequence lengths
for(i=1;i<=n-len+1;i++){ j = i +
len - 1;
m[i, j]=MAXINT;
for(k =i;k <=j-1;k++) {
cost= m[i,k]+m[k+1,j]+dims[i-1]*dims[k]*dims[j];
if(cost<m[i,j]){ m[i,
j] = cost;
s[i,j]=k;
//Indexofthesubsequencesplitthatachievedminimalcost
}
```

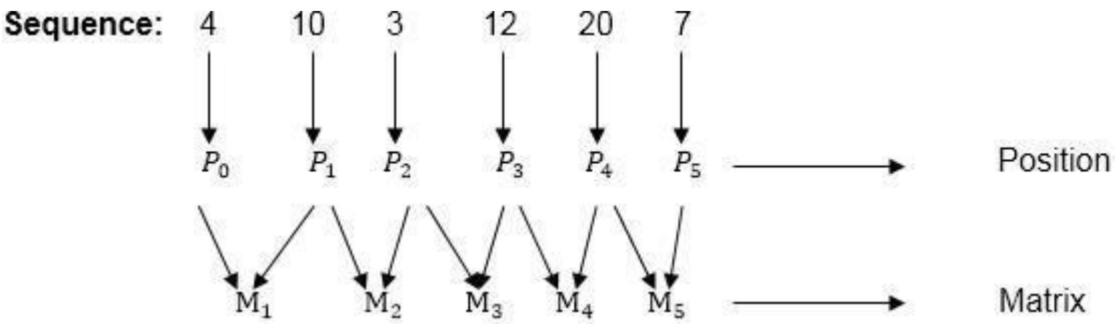
}
}
}
}

ExampleofMatrixChainMultiplication

Example: Wearegiventhesequence {4, 10,3, 12,20, and7}.Thematriceshavesize4 x10, 10x3,3x12,12x20,20x7.We needtocomputeM[i,j],0 ≤i, j≤ 5.We knowM [i,i]=0 for all i.

1	2	3	4	5	
0					1
	0				2
		0			3
			0		4
				0	5

Letusproceedwithworkingawayfromthediagonal.We computetheoptimalsolutionfor the product of 2 matrices.



InDynamicProgramming,initializationofeverymethoddoneby‘0’.Soweinitializeitby ‘0’.It will sort out diagonally.

Wehavetosortoutallthecombinationbuttheminimumoutputcombinationistakeninto consideration.

CalculationofProductof2matrices:

1. m (1,2)=m1x m2
   =4x 10x10x3
   =4x 10x 3=120

2. m (2,3)=m2x m3
   =10x 3x3x 12
   =10x 3x12=360

3. m (3,4)=m3x m4

=3x12x12x20
=3x12x20=720

4. m (4,5)=m4x m5
=12x 20x20x 7
=12x 20x 7=1680

1	2	3	4	5	
0	120				1
	0	360			2
		0	720		3
			0	1680	4
				0	5

- Weinitializethediagonalelementwithequali,j valuewith‘0’.
- Afterthatseconddiagonalissorted outandwegetallthevaluescorrespondedtoit Now

the third diagonal will be solved out in the same way.

Nowproductof3 matrices:

M[1,3] =M1M2 M3

1. Therearetwocasesbywhichwecansolvethismultiplication:(M1xM2)+M3,M1+ (M2x M3)
2. Aftersolvingbothcaseswechoosethecase inwhichminimumoutputisthere.

$$M\,[1,3] = \min \begin{Bmatrix} M\,[1,2] + M\,[3,3] + p_0\,p_2p_3 = 120 + 0 + 4.3.12 \;=\; 264 \\ M\,[1,1] + M\,[2,3] + p_0\,p_1p_3 = 0 + 360 + 4.10.12 \;=\; 840 \end{Bmatrix}$$

M[1,3]=264

AsComparingbothoutput**264**isminimuminbothcasesso weinsert**264**intableand(M1 x M2) + M3 this combination is chosen for the output making.

M[2,4] =M2M3 M4

1. Therearetwocasesbywhichwecansolvethismultiplication:(M2xM3)+M4, M2+(M3 x M4)
2. Aftersolvingbothcaseswechoosethecase inwhichminimumoutputisthere.

$$M\,[2,4] = \min \begin{Bmatrix} M[2,3] + M[4,4] + \; p_1p_3p_4 = 360 + 0 + 10.12.20 = 2760 \\ M[2,2] + \; M[3,4] + \; p_1p_2p_4 = 0 + 720 + 10.3.20 = \; 1320 \end{Bmatrix}$$

M[2,4]=1320

AsComparingbothoutput**1320**isminimuminbothcasessoweinsert**1320**intableand M2+(M3 x M4) this combination is chosen for the output making.

M[3,5]= M3M4M5

1. Therearetwocasesbywhichwecansolvethismultiplication:(M3xM4)+M5,M3+ (M4xM5)
2. Aftersolvingbothcaseswechoosethecase inwhichminimumoutputisthere.

$$M\,[3,5] = \min \begin{cases} M[3,4] + M[5,5] + p_2p_4p_5 = 720 + 0 + 3\,.20.7 = 1140 \\ M[3,3] + M\,[4,5] + p_2p_3p_5 = 0 + 1680 + 3.12.7 = 1932 \end{cases}$$

M[3,5]=1140

AsComparingbothoutput**1140**isminimuminbothcasessoweinsert**1140**intableand (M3 x M4) + M5this combination is chosen for the output making.

1	2	3	4	5	
0	120				1
	0	360			2
		0	720		3
			0	1680	4
				0	5

$\rightarrow$

1	2	3	4	5	
0	120	264			1
	0	360	1320		2
		0	720	1140	3
			0	1680	4
				0	5

NowProductof4matrices:

M[1,4] =M1M2M3 M4

Therearethreecasesbywhich wecansolvethismultiplication:

1. (M1 xM2 x M3)M4
2. M1x(M2x M3xM4)
3. (M1xM2)x (M3xM4)

Aftersolvingthesecaseswechoosethecase inwhichminimumoutputisthere

$$M\,[1,4] = \min \begin{cases} M[1,3] + M[4,4] + p_0p_3p_4 = 264 + 0 + 4.12.20 = 1224 \\ M[1,2] + M[3,4] + p_0p_2p_4 = 120 + 720 + 4.3.20 = 1080 \\ M[1,1] + M[2,4] + p_0p_1p_4 = 0 + 1320 + 4.10.20 = 2120 \end{cases}$$

M[1,4]=1080

Ascomparing theoutputofdifferentcases then‘**1080**’is minimumoutput,sowe insert 1080inthetableand(M1xM2) x(M3xM4) combinationistakenoutinoutputmaking,

M[2,5] =M2 M3M4 M5

Therearethreecasesbywhich wecansolvethismultiplication:

1. (M2x M3x M4)x M5
2. M2x(M3 x M4xM5)

3. (M2x M3)x(M4xM5)

Aftersolvingthesecaseswechoosethecase inwhichminimumoutputisthere

$$M\,[2,5] = \min \begin{cases} M[2,4] + M[5,5] + p_1p_4p_5 = 1320 + 0 + 10.20.7 = 2720 \\ M[2,3] + M[4,5] + p_1p_3p_5 = 360 + 1680 + 10.12.7 = 2880 \\ M[2,2] + M[3,5] + p_1p_2p_5 = 0 + 1140 + 10.3.7 = 1350 \end{cases}$$

M[2,5]=1350

Ascomparingtheoutputofdifferentcasesthen‘**1350**’isminimumoutput,sowe insert 1350 in the table and M2 x(M3 x M4xM5)combination is taken out in output making.

1	2	3	4	5	
0	120	264			1
	0	360	1320		2
		0	720	1140	3
			0	1680	4
				0	5

→

1	2	3	4	5	
0	120	264	1080		1
	0	360	1320	1350	2
		0	720	1140	3
			0	1680	4
				0	5

NowProductof5matrices:

M[1,5] =M1M2M3M4 M5

Therearefivecasesbywhichwe cansolvethismultiplication:

1. (M1x M2xM3x M4)xM5
2. M1x(M2 xM3x M4xM5)
3. (M1x M2xM3)xM4 xM5
4. M1x M2x(M3x M4xM5)

Aftersolvingthesecaseswechoosethecase inwhichminimumoutputisthere

$$M\,[1,5] = \min \begin{cases} M[1,4] + M[5,5] + p_0p_4p_5 = 1080 + 0 + 4.20.7 = 1544 \\ M[1,3] + M[4,5] + p_0p_3p_5 = 264 + 1680 + 4.12.7 = 2016 \\ M[1,2] + M[3,5] + p_0p_2p_5 = 120 + 1140 + 4.3.7 = 1344 \\ M[1,1] + M[2,5] + p_0p_1p_5 = 0 + 1350 + 4.10.7 = 1630 \end{cases}$$

M[1,5]=1344

As comparing the output of different cases then ‘**1344**’ is minimum output, so we insert 1344inthetableandM1xM2x(M3xM4xM5)combinationistakenoutinoutputmaking.

FinalOutputis:

1	2	3	4	5	
0	120	264	1080		1
	0	360	1320	1350	2
		0	720	1140	3
			0	1680	4
				0	5

→

1	2	3	4	5	
0	120	264	1080	1344	1
	0	360	1320	1350	2
		0	720	1140	3
			0	1680	4
				0	5

Sowe cangettheoptimalsolutionofmatrices multiplication….

MultiStageGraph

MultistageGraphproblemisdefinedas follow:

- Multistage graph G = (V, E, W) is a weighted directed graph in which vertices are partitioned into k ≥ 2 disjoint sub sets V = {V_1, V_2, …, V_k} such that if edge (u, v) is presentinE thenu∈ $V_i$andv∈ V_{i+1},1 ≤i≤ k.Thegoalofmultistagegraphproblemis to find minimum cost path from source to destination vertex.
- Theinputtothealgorithmisak-stagegraph,nverticesareindexedinincreasing order of stages.
- Thealgorithmoperatesinthebackwarddirection,i.e.itstartsfromthelast vertexof the graph and proceeds in a backward direction to find minimum cost path.
- Minimumcostofvertexj∈$V_i$fromvertexr∈V_{i+1}isdefinedas, Cost[j]

= min{ c[j, r] + cost[r] }

where,c[j, r]istheweightofedge<j, r>andcost[r]isthecostofmovingfromend vertex to vertex r.

- Algorithmforthemultistagegraphisdescribedbelow:

Algorithm for Multistage Graph

AlgorithmMULTI_STAGE(G,k,n,p)
//Description:Solvemulti-stageproblemusingdynamicprogramming

//Input:
k:NumberofstagesingraphG=(V,E) c[i,
j]:Cost of edge (i, j)

//Output:p[1:k]:Minimumcostpath

cost[n] ← 0
forj←n–1to1**do**

//Letrbeavertexsuchthat(j,r)inEandc[j,r]+cost[r]isminimum cost[j] ← c[j,
r] + cost[r]
π[j]←r
end

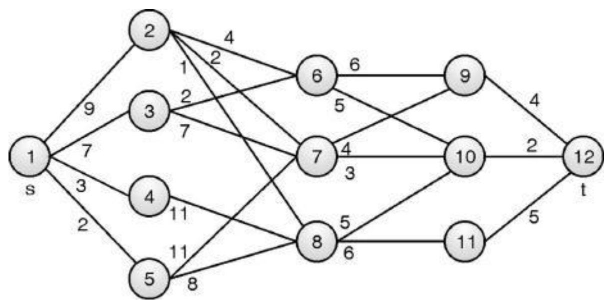
//Findminimumcostpath
p[1] ← 1
p[k]←n

forj←2tok-1**do**
p[j]←π[p[j-1]]
end
ComplexityAnalysisofMultistageGraph

IfgraphGhas|E|edges,thencostcomputationtimewouldbeO(n +|E|).Thecomplexity of tracing the minimum cost path would be O(k), k < n. Thus total time complexity of multistage graph using dynamic programming would be O(n + |E|).

Example

Example:Findminimumpathcostbetweenvertexsandtforfollowingmultistagegraph using dynamic programming.



Solution:

Solutiontomultistagegraphusingdynamicprogrammingisconstructedas, Cost[j] =

min{c[j, r] + cost[r]}

Here,numberofstagesk=5,numberofverticesn=12, sources=1 andtargett =12 Initialization:

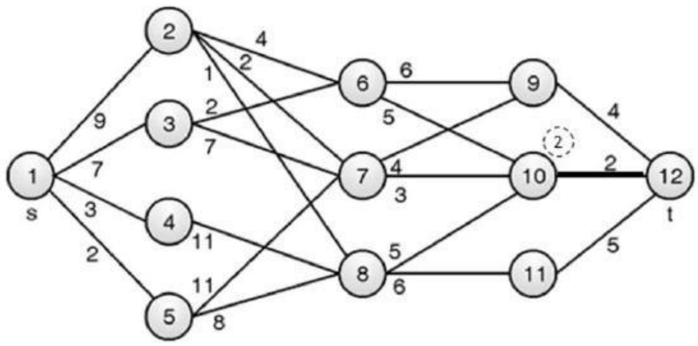
Cost[n]=0⇒Cost[12]=0.

p[1] = s ⇒ p[1] = 1

p[k]=t⇒p[5]=12. r =

t = 12.

Stage4:



Stage3:

Vertex6isconnected tovertices9and10:

Cost[6]=min{c[6,10]+Cost[10],c[6,9]+ Cost[9]}

=min{5+2,6+ 4}=min{7,10}=7

p[6]=10

Vertex7isconnected tovertices9and10:

Cost[7]=min{c[7,10]+Cost[10],c[7,9]+ Cost[9]}

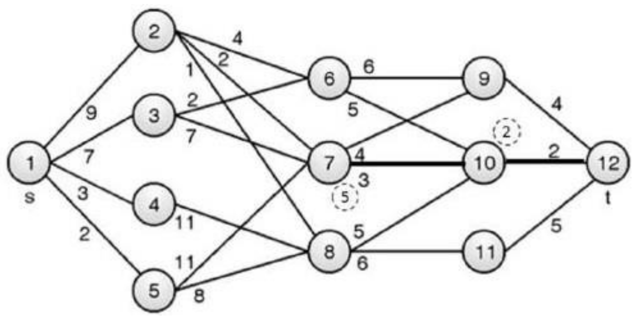
=min{3+2,4+ 4}=min{5,8}=5

p[7]=10

Vertex8isconnected tovertex 10and11:

Cost[8]=min{c[8,11]+Cost[11],c[8,10]+Cost[10]}

=min{6+5,5+2}=min{11,7}=7p[8]=10



Stage2:

Vertex2isconnected tovertices6,7and8:

Cost[2]=min{c[2,6]+Cost[6], c[2,7]+Cost[7], c[2,8] +Cost[8]}

=min{4+7,2+5,1+7}=min{11,7, 8}=7

p[2]=7

Vertex3isconnectedtovertices6and7:

Cost[3]=min{c[3,6]+Cost[6],c[3,7]+Cost[7]}

=min{2+7,7+ 5}=min{9,12}=9

p[3]=6

Vertex4isconnectedtovertex 8:

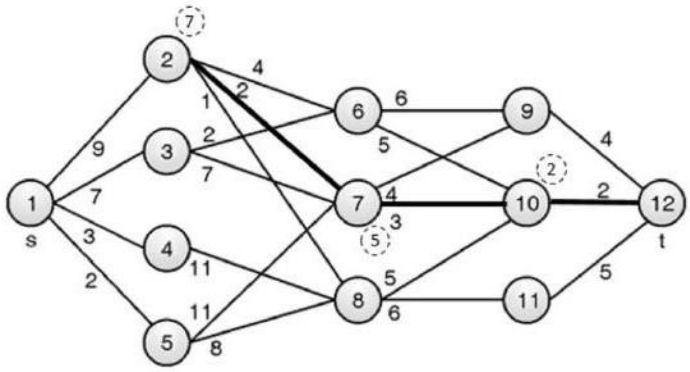
Cost[4]=c[4, 8]+Cost[8]= 11+7=18

p[4]=8

Vertex5isconnected tovertices7and8:

Cost[5]=min{c[5,7]+Cost[7],c[5,8]+Cost[8]}

=min{11+5,8+7}=min{16,15}=15p[5]=8



Stage1:

Vertex1isconnected tovertices2,3, 4and5:

Cost[1]=min{c[1,2]+Cost[2],c[1, 3]+ Cost[3],c[1,4]+ Cost[4],c[1,5]+Cost[5]}

=min{9+7,7+9,3+18,2+15 }

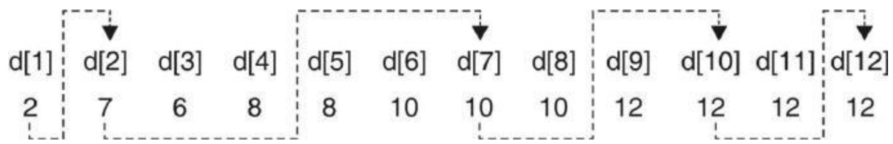
=min{16,16,21,17}=16p[1]=2

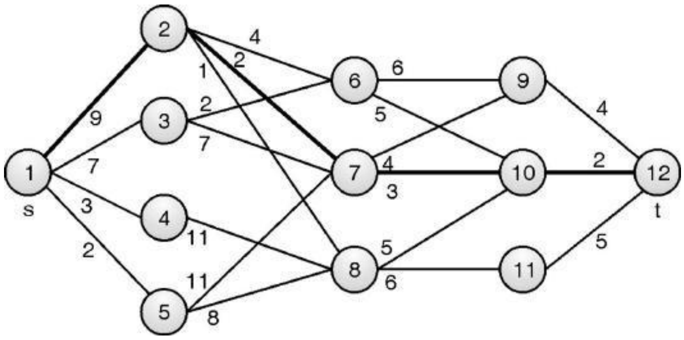
Tracethe solution:

p[1]=2

p[2]=7

p[7]=10





p[10]=12

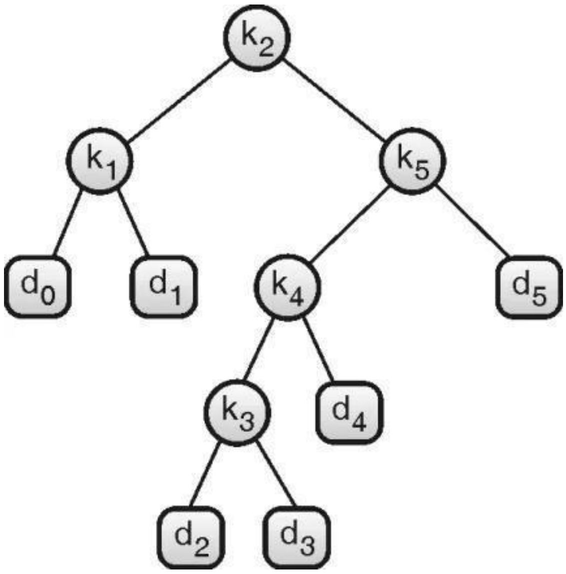
Minimumcostpathis: 1–2–7–10–12

Costofthepathis:9+2+3+2=16

OptimalBinarySearchTree

- OptimalBinary SearchTreeextends theconceptofBinary searctree. BinarySearch Tree(BST) isa*nonlinear*datastructurewhich isusedinmanyscientificapplications for reducing the search time. In BST, left child is smaller than root and right child is greater than root. This arrangement simplifies the search procedure.
- Optimal Binary Search Tree (OBST) is very useful in dictionary search. The probability ofsearchingisdifferentfor differentwords. OBST hasgreat applicationintranslation. If we translate the book from English to German, equivalent words are searched fromEnglishtoGermandictionaryandreplacedintranslation.Wordsaresearched same as in binary search tree order.
- Binarysearchtreesimplyarrangesthewordsinlexicographicalorder.Words like *'the'*, *'is'*, *'there'* are very frequent words, whereas words like*'xylophone'*,*'anthropology'*etc.appearsrarely.
- Itisnotawise ideatokeeplessfrequentwordsnearrootinbinarysearchtree. Instead of storing words in binary search tree in lexicographical order, we shall arrange them according to their probabilities. This arrangement facilitates few searches for frequent words as they would be near the root. Such tree is called**OptimalBinarySearch Tree**.
- Considerthesequenceof*n*keysK=<k_1,k_2,k_3,…,k_n>ofdistinctprobabilityinsorted order such that k_1<k_2<…<k_n.Wordsbetweeneachpairofkeyleadtounsuccessfulsearch,soforn keys, binary search tree contains n + 1 dummy keys d_i, representing unsuccessful searches.

- TwodifferentrepresentationofBSTwithsamefivekeys{k_1,k_2,k_3,k_4,k_5}probability is shown in following figure
- With n nodes, there exist (2n)!/((n + 1)! * n!) different binary search trees. An exhaustivesearchforoptimalbinarysearch treeleadstohugeamountoftime.
- The goal is to construct a tree which minimizes the total search cost. Such tree is calledoptimalbinarysearchtree.OBSTdoesnotclaimminimumheight.It isalsonot necessary that parent of sub tree has higher priority than its child.
- Dynamicprogramming canhelpustofindsuchoptima tree.



Binarysearchtreeswith5keys

Mathematicalformulation

- WeformulatetheOBSTwithfollowing observations
- AnysubtreeinOBST containskeysinsortedorderk_i…k_j,where1≤i≤j≤ n.
- Subtreecontainingkeysk_i…k_j hasleaveswithdummykeysd_{i-1}….d_j.
- Supposek$_r$istherootofsubtreecontainingkeysk_i…..k_j.So,leftsubtreeofroot k_r contains keys k_i….k_{r-1}andrightsubtreecontainkeysk_{r+1}tok_j.Recursively,optimalsubtreesare constructed from the left and right sub trees of k_r.
- Lete[i,j]representstheexpected costofsearchingOBST. Withnkeys,ouraimisto find and minimize e[1, n].
- Basecaseoccurswhenj=i–1,becausewejusthavethedummykeyd_{i-1}forthis case. Expected search cost for this case would be e[i, j] = e[i, i – 1] = q_{i-1}.
- Forthecasej≥i,we havetoselectanykey$k_r$fromk_i…$k_j$asarootofthetree.
- With$k_r$asarootkey andsubtreek_i…k_j,sumofprobability isdefinedas

$$w(i, j) = \sum_{m=i}^{j} p_m + \sum_{m=i-1}^{j} q_m$$

(Actualkeystartsatindex1anddummykeystartsatindex0)

Thus,arecursiveformulaforformingtheOBSTisstatedbelow:

$$e[i, j] = \begin{cases} q_{i-1} & \text{if } j = i-1 \\ \min\limits_{i \le r \le j} \{e[i, r-1] + e[r+1, j] + w(i, j)\} & \text{if } i \le j \end{cases}$$

e[i,j]givestheexpectedcostintheoptimalbinarysearchtree.

AlgorithmforOptimalBinarySearchTree

Thealgorithmforoptimalbinary searchtree isspecifiedbelow:

AlgorithmOBST(p, q,n)

//e[1…n+1,0…n]: Optimalsubtree

//w[1…n+1,0…n]:Sumofprobability

//root[1…n,1…n]:UsedtoconstructOBST

fori←1ton+1 **do**

e[i,i–1]←qi–1

w[i, i–1]←qi–1

end

form←1ton **do**

fori←1ton–m+1 **do**

j←i+m–1 e[i,

j] ← ∞

w[i,j]←w[i,j–1]+pj+qj

forr←itoj**do**

t←e[i,r–1]+e[r+1,j]+w[i,j]

```
        **if**t<e[i,j]**then**
            e[i, j] ← t
            root[i, j] ← r
        **end**
    **end**
  **end**
**end**
**return**(e,root)
```

ComplexityAnalysisofOptimalBinarySearchTree

Itisverysimpletoderivethecomplexityofthisapproachfromtheabovealgorithm.It uses threenestedloops.Statementsin theinnermostloopruninQ(1)time.Therunningtimeof the algorithm is computed as

$$
\begin{aligned}
T(n) &= \sum_{m=1}^{n} \sum_{i=1}^{n-m+1} \sum_{j=i}^{n-1+1} \Theta(1) \\
&= \sum_{m=1}^{n} \sum_{i=1}^{n-m+1} n = \sum_{m=1}^{n} n^2 \\
&= \Theta(n^3)
\end{aligned}
$$

Thus,theOBSTalgorithmrunsincubictime

Example

Problem:Let p (1:3)= (0.5,0.1,0.05)q(0:3)=(0.15,0.1,0.05,0.05)Computeand constructOBSTforabovevaluesusingDynamicapproach.

Solution:

Here,giventhat

i	0	1	2	3
p_i		0.5	0.1	0.05
q_i	0.15	0.1	0.05	0.05

RecursiveformulatosolveOBST problemis

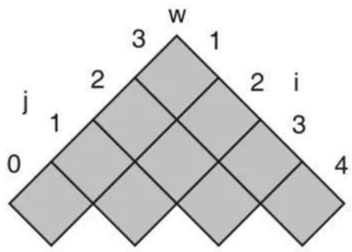
$$
e[i, j] = \begin{cases} q_{i-1} & \text{if } j = i-1 \\ \min\limits_{i \le r \le j} \{e[i, r-1] + e[r+1, j] + w(i, j)\} & \text{if } i \le j \end{cases}
$$

Where,

$$w(i, j) = \sum_{m=i}^{j} p_m + \sum_{m=i-1}^{j} q_m$$

Initially,

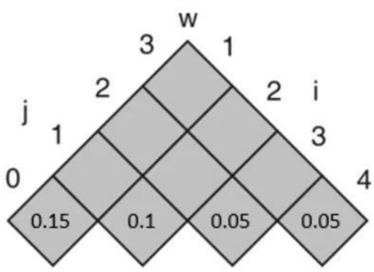
$$w[1, 0] = \sum_{m=1}^{0} p_m + \sum_{m=0}^{0} q_m = q_0 = 0.15$$



$$w[2, 1] = \sum_{m=2}^{1} p_m + \sum_{m=1}^{1} q_m = q_1 = 0.1$$

$$w[3, 2] = \sum_{m=3}^{2} p_m + \sum_{m=2}^{2} q_m = q_2 = 0.05$$

$$w[4, 3] = \sum_{m=4}^{3} p_m + \sum_{m=3}^{3} q_m = q_3 = 0.05$$



$$w[1, 1] = \sum_{m=1}^{1} p_m + \sum_{m=0}^{1} q_m = p_1 + q_0 + q_1$$

$$= 0.5 + 0.25 = 0.75$$

$$w[2, 2] = \sum_{m=2}^{2} p_m + \sum_{m=1}^{2} q_m = p_2 + q_1 + q_2$$

$$= 0.1 + 0.15 = 0.25$$

$$w[3, 3] = \sum_{m=3}^{3} p_m + \sum_{m=2}^{3} q_m = p_3 + q_2 + q_3$$

$$= 0.05 + 0.10 = 0.15$$

$$w[1, 2] = \sum_{m=1}^{2} p_m + \sum_{m=0}^{2} q_m$$

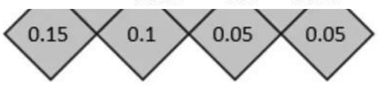
$$= (p_1 + p_2) + (q_0 + q_1 + q_2)$$

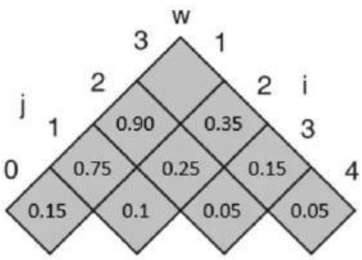
$$= 0.6 + 0.3 = 0.90$$

$$w[2, 3] = \sum_{m=2}^{3} p_m + \sum_{m=1}^{3} q_m$$

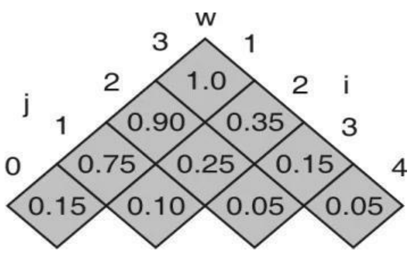
$$= (p_2 + p_3) + (q_1 + q_2 + q_3)$$

$$= 0.15 + 0.2 = 0.35$$



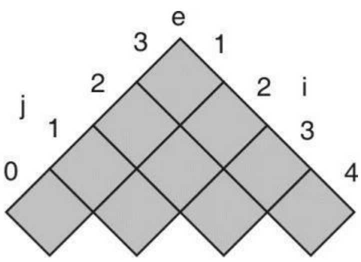


$$w[1, 3] = \sum_{m=1}^{3} p_m + \sum_{m=0}^{3} q_m$$
$$= (p_1 + p_2 + p_3) + (q_0 + q_1 + q_2 + q_3)$$
$$= 0.65 + 0.35 = 1$$



Now,we willcompute e[i,j]

Initially,

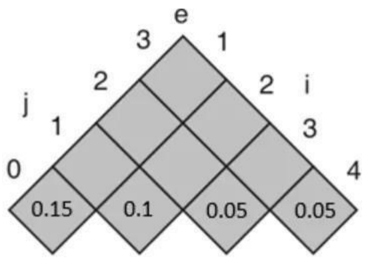


e[1,0]=q_0=0.15(∵j=i–1)

e[2,1]= q_1=0.1 (∵j=i–1)

e[3,2]=q_2=0.05(∵j=i–1)

e[4,3]=q_3=0.05(∵j=i–1)



e[1,1]=min{e[1,0]+e[2,1]+w(1,1)}

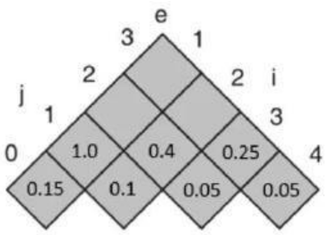
=min{0.15+0.1+0.75}= 1.0

e[2,2]=min{e[2,1]+e[3,2]+w(2,2)}

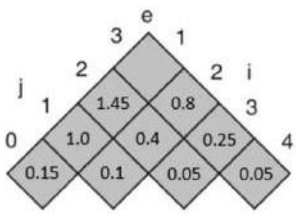
=min{0.1+0.05+0.25}= 0.4

e[3,3]=min{e[3,2]+e[4,3]+w(3,3) }

=min{0.05+0.05+ 0.15}=0.25



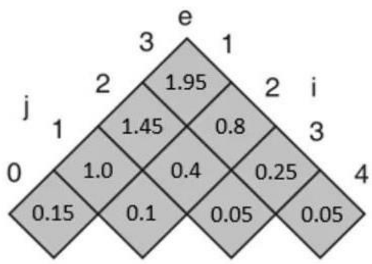
$$
\begin{aligned}
e[1,2] &= \min\left\{\begin{array}{l} e\,[1,0] + e[2,2] + w[1,2] \\ e\,[1,1] + e[3,2] + w[1,2] \end{array}\right\} \\
&= \min\left\{\begin{array}{l} 0.15 + 0.4 + 0.90 \\ 1.0 + 0.05 + 0.90 \end{array}\right\} \\
&= \min\left\{\begin{array}{l} 1.45 \\ 1.95 \end{array}\right\} = 1.45 \\
e[2,3] &= \min\left\{\begin{array}{l} e\,[2,1] + e[3,3] + w(2,3) \\ e\,[2,2] + e[4,3] + w(2,3) \end{array}\right\} \\
&= \min\left\{\begin{array}{l} 0.1 + 0.25 + 0.35 \\ 0.4 + 0.05 + 0.35 \end{array}\right\} \\
&= \min\left\{\begin{array}{l} 0.90 \\ 0.80 \end{array}\right\} = 0.8
\end{aligned}
$$



$$e[1,3] = \min\begin{Bmatrix} e[1,0] + e[2,3] + w(1,3) \\ e[1,1] + e[3,3] + w(1,3) \\ e[1,2] + e[4,3] + w(1,3) \end{Bmatrix}$$

$$= \min\begin{Bmatrix} 0.15 + 0.80 + 1.0 \\ 1.0 + 0.25 + 1.0 \\ 1.45 + 0.05 + 1.0 \end{Bmatrix}$$

$$= \min\begin{Bmatrix} 1.95 \\ 2.25 \\ 2.5 \end{Bmatrix} = 1.95$$



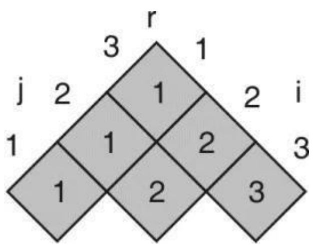
e[1,3]is minimumforr=1,so r[1,3]=1

e[2,3]is minimumforr=2,so r[2,3]=2

e[1,2]is minimumforr=1,so r[1,2]=1

e[3,3]is minimumforr=3,so r[3,3]=3

e[2,2]is minimumforr=2,so r[2,2]=2



e[1, 1] is minimum for r = 1, so r[1, 1] = 1

LetusnowconstructOBSTforgivendata.

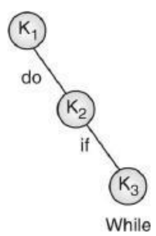
r[1,3] =1, so k_1 will be at the root.

$k_{2....3}$ are on right side of k_1

r[2,3]=2,So$k_2$willbetherootofthissubtree. k_3 will

be on the right of k_2.

Thus,finally,weget.



Greedy

TechniqueActivitySelectio

n Problem

ActivitySelection problemisaapproachofselectingnon-conflictingtasks basedon startand endtimeandcan besolved inO(N logN)timeusingasimplegreedyapproach.Modifications of this problem are complex and interesting which we will explore as well. Suprising, if we use a Dynamic Programming approach, the time complexity will be O(N^3) that is lower performance.

The problem statement for Activity Selection is that "Given a set of n activities with their start and finish times, we need to select maximum number of non-conflicting activities that can be performed by a single person, given that the person can handle only one activity at a time." The Activity Selection problem follows Greedy approach i.e. at every step, we can make a choice that looks best at the moment to get the optimal solution of the complete problem.

Our objective is to complete maximum number of activities. So, choosing the activity which is going to finish first will leave us maximum time to adjust the later activities. This is the intuition that greedily choosing the activity with earliest finish time will give us an optimal solution. By induction on the number of choices made, making the greedy choice at every step produces an optimal solution, so we chose the activity which finishes first. If we sort elements based on their starting time, the activity with least starting time could take the maximum duration for completion, therefore we won't be able to maximise number of activities.

Algorithm

ThealgorithmofActivitySelectionisasfollows:

```
Activity-Selection(Activity, start, finish)
    SortActivitybyfinishtimesstoredinfinish
    Selected = {Activity[1]}
    n=Activity.length j
    = 1
    fori=2to n:
       ifstart[i]≥finish[j]:
          Selected=SelectedU{Activity[i]} j
          = i
 return Selected
```

Complexity

TimeComplexity:
Whenactivitiesaresortedbytheirfinishtime:**O(N)**
Whenactivitiesarenotsortedbytheirfinishtime,thetimecomplexityis**O(N logN)**dueto complexity of sorting

START	1	3	2	0	5	8	11
END	3	4	5	7	9	10	12

START[4]>=END[2], SELECTED

START	1	3	2	0	5	8	11
END	3	4	5	7	9	10	12

START[5]<END[4], REJECTED

START	1	3	2	0	5	8	11
END	3	4	5	7	9	10	12

START[6]>=END[4], SELECTED

	0	1	2	3	4	5	6
START	1	3	2	0	5	8	11
END	3	4	5	7	9	10	12

SELECTED

START	1	3	2	0	5	8	11
END	3	4	5	7	9	10	12

START[1]>=END[0], SELECTED

START	1	3	2	0	5	8	11
END	3	4	5	7	9	10	12

START[2]<END[1], REJECTED

START	1	3	2	0	5	8	11
END	3	4	5	7	9	10	12

START[3]<END[1], REJECTED

Inthisexample,wetakethestartandfinishtimeofactivitiesasfollows: start = [1, 3, 2, 0, 5, 8, 11]
finish=[3,4,5, 7,9,10,12]
Sorted by their finish time, the activity 0 gets selected. As the activity 1 has starting time whichisequaltothe finishtimeofactivity0, itgetsselected.Activities2and3havesmaller starting time than finish time of activity 1, so they get rejected. Based on similar comparisons, activities 4 and 6 also get selected, whereas activity 5 gets rejected. In this example, in all the activities 0, 1, 4 and 6 get selected, while others get rejected.

OptimalMerge Pattern

Mergea setofsortedfilesofdifferentlengthintoa singlesortedfile.Weneedtofindan optimal solution, where the resultant file will be generated in minimum time.

Ifthenumberofsortedfilesaregiven,therearemanywaystomergethemintoasingle sorted file.This merge can be performed pairwise. Hence,this type ofmergingis called as 2-way merge patterns.

As, different pairings require different amounts of time, in this strategy we want to determineanoptimalwayofmergingmanyfilestogether.Ateachstep,twoshortest sequences are merged.

Tomergeap-recordfileandaq-recordfilerequirespossiblyp +qrecordmoves,the obvious choice being, merge the two smallest files together at each step.

Two-way merge patterns can be represented by binary merge trees. Let us consider a set of**n**sortedfiles**{f_1,f_2,f_3,…,f_n}**.Initially,eachelementofthisisconsideredasasinglenode binary tree. To find this optimal solution, the following algorithm is used.

Algorithm:TREE(n)

```
    fori :=1ton– 1do
  declare new node
  node.leftchild := least (list)
  node.rightchild:=least(list)
  node.weight):=((node.leftchild).weight)+((node.rightchild).weight) insert
  (list, node);
returnleast (list);
```

Attheendofthisalgorithm,the weightoftherootnoderepresentstheoptimalcost. Example

Letusconsiderthegivenfiles,f_1,f_2,f_3,$f_4$and$f_5$with20,30,10,5and30numberof elements respectively.

Ifmergeoperationsareperformedaccordingtotheprovidedsequence,then M_1 =

merge f_1 and f_2 => 20 + 30 = 50

M_2=mergeM_1andf_3=>50+10=60 M_3 =

merge M_2 and f_4 => 60 + 5 = 65 M_4

=mergeM_3andf_5=>65+30=95

Hence,thetotalnumberofoperationsis 50 +

60 + 65 + 95 = 270

Now,thequestionarisesisthereanybetter solution?

Sortingthenumbersaccordingtotheirsizeinanascendingorder, wegetthefollowing sequence −

f_4,f_3,f_1,f_2,f_5

Hence,mergeoperationscanbeperformedonthissequence M_1

= merge f_4 and f_3 => 5 + 10 = 15

M_2=mergeM_1andf_1=>15+20=35

M_3=mergeM_2andf_2=>35+30=65 M_4

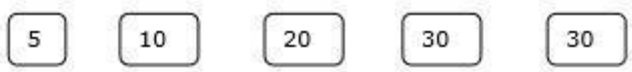
=mergeM_3andf_5=>65+30=95

Therefore,thetotalnumberofoperationsis 15 +

35 + 65 + 95 = 210

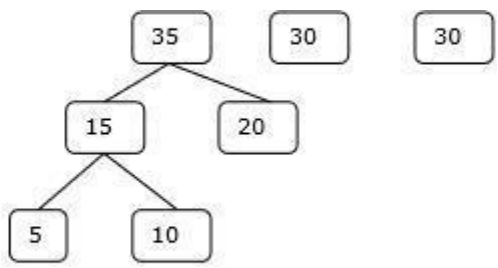
Obviously,thisisbetterthanthepreviousone.

Inthiscontext,wearenowgoingtosolvetheproblemusingthisalgorithm. Initial Set

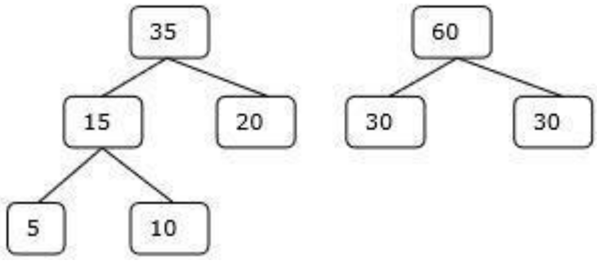


Step1

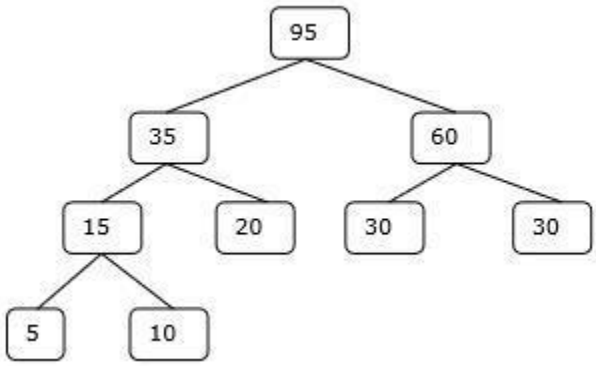
Step2



Step3



Step4



Hence,thesolutiontakes15+ 35+60+ 95= 205numberofcomparisons.

Huffman Tree

Huffman coding provides codes to characters such that the length of the code depends on the relative frequency or weight of the corresponding character. Huffman codes are of variable-length, and without any prefix (that means no code is a prefix of any other). Any prefix-free binary code can be displayed or visualized as a binary tree with the encoded characters stored at the leaves.

Huffman tree or Huffman coding tree defines as a full binary tree in which each leaf of the tree corresponds to a letter in the given alphabet.

The Huffman tree is treated as the binary tree associated with minimum external path weight that means, the one associated with the minimum sum of weighted path lengths for the given set of leaves. So the goal is to construct a tree with the minimum external path weight.

Anexampleisgivenbelow-

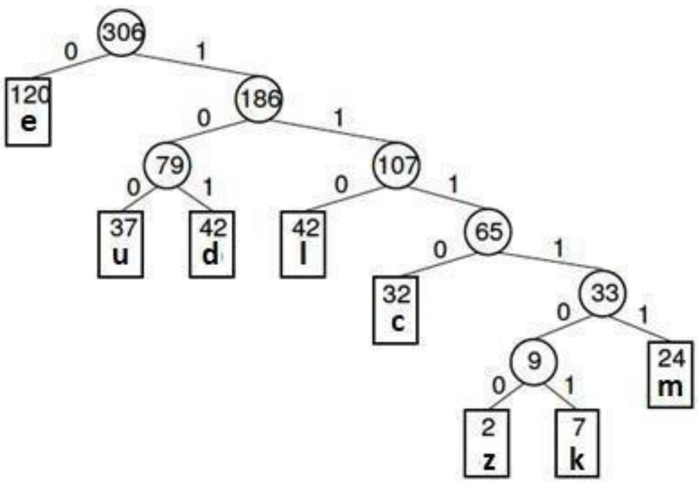
Letter frequency table

Letter	z	k	m	c	u	d	l	e

Frequency	2	7	24	32	37	42	42	120

Huffmancode

Letter	Freq	Code	Bits
e	120	0	1
d	42	101	3
l	42	110	3
u	37	100	3
c	32	1110	4
m	24	11111	5
k	7	111101	6
z	2	111100	6



TheHuffmantree(fortheaboveexample)isgivenbelow-

Algorithm Huffman (c)
{

```
  n=|c|

  Q = c
  fori<-1to n-1

  do
  {

     temp<-getnode()

    left(temp]Get_min(Q)right[temp]GetMin(Q) a =

    left [templ b = right [temp]

    F[temp]<-f[a]+[b]

    insert (Q, temp)

   }

returnGet_min (0)
}
```

UNIT4

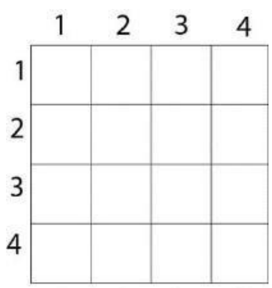
Backtracking

NqueenProblem

N-Queensproblemistoplacen-queensinsuchamanneronannxn chessboardthatnoqueensattack each other by being in the same row, column or diagonal.

Itcanbe seenthatforn=1,theproblemhasatrivialsolution,andnosolutionexistsforn=2andn=3.So first we will consider the 4 queens problem and then generate it to n - queens problem.

Givena4x4chessboardandnumbertherowsandcolumnofthechessboard1through4.



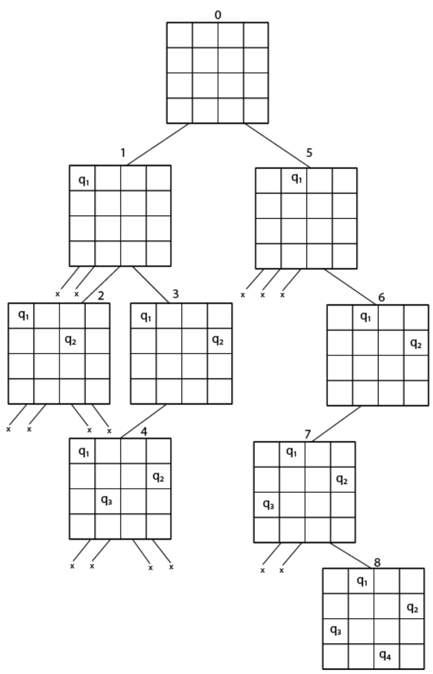
4x4 chessboard

Since, we have to place 4 queens such as $q_1 q_2 q_3$and $q_4$on the chessboard, such that no two queens attack eachother.Insuch aconditionaleachqueenmustbe placedona different row,i.e.,weput queen"i"onrow "i."

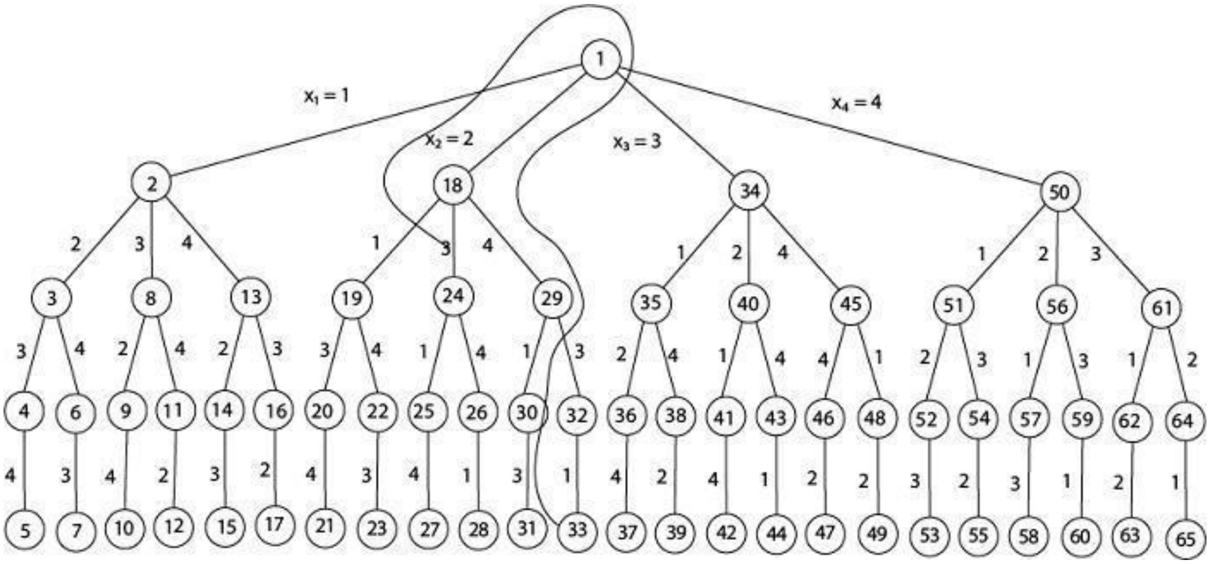
Now, we place queen q_1 in the very first acceptable position (1, 1). Next, we put queen q_2 so that both these queens do not attack each other. We find that if we place q_2 in column 1 and 2, then the dead end is encountered. Thus the first acceptable position for q_2 in column 3, i.e. (2, 3) but then no position is left for placing queen 'q_3' safely. So we backtrack one step and place the queen 'q_2' in (2, 4), the next best possible solution. Then we obtain the position for placing 'q_3' which is (3, 2). But later this position also leads to adead end, and no place is found where 'q_4' can be placed safely. Then we have to backtrack till 'q_1' and place it to (1, 2) and then all other queens are placed safely by moving q_2 to (2, 4), q_3 to (3, 1) and q_4 to (4, 3). That is, we get the solution (2, 4, 1, 3). This is one possible solution for the 4-queens problem. For anotherpossible solution, the whole method is repeated for all partial solutions. The other solutions for 4 - queens problems is (3, 1, 4, 2) i.e.

	1	2	3	4
1			q_1	
2	q_2			
3				q_3
4		q_4		

Theimplicittreefor4-queenproblemforasolution(2,4,1,3)isasfollows:



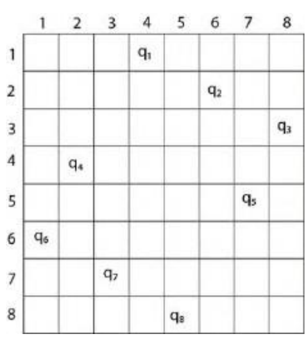
Figshowsthecompletestatespacefor4-queensproblem.But wecanusebacktrackingmethodtogenerate the necessary node and stop if the next node violates the rule, i.e., if two queens are attacking.



4-Queenssolutionspacewithnodesnumberedin DFS

Itcanbe seenthatallthe solutionstothe4queensproblemcanbe representedas4-tuples(x_1,x_2,x_3,x_4) where x_i represents the column on which queen "q_i" is placed.

Onepossiblesolutionfor8queensproblemisshowninfig:



1. Thus,thesolution**for**8-queenproblem**for**(4,6,8,2,7,1,3,5).
2. Iftwoqueensare placedatposition(i,j)and(k,l).
3. Thentheyareonsamediagonalonly**if**(i-j)= k-lori+ j=k +l.
4. Thefirstequationimpliesthatj-l=i-k.
5. Thesecondequationimpliesthatj-l=k-i.
6. Therefore,twoqueenslieontheduplicatediagonal**if**andonly**if**|j-l|=|i-k|

Place (k, i) returns a Boolean value that is true if the kth queen can be placed in column i. It tests both whether i is distinct from all previous costs x_1, x_2,... x_{k-1}andwhetherthereisnootherqueenonthesame diagonal.

Usingplace,wegiveaprecisesolutiontothenn-queens problem.

1. Place(k, i)
2. {

```
   Forj←1tok- 1
    doif(x[j]=i)
     or(Absx[j]) -i)=(Abs(j- k))
  thenreturnfalse;
   returntrue;
}
```

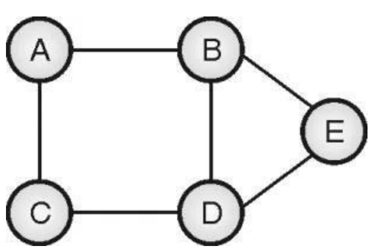
Place(k,i)returntrueifaqueencanbe placedinthekthrowandithcolumnotherwisereturnisfalse. x [] is a global array whose final k - 1 values have been set. Abs (r) returns the absolute value of r.

```
N-Queens(k,n)
{
  Fori←1ton
     doifPlace(k,i)then
  {
    x[k]←i;
    if(k==n)then
      write(x[1.....n));
    else
    N- Queens(k+1, n);
  }
}
```

HamiltonianCircuit

*The**Hamiltoniancycle**isthecycleinthegraphwhichvisitsalltheverticesingraphexactlyonceand terminates at the starting node. It may not include all the edges*

- **The**Hamiltoniancycleproblemistheproblemoffinding aHamiltoniancycleinagraphifthereexists any such cycle.
- The input to the problem is an undirected, connected graph. For the graph shown in Figure (a), a pathA–B– E– D–C–AformsaHamiltoniancycle.Itvisitsall theverticesexactlyonce,but does not visit the edges <B, D>.



- TheHamiltoniancycleproblemisalsoboth,decisionproblemandanoptimizationproblem.A decision problem is stated as, “Given a path, is it a Hamiltonian cycle of the graph?”.
- Theoptimizationproblemisstatedas,“GivengraphG,findtheHamiltoniancycleforthegraph.”
- Wecandefinetheconstraintforthe Hamiltoniancycleproblemas follows:
  - Inanypath,vertex iand (i+ 1) must be adjacent.

- 1stand(n–1)thvertexmustbeadjacent(nthofcycleistheinitialvertexitself).
- Verteximustnotappearinthefirst(i– 1)verticesofany path.

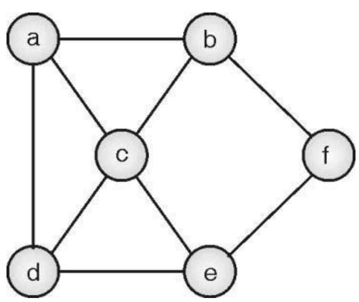
- Withtheadjacencymatrixrepresentationofthegraph,theadjacencyoftwoverticescanbeverified in constant time.

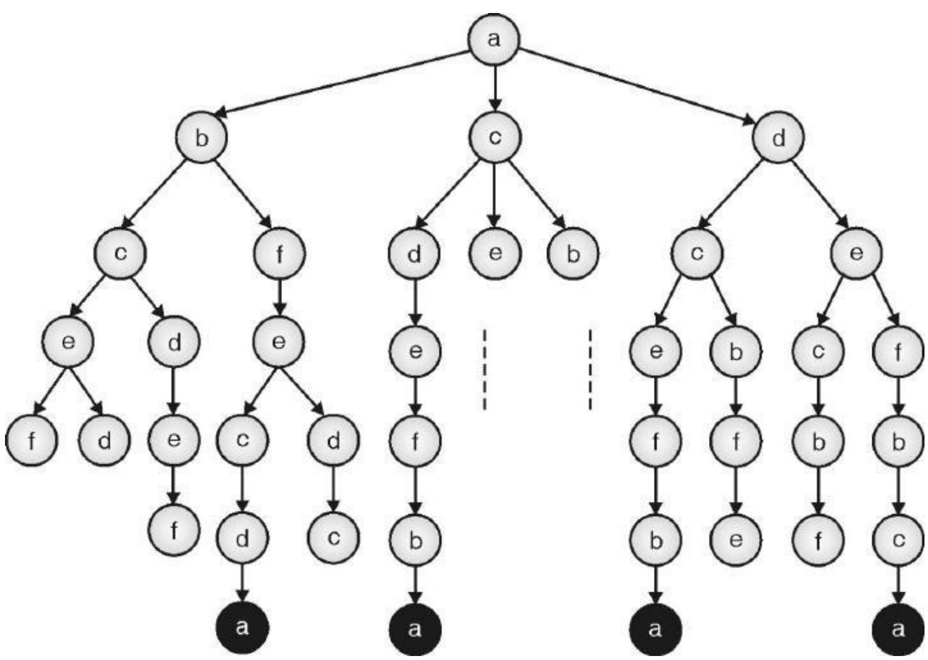
Algorithm
HAMILTONIAN(i)
//Description:SolveHamiltoniancycleproblemusingbacktracking.
//Input:Undirected,connectedgraphG=<V,E>andinitialvertexi
//Output:Hamiltoniancycle
if
FEASIBLE(i)
then
if
(i==n-1)
then
PrintV[0…n– 1]
else
j ←2
while
(j ≤ n)
do
V[i] ← j
HAMILTONIAN(i+1)
j←j+1 e**nd**
end
end
function
FEASIBLE(i)
flag←1
for
j ←1toi –1
do
if
Adjacent(Vi,Vj)
then
flag←0
end
end
if
Adjacent(Vi,Vi-1)
then
flag←1
else

flag←0
end
return
flag

ComplexityAnalysis
Lookingatthe statespacegraph,inworstcase,totalnumberofnodesintreewouldbe, $T(n) = 1 + (n - 1) + (n - 1)^2 + (n - 1)^3 + \dots + (n - 1)^{n-1}$
=frac(n−1)n−1n−2
$T(n)=O(n^n)$.Thus,theHamiltoniancyclealgorithmrunsinexponentialtime.

Example:FindtheHamiltoniancyclebyusingthebacktrackingapproachforagivengraph.





The backtracking approach uses a state-space tree to check if there exists a Hamiltonian cycle in the graph. Figure (g) shows the simulation of the Hamiltonian cycle algorithm. For simplicity, we have not explored all possible paths, the concept is self-explanatory. It is not possibleto include all the paths in the graph, so few ofthesuccessfulandunsuccessfulpathsaretracedinthe graph.BlacknodesindicatetheHamiltoniancycle.

SubsetSum Problem

SumofSubsetsProblem: Givenasetofpositiveintegers,findthe combinationofnumbersthatsumtogiven value M. Sumofsubsetsproblemisanalogoustothe**knapsackproblem**.TheKnapsackProblemtriestofillthe knapsack using a given set of items to maximize the profit. Items are selected in such a way that the total weight in the knapsack does not exceed the capacity of the knapsack. The inequality condition in the knapsack problem is replaced by equality in the sum of subsets problem.
Given the set of n positive integers, W = {w1, w2, …, wn}, and given a positive integer M, the sum of the subsetproblemcanbeformulatedasfollows(wherewiandMcorrespondtoitemweightsandknapsack capacity in the knapsack problem):

$$sum_{i=1}^{n} w_i x_i = M$$

Where,

$$x_i in 0, 1$$

Numbers are sorted in ascending order, such that $w_1 < w_2 < w_3 < \ldots < w_n$. The solution is often represented using the solution vector X. If the ithitemis included, set xito 1 else set it to 0. Ineach iteration, oneitem is tested.Iftheinclusionofanitemdoesnotviolethe constraintoftheproblem,addit.Otherwise,backtrack, removethepreviouslyaddeditem,andcontinuethe sameprocedurefor allremainingitems.Thesolutionis easily described by the state space tree. Each left edge denotes the inclusion of wi and the right edge denotes the exclusionof w_i. Any path fromthe root to the leaf forms asubset. Astate-space tree for n = 3 is demonstrated in Fig. (a).

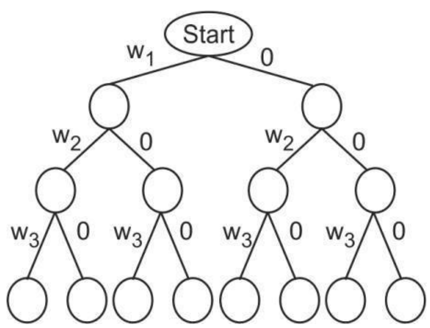


Fig.(a):Statespacetreeforn= 3

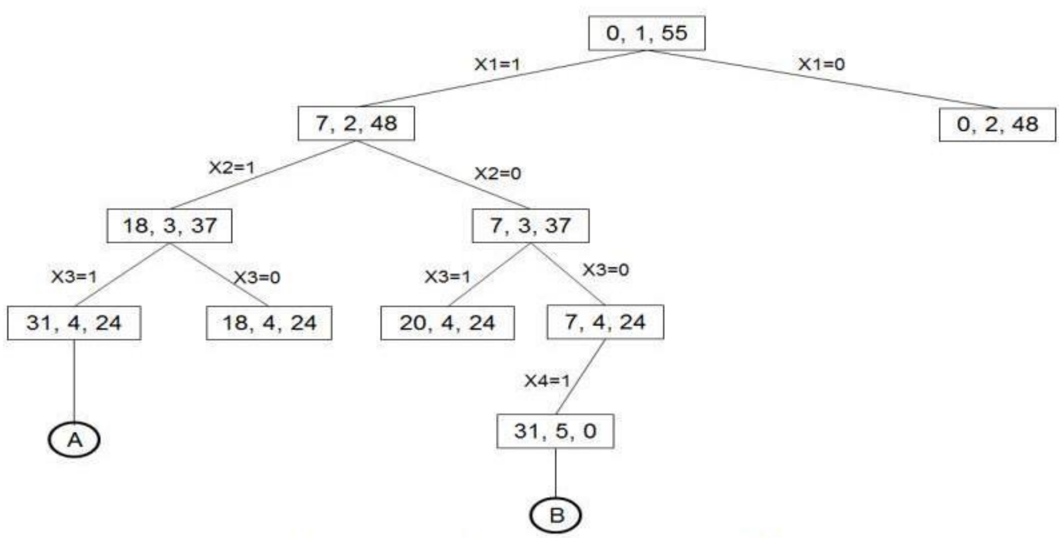
AlgorithmforSumofsubsets

Thealgorithmforsolvingthesumofsubsetsproblemusingrecursionisstatedbelow:

```
Algorithm sumofsubsets(s,k,r)
{
  X[k]:=1;
  if (s+w[k]=m) then write (x[1:k]); // subset found
  else
    if (s+w[k]+w[k+1]<=m) then
       sumofsubsets(s+w[k],k+1,r-w[k]);

   //generate right child and evaluate Bk.

   if ((s+r-w[k]>=m) and (s+w[k+1]<=m)) then
    {
      X[k]:=0;
      sumofsubsets(s,k+1,r-w[k]);
    }
}
```

Examples

Ex) n=4, (w_1,w_2,w_3,w_4)=(7,11,13,24) and m=31
Solution Vector=(x[1],x[2],x[3],x[4])



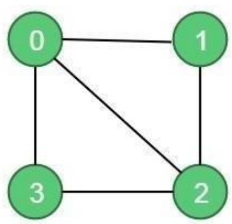
Portion of state space Tree

Solution A = {1,1,1,0}
Solution B = {1,0,0,1}

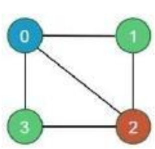
GraphColouring

In this problem,an undirected graphis given.Thereis alsoprovided m colors.Theproblem isto find if itis possibletoassignnodeswithmdifferentcolors,suchthatnotwoadjacentverticesofthegraphare ofthe same colors. If the solution exists, then display which color is assigned on which vertex.
Starting from vertex0, wewill try to assign colors one by one to different nodes. But before assigning, we havetocheckwhetherthecolorissafeornot.Acolorisnotsafewhetheradjacentverticesare containing the same color.
InputandOutput Input:
TheadjacencymatrixofagraphG(V,E)andanintegerm,whichindicatesthemaximumnumberofcolors that can be used.

0	1	1	1
1	0	1	0
1	1	0	1
1	0	1	0



Letthemaximumcolorm=3.
Output:
Thisalgorithmwillreturnwhichnodewillbe assignedwithwhichcolor.Ifthesolutionisnotpossible,it will return false.
Forthisinputtheassignedcolors are:
Node0-> color1
Node1-> color2
Node2-> color3
Node3-> color2



Algorithm
isValid(vertex,colorList,col)
Input–Vertex,colorListtocheck,andcolor,whichistryingtoassign.
Output–Trueifthecolorassigningisvalid,otherwisefalse.
Begin

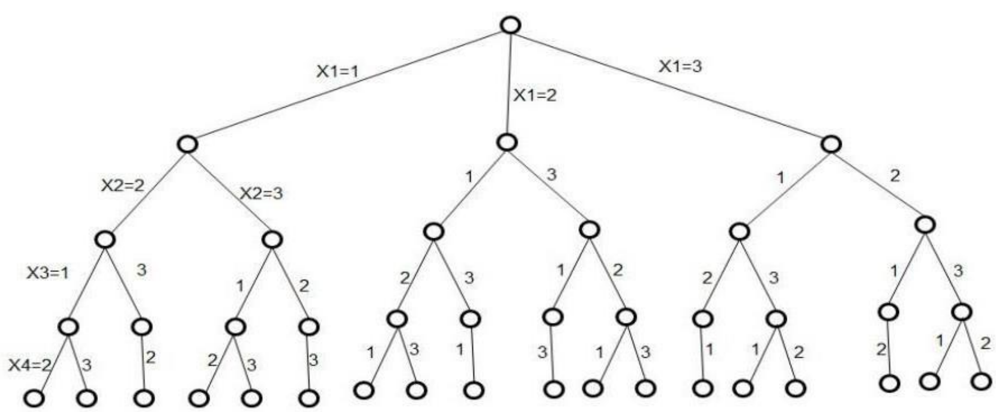
```
  forallverticesvofthegraph,do
    ifthereisanedgebetweenvandi,andcol=colorList[i],then return false
  done
  returntrue
End
```

graphColoring(colors,colorList,vertex)

Input−Mostpossiblecolors,thelistforwhichverticesarecoloredwithwhichcolor,andthestartingvertex.

Output−True,whencolorsareassigned,otherwisefalse.

```
Begin
  ifallverticesarechecked,then
    return true
  forallcolorscolfromavailablecolors,do if
    isValid(vertex, color, col), then
      addcoltothecolorListfor vertex
      ifgraphColoring(colors,colorList,vertex+1)=true,then return
        true
      removecolorforvertex done
  returnfalse
```



A 4-node graph and all possible 3-colorings

End

BranchandBound

Solving15puzzleProblem(LCBB)

The problem cinsist of 15numbered (0-15) tiles ona square box with16 tiles(one tile is blank or empty). Theobjective ofthisproblemistochange thearrangementofinitialnodetogoalnodebyusing seriesof legal moves.

TheInitialandGoalnodearrangementisshownbyfollowingfigure.

1	2	4	15
2		5	12
7	6	11	14
8	9	10	13

InitialArrangement

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	

FinalArrangement

Ininitial nodefourmovesarepossible.Usercanmoveanyoneofthetilelike2,or 3,or5,or6totheempty tile. From this we have four possibilities to move from initial node.

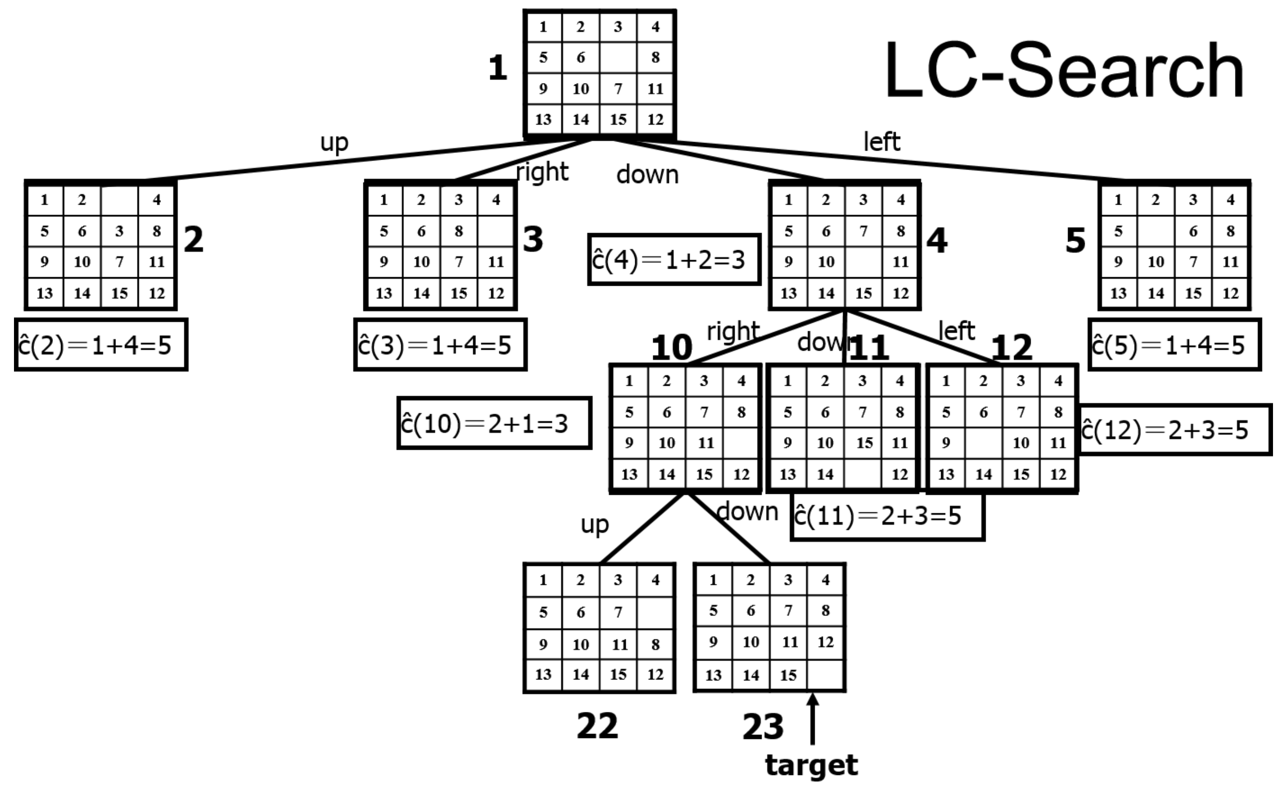
Thelegalmovesareforadjacenttilenumberisleft,right,up,down,onesatatime.

Each and every move creates a new arrangement, and this arrangement is called state of puzzle problem.

Byusingdifferentstates,astatespacetreediagramiscreated,inwhichedgesarelabeledaccordingtothe direction in which the empty space moves.

Thestatespacetreeisverylargebecauseitcanbe16!Differentarrangements.
Instatespacetree,nodesarenumberedasperthe level.Ineachlevelwemustcalculatethevalue or cost of each node by using given formula:
C(x)=f(x)+g(x),
f(x)islengthofpathfromrootorinitialnodetonodex,
g(x)isestimatedlengthofpathfromxdownwardtothegoalnode.Numberofnonblank tilenotin their correct position.
C(x)<Infinity.(initiallysetbound).
Eachtimenodewithsmallestcost isselectedforfurtherexpansiontowardsgoalnode.Thisnode become the e-node.

StateSpacetreewithnodecostisshownin diagram.



AssignmentProblem

ProblemStatement

Let'sfirstdefine ajobassignment problem.Inastandardversionofajobassignment problem,there canbe jobsand workers.Tokeepitsimple,we'retaking jobs and workersinourexample:

	Job 1	Job 2	Job 3
A	9	3	4
B	7	8	4
C	10	5	2

Wecanassignanyofthe availablejobstoanyworkerwiththeconditionthatifajobisassignedtoa worker, the other workers can’t take that particular job. We should also notice that each job has some cost associated with it, and it differs from one worker to another.

Herethemainaimistocomplete allthejobsby assigningonejobtoeachworkerinsuchawaythat the sum of the cost of all the jobs should be minimized.

BranchandBoundAlgorithmPseudocode

Nowlet’sdiscusshowtosolvethejobassignmentproblemusingabranchandboundalgorithm. Let’s see the pseudocode first:

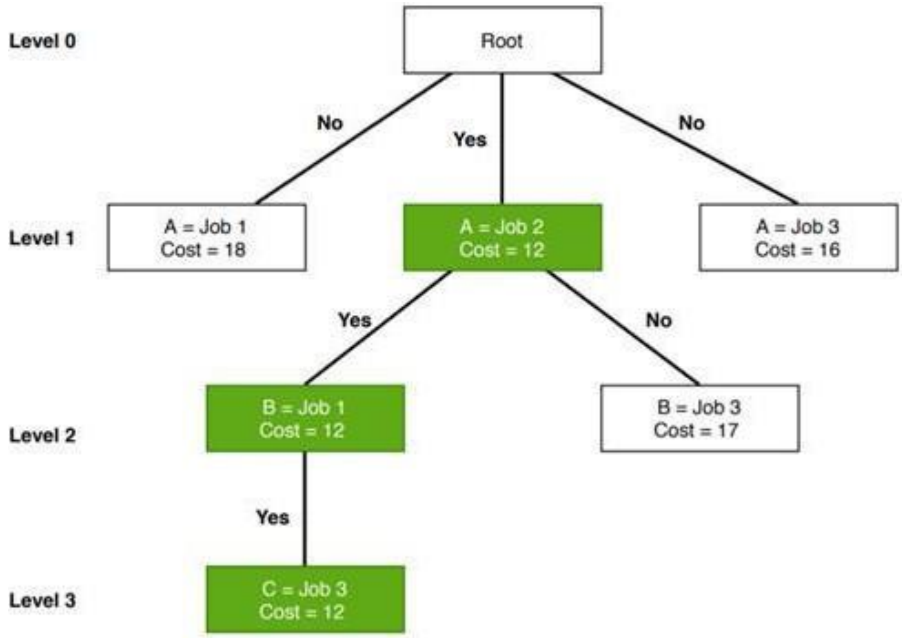
```
Algorithm 1: Job Assignment Problem Using Branch And Bound
  Data: Input cost matrix M[][]
  Result: Assignment of jobs to each worker according to optimal
          cost
  Function MinCost(M[][])
  while True do
    E = LeastCost();
    if E is a leaf node then
      print();
      return;
    end
    for each child S of E do
      Add(S);
      S → parent = E;
    end
  end
```

Here,is the input cost matrix that contains information like the number ofavailable jobs, a list of available workers, and the associated cost for each job. The function MinCost() maintains a list of active nodes. The function Leastcost()calculates the minimum cost of the active node at each level of the tree. After finding the node with minimum cost, we remove the node from the list of active nodes and return it.

We’re using the add() function in the pseudocode, which calculates the cost of a particular node and adds it to the list of active nodes.

In the search space tree, each node contains some information, such as cost, a total number of jobs, as well as a total number of workers.

Nowlet’srunthealgorithmonthesampleexamplewe’vecreated:



Advantages

Inabranchandboundalgorithm,wedon’t exploreallthe nodesinthetree.That’swhy**thetime complexity of the branch and bound algorithm is less when compared with other algorithms.**

Iftheproblemisnotlargeandifwecandothebranching inareasonableamount oftime,**itfindsan optimal solution for a given problem.**

Thebranchandboundalgorithmfindaminimalpathtoreachtheoptimalsolutionforagiven problem. **It doesn’t repeat nodes while exploring the tree.**

Disadvantages

Thebranchandbound algorithmaretime-consuming. Dependingonthe sizeofthegivenproblem, the number of nodes in the tree can be too large in the worst case.

KnapsackProblemusingbranchandbound

ProblemStatement

Weare a givenasetof*n*objectswhichhaveeachhavea value$v_i$anda weightw_i. Theobjectiveof the0/1Knapsackproblemistofindasubsetofobjectssuchthatthetotalvalueismaximized,and

Initially, we've 3 jobs available. The worker A has the option to take any of the available jobs. So at level 1, we assigned all the available jobs to the worker A and calculated the cost. We can see that when we assigned jobs 2 to the worker A, it gives the lowest cost in level 1 of the search space tree. **So we assign the job 2 to worker A and continue the algorithm. “Yes” indicates that this is currently optimal cost.**

After assigning the job 2 to worker A, we still have two open jobs. Let's consider worker B now. We're trying to assign either job 1 or 3 to worker B to obtain optimal cost.

Either we can assign the job 1 or 3 to worker B. **Again we check the cost and assign job 1 to worker B as it is the lowest in level 2.**

Finally, we assign the job 3 to worker C, and the optimal cost is 12.

thesumofweightsoftheobjectsdoesnotexceedagiventhreshold*W*.Animportant conditionhere is that one can either take the entire object or leave it. It is not possible to take a fraction of the object.

Consideranexamplewhere*n*=4,andthevaluesaregivenby {10,12,12, 18}andtheweightsgiven by {2, 4, 6, 9}. The maximum weight is given by *W = 15*. Here, the solution to the problem will be including the first, third and the fourth objects.

Here,theproceduretosolvetheproblemisasfollows are:

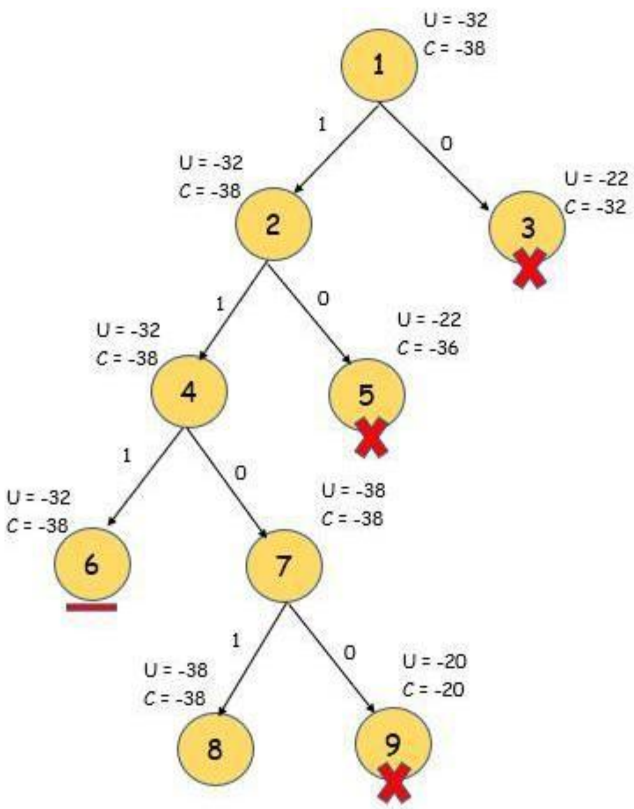
- Calculatethe costfunctionandtheUpperboundforthetwochildrenofeachnode.Here, the (i + 1)th level indicates whether the ith object is to be included or not.
- If the cost function for a given node is greater than the upper bound, then the node neednot be explored further. Hence, we can kill this node. Otherwise, calculate the upper bound forthisnode.Ifthisvalueislessthan*U*,thenreplacethe valueof*U*withthisvalue.Then,kill all unexplored nodes which have cost function greater than this value.
- Thenextnodetobecheckedafterreachingallnodesinaparticularlevelwillbe theonewith the least cost function value among the unexplored nodes.
- Whileincludinganobject,oneneedstocheckwhetherthe adding theobjectcrossedthe threshold. If it does, one has reached the terminal point in that branch, and all the succeeding objects will not be included.

TimeandSpaceComplexity

Even though this method is more efficient than the other solutions to this problem, its worst case timecomplexityisstillgivenbyO(2^n),incaseswheretheentiretreehastobeexplored.However,in its best case, only one path through the tree will have to explored, and hence its best case time complexity isgivenby O(n).Sincethis method requiresthecreationofthestatespacetree, itsspace complexity will also be exponential.

SolvinganExample

Considerthe problemwithn=4, V ={10,10,12, 18}, w={2,4,6,9}andW= 15.Here,wecalculate the initital upper bound to be U = 10 + 10 + 12 = 32. Note that the 4th object cannot be included here, since that would exceed W. For the cost, we add 3/9 th of the final value, and hence the cost function is 38. Remember to negate the values after calculation before comparison.
Aftercalculatingthecost ateachnode,killnodesthat donotneedexploring.Hence,thefinalstate space tree will be as follows (Here, the number of the node denotes the order in which the state space tree was explored):



Note here that node 3 and node 5 have been killed after updating U at node 7. Also, node 6 is not explored further, since adding any more weight exceeds the threshold. At the end, only nodes 6 and 8remain. SincethevalueofU islessfor node8,weselect thisnode.Hencethesolutionis{1,1,0,1}, and we can see here that the total weight is exactly equal to the threshold value in this case.

Travellingsalesmanproblem

- TravellingSalesmanProblem(TSP)isaninterestingproblem.Problemisdefinedas“givenn cities and distance between each pair of cities, find out the path which visits each city exactlyonceandcomebacktostartingcity, withtheconstraintofminimizing thetravelling distance.”
- TSPhasmanypracticalapplications.Itisusedinnetworkdesign,andtransportationroute design. The objective is to minimize the distance. We can start tour fromany randomcity and visit other cities in any order. With n cities, n! different permutations are possible. Exploring all paths using brute force attacks may not be useful in real life applications.

LCBBusingStaticStateSpaceTreeforTravellingSalsemanProblem

- Branchand boundisaneffectivewaytofindbetter,ifnotbest,solutioninquicktime by pruning some of the unnecessary branches of search tree.
- Itworksasfollow:

Considerdirectedweightedgraph G=(V,E,W),wherenode representscitiesand weighted directed edges represents direction and distance between two cities.

1. Initially,graphisrepresentedbycostmatrixC,where

   C_{ij}=cost ofedge,ifthereisadirectpathfromcityitocityj $C_{ij}=\infty$, if there is no direct path from city i to city j.

2. Convertcostmatrixtoreducedmatrixbysubtractingminimumvaluesfromappropriaterows and columns, such that each row and column contains at least one zero entry.

3. Findcostofreducedmatrix.Costisgivenby summationofsubtractedamountfromthecost matrix to convert it in to reduce matrix.
4. Preparestatespacetreeforthereducematrix
5. FindleastcostvaluednodeA(i.e.E-node),bycomputingreducedcostnodematrix withevery remaining node.
6. If<i,j>edgeistobeincluded,thendofollowing:
(a) SetallvaluesinrowiandallvaluesincolumnjofAto∞
(b) SetA[j,1]= ∞
(c) ReduceAagain,exceptrowsandcolumnshavingall∞entries.
7. Computethecostofnewlycreatedreducedmatrixas,
Cost=L + Cost(i, j) + r
Where,LiscostoforiginalreducedcostmatrixandrisA[i,j].
8. Ifallnodesarenotvisitedthengotostep4.
Reduction procedure is described below :

RawReduction:

MatrixMis calledreducedmatrixif eachof itsrowandcolumnhasatleastonezeroentryorentire row or entire column has ∞ value. Let M represents the distance matrix of 5 cities. M can be reduced as follow:

$M_{RowRed}=\{M_{ij}- \min\{M_{ij} | 1\le j\le n, \text{and } M_{ij}< \infty\}\}$

Consider the following distance matrix:

$$M = \begin{bmatrix} \infty & 20 & 30 & 10 & 11 \\ 15 & \infty & 16 & 4 & 2 \\ 3 & 5 & \infty & 2 & 4 \\ 19 & 6 & 18 & \infty & 3 \\ 16 & 4 & 7 & 16 & \infty \end{bmatrix}$$

Findtheminimumelementfromeachrowand subtractitfromeachcellof matrix.

$$M = \begin{bmatrix} \infty & 20 & 30 & 10 & 11 \\ 15 & \infty & 16 & 4 & 2 \\ 3 & 5 & \infty & 2 & 4 \\ 19 & 6 & 18 & \infty & 3 \\ 16 & 4 & 7 & 16 & \infty \end{bmatrix} \begin{matrix} \to 10 \\ \to 2 \\ \to 2 \\ \to 3 \\ \to 4 \end{matrix}$$

Reducedmatrixwouldbe:

$$M_{RowRed} = \begin{bmatrix} \infty & 10 & 20 & 0 & 1 \\ 13 & \infty & 14 & 2 & 0 \\ 1 & 3 & \infty & 0 & 2 \\ 16 & 3 & 15 & \infty & 0 \\ 12 & 0 & 3 & 12 & \infty \end{bmatrix}$$

Rowreductioncostisthesummationofallthevaluessubtractedfromeachrows: Row reduction cost (M) = 10 + 2 + 2 + 3 + 4 = 21

Columnreduction:

MatrixM_{RowRed}isrowreducedbut notthecolumnreduced.Matrixiscalledcolumnreducedifeach of its column has at least one zero entry or all ∞ entries.

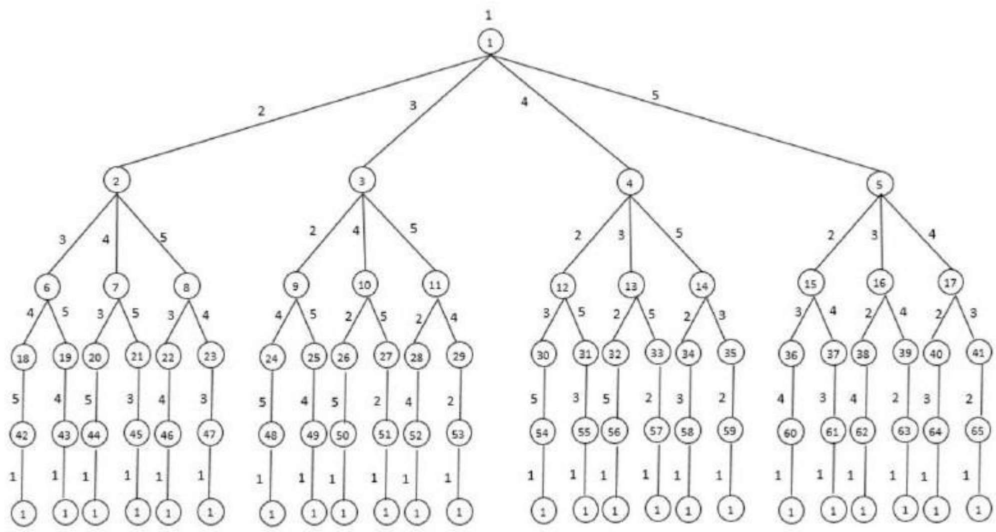
$M_{ColRed}=\{M_{ji}-\min\{M_{ji} \mid 1\le j\le n, \text{ and } M_{ji}<\infty\}\}$
Toreducedabovematrix,wewillfindtheminimumelementfromeachcolumnand subtractit from each cell of matrix.

$$M_{RowRed} = \begin{array}{|c|c|c|c|c|} \hline \infty & 10 & 20 & 0 & 1 \\ \hline 13 & \infty & 14 & 2 & 0 \\ \hline 1 & 3 & \infty & 0 & 2 \\ \hline 16 & 3 & 15 & \infty & 0 \\ \hline 12 & 0 & 3 & 12 & \infty \\ \hline \end{array}$$
$$\begin{array}{ccccc} \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\ 1 & 0 & 3 & 0 & 0 \end{array}$$

ColumnreducedmatrixM$_{ColRed}$wouldbe:

$$M_{ColRed} = \begin{array}{|c|c|c|c|c|} \hline \infty & 10 & 17 & 0 & 1 \\ \hline 12 & \infty & 11 & 2 & 0 \\ \hline 0 & 3 & \infty & 0 & 2 \\ \hline 15 & 3 & 12 & \infty & 0 \\ \hline 11 & 0 & 0 & 12 & \infty \\ \hline \end{array}$$

Eachrowand columnofM$_{ColRed}$ hasatleastonezeroentry,sothismatrixisreducedmatrix. Column reduction cost (M) = 1 + 0 + 3 + 0 + 0 = 4
Statespacetreefor5cityproblemisdepictedinFig.6.6.1.Numberwithincircleindicatestheorder in which the node is generated, and number of edge indicates the city being visited.



Example
Example:Findthesolutionoffollowingtravellingsalesmanproblemusingbranchandbound method.

Cost Matrix =

∞	20	30	10	11
15	∞	16	4	2
3	5	∞	2	4
19	6	18	∞	3
16	4	7	16	∞

Solution:

- Theprocedurefordynamicreductionisasfollow:
- Drawstatespacetreewithoptimalreductioncostatrootnode.
- Derivecost ofpathfromnodeitojbysettingallentriesinithrowandjthcolumnas∞. Set M[j][i] = ∞
- Costofcorresponding nodeNforpathitojissummationofoptimalcost +reductioncost+ M[j][i]
- Afterexploringall nodesat leveli,setnodewithminimumcost asEnodeandrepeatthe procedure until all nodes are visited.
- Givenmatrixisnotreduced. Inordertofindreducedmatrix of it,wewillfirstfindtherow reduced matrix followed by column reduced matrix if needed. We can find row reduced matrixbysubtractingminimum elementofeachrowfromeachelementofcorresponding row. Procedure is described below:
- Reduceabovecostmatrixbysubtractingminimumvaluefromeachrowandcolumn.

∞	20	30	10	11	→ 10
15	∞	16	4	2	→ 2
3	5	∞	2	4	→ 2
19	6	18	∞	3	→ 3
16	4	7	16	∞	→ 4

⇒

∞	10	20	0	1
13	∞	14	2	0
1	3	∞	0	2
16	3	15	∞	0
12	0	3	12	∞
↓	↓	↓	↓	↓
1	0	3	0	0

$= M_1'$

M‘1

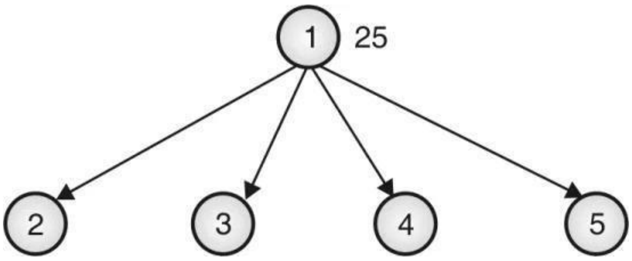
isnotreducedmatrix.Reduceitsubtractingminimumvaluefromcorrespondingcolumn.Doingthis we get,

∞	10	17	0	1
12	∞	11	2	0
0	3	∞	0	2
15	3	12	∞	0
11	0	0	12	∞

$= M_1$

Cost of $M_1 = C(1)$
= Row reduction cost + Column reduction cost
$=(10+2+2+3+4)+(1+3)=25$
This means all tours in graph has length at least 25. This is the optimal cost of the path.

State space tree



Let us find cost of edge from node 1 to 2,3,4,5.

Select edge 1-2:

Set $M_1[1][] = M_1[][2] = \infty$ Set
$M_1[2]\,[1] = \infty$
Reduce the resultant matrix if required.

$$\begin{array}{|c|c|c|c|c|l}
\hline
\infty & \infty & \infty & \infty & \infty & \rightarrow x \\
\hline
\infty & \infty & 11 & 2 & 0 & \rightarrow 0 \\
\hline
0 & \infty & \infty & 0 & 2 & \rightarrow 0 \\
\hline
15 & \infty & 12 & \infty & 0 & \rightarrow 0 \\
\hline
11 & \infty & 0 & 12 & \infty & \rightarrow 0 \\
\hline
\multicolumn{1}{c}{\downarrow} & \multicolumn{1}{c}{\downarrow} & \multicolumn{1}{c}{\downarrow} & \multicolumn{1}{c}{\downarrow} & \multicolumn{1}{c}{\downarrow} & \\
\multicolumn{1}{c}{0} & \multicolumn{1}{c}{x} & \multicolumn{1}{c}{0} & \multicolumn{1}{c}{0} & \multicolumn{1}{c}{0} &
\end{array} = M_2$$

M_2 is already reduced.
Cost of node 2 :
$C(2) = C(1) + \text{Reduction cost} + M_1[1][2]$
$=25+0+10=35$

Select edge 1-3

Set $M_1[1][] = M_1[][3] = \infty$ Set M_1
$[3][1] = \infty$
Reduce the resultant matrix if required.

$$M_1 \Rightarrow \begin{array}{|c|c|c|c|c|l}
\hline
\infty & \infty & \infty & \infty & \infty & \rightarrow x \\
\hline
12 & \infty & 11 & 2 & 0 & \rightarrow 0 \\
\hline
0 & \infty & \infty & 0 & 2 & \rightarrow 0 \\
\hline
15 & \infty & 12 & \infty & 0 & \rightarrow 0 \\
\hline
11 & \infty & 0 & 12 & \infty & \rightarrow 0 \\
\hline
\multicolumn{1}{c}{\downarrow} & \multicolumn{1}{c}{\downarrow} & \multicolumn{1}{c}{\downarrow} & \multicolumn{1}{c}{\downarrow} & \multicolumn{1}{c}{\downarrow} & \\
\multicolumn{1}{c}{11} & \multicolumn{1}{c}{0} & \multicolumn{1}{c}{x} & \multicolumn{1}{c}{0} & \multicolumn{1}{c}{0} &
\end{array} \Rightarrow \begin{array}{|c|c|c|c|c|}
\hline
\infty & \infty & \infty & \infty & \infty \\
\hline
1 & \infty & \infty & 2 & 0 \\
\hline
\infty & 3 & \infty & 0 & 2 \\
\hline
4 & 3 & \infty & \infty & 0 \\
\hline
0 & 0 & \infty & 12 & \infty \\
\hline
\end{array} = M_3$$

Cost of node 3:
$C(3) = C(1) + \text{Reduction cost} + M_1[1][3]$
$=25+11+17=53$

Selectedge1-4:

SetM_1[1][]=M_1[][4]=∞ Set

M_1 [4][1] = ∞

Reduceresultantmatrixifrequired.

$$M_1 \Rightarrow \begin{array}{c} \begin{array}{|c|c|c|c|c|} \hline \infty & \infty & \infty & \infty & \infty \\ \hline 12 & \infty & 11 & \infty & 0 \\ \hline 0 & 3 & \infty & \infty & 2 \\ \hline \infty & 3 & 12 & \infty & 0 \\ \hline 11 & 0 & 0 & \infty & \infty \\ \hline \end{array} \\ \downarrow \;\; \downarrow \;\; \downarrow \;\; \downarrow \;\; \downarrow \\ 0 \;\; 0 \;\; 0 \;\; x \;\; 0 \end{array} \begin{array}{c} \rightarrow x \\ \rightarrow 0 \\ \rightarrow 0 \\ \rightarrow 0 \\ \rightarrow 0 \\ \\ \\ \end{array} = M_4$$

MatrixM_4isalreadyreduced. Cost

of node 4:

C(4)=C(1)+Reductioncost +M_1[1][4]

=25+0+0=25

Selectedge1-5:

SetM_1[1][]=M_1[][5]=∞ Set

M_1 [5] [1] = ∞

Reducetheresultantmatrixifrequired.

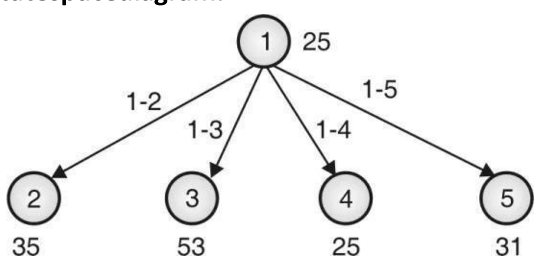
$$M_1 \Rightarrow \begin{array}{|c|c|c|c|c|} \hline \infty & \infty & \infty & \infty & \infty \\ \hline 12 & \infty & 11 & 2 & \infty \\ \hline 0 & 3 & \infty & 0 & \infty \\ \hline 15 & 3 & 12 & \infty & \infty \\ \hline \infty & 0 & 0 & 12 & \infty \\ \hline \end{array} \begin{array}{c} \rightarrow x \\ \rightarrow 2 \\ \rightarrow 0 \\ \rightarrow 3 \\ \rightarrow 0 \end{array} \Rightarrow \begin{array}{c} \begin{array}{|c|c|c|c|c|} \hline \infty & \infty & \infty & \infty & \infty \\ \hline 10 & \infty & 9 & 0 & \infty \\ \hline 0 & 3 & \infty & 0 & \infty \\ \hline 12 & 0 & 9 & \infty & \infty \\ \hline \infty & 0 & 0 & 12 & \infty \\ \hline \end{array} \\ \downarrow \;\; \downarrow \;\; \downarrow \;\; \downarrow \;\; \downarrow \\ 0 \;\; 0 \;\; 0 \;\; 0 \;\; x \end{array} = M_5$$

Costofnode5:

C(5)=C(1)+reductioncost +M_1[1][5]

=25+5+1=31

Statespacediagram:



Node4hasminimumcost forpath1-4.Wecangotovertex2,3 or5.Let'sexploreallthreenodes.

Selectpath1-4-2:(Addedge4-2)

SetM_4[1][]=M_4[4][]=M_4[] [2]=∞ Set M_4 [2]

[1]=∞

Reduceresultantmatrixifrequired.

$$M_4 \Rightarrow \begin{array}{c} \begin{array}{|c|c|c|c|c|} \hline \infty & \infty & \infty & \infty & \infty \\ \hline \infty & \infty & 11 & \infty & 0 \\ \hline 0 & \infty & \infty & \infty & 2 \\ \hline \infty & \infty & \infty & \infty & \infty \\ \hline 11 & \infty & 0 & \infty & \infty \\ \hline \end{array} \\ \downarrow \;\; \downarrow \;\; \downarrow \;\; \downarrow \;\; \downarrow \\ 0 \;\; 0 \;\; 0 \;\; x \;\; 0 \end{array} \begin{array}{c} \rightarrow x \\ \rightarrow 0 \\ \rightarrow 0 \\ \rightarrow x \\ \rightarrow 0 \\ \\ \\ \end{array} = M_6$$

MatrixM_6isalreadyreduced.
Cost of node 6:
C(6)=C(4)+Reductioncost +M_4[4][2]
=25+0+3=28
Selectedge4-3(Path1-4-3):
SetM_4[1][]=M_4[4][]= M_4[][3]=∞ Set M
[3][1]=∞
Reducetheresultantmatrixifrequired.

$$M_4 \Rightarrow \begin{array}{|c|c|c|c|c|l}\hline \infty & \infty & \infty & \infty & \infty & \rightarrow x \\ \hline 12 & \infty & \infty & \infty & 0 & \rightarrow 0 \\ \hline \infty & 3 & \infty & \infty & 2 & \rightarrow 2 \\ \hline \infty & \infty & \infty & \infty & \infty & \rightarrow \infty \\ \hline 11 & 0 & \infty & \infty & \infty & \rightarrow 0 \\ \hline \end{array} \Rightarrow \begin{array}{|c|c|c|c|c|}\hline \infty & \infty & \infty & \infty & \infty \\ \hline 12 & \infty & \infty & \infty & 0 \\ \hline \infty & 1 & \infty & \infty & 0 \\ \hline \infty & \infty & \infty & \infty & \infty \\ \hline 11 & 0 & \infty & \infty & \infty \\ \hline \multicolumn{1}{c}{\downarrow} & \multicolumn{1}{c}{\downarrow} & \multicolumn{1}{c}{\downarrow} & \multicolumn{1}{c}{\downarrow} & \multicolumn{1}{c}{\downarrow} \\ \multicolumn{1}{c}{11} & \multicolumn{1}{c}{0} & \multicolumn{1}{c}{x} & \multicolumn{1}{c}{x} & \multicolumn{1}{c}{0} \end{array} = M'_7$$

M‘7

isnotreduced.Reduceitbysubtracting11fromcolumn1.

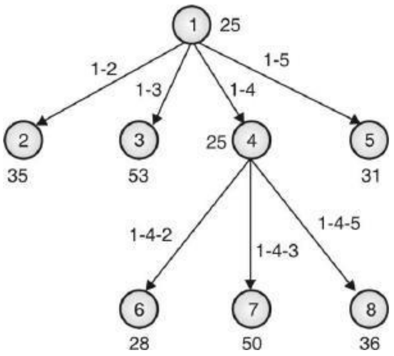
$$\therefore M'_7 \Rightarrow \begin{array}{|c|c|c|c|c|}\hline \infty & \infty & \infty & \infty & \infty \\ \hline 1 & \infty & \infty & \infty & 0 \\ \hline \infty & 1 & \infty & \infty & 2 \\ \hline \infty & \infty & \infty & \infty & \infty \\ \hline 0 & 0 & \infty & \infty & \infty \\ \hline \end{array} = M_7$$

Costofnode7:
C(7)=C(4)+Reductioncost +M_4[4][3]
=25+2+11+12=50
Selectedge4-5(Path1-4-5):

$$M_4 \Rightarrow \begin{array}{|c|c|c|c|c|l}\hline \infty & \infty & \infty & \infty & \infty & \rightarrow x \\ \hline 12 & \infty & 11 & \infty & \infty & \rightarrow 11 \\ \hline 0 & 3 & \infty & \infty & \infty & \rightarrow 0 \\ \hline \infty & \infty & \infty & \infty & \infty & \rightarrow x \\ \hline \infty & 0 & 0 & \infty & \infty & \rightarrow 0 \\ \hline \end{array} \Rightarrow \begin{array}{|c|c|c|c|c|}\hline \infty & \infty & \infty & \infty & \infty \\ \hline 1 & \infty & 0 & \infty & \infty \\ \hline 0 & 3 & \infty & \infty & \infty \\ \hline \infty & \infty & \infty & \infty & \infty \\ \hline \infty & 0 & 0 & \infty & \infty \\ \hline \multicolumn{1}{c}{\downarrow} & \multicolumn{1}{c}{\downarrow} & \multicolumn{1}{c}{\downarrow} & \multicolumn{1}{c}{\downarrow} & \multicolumn{1}{c}{\downarrow} \\ \multicolumn{1}{c}{0} & \multicolumn{1}{c}{0} & \multicolumn{1}{c}{0} & \multicolumn{1}{c}{x} & \multicolumn{1}{c}{x} \end{array} = M_8$$

MatrixM_8isreduced. Cost
of node 8:
C(8)=C(4)+Reductioncost +M_4[4][5]
=25+11+0=36
Statespacetree

Path1-4-2leadstominimumcost.Let'sfindthecostfortwopossiblepaths.



Addedge2-3(Path1-4-2-3):
SetM$_6$ [1][ ]=M$_6$ [4][] =M$_6$[2][ ]
=M$_6$ [][3]=∞
SetM$_6$[3][1]=∞
Reduceresultantmatrixifrequired.

$$M_6 \Rightarrow \begin{array}{|c|c|c|c|c|} \hline \infty & \infty & \infty & \infty & \infty \\ \hline \infty & \infty & \infty & \infty & \infty \\ \hline 0 & \infty & \infty & \infty & 2 \\ \hline \infty & \infty & \infty & \infty & \infty \\ \hline 11 & \infty & \infty & \infty & \infty \\ \hline \end{array} \begin{array}{l} \to x \\ \to x \\ \to 0 \\ \to x \\ \to 11 \end{array} \Rightarrow \begin{array}{c} \begin{array}{|c|c|c|c|c|} \hline \infty & \infty & \infty & \infty & \infty \\ \hline \infty & \infty & \infty & \infty & \infty \\ \hline 0 & \infty & \infty & \infty & 2 \\ \hline \infty & \infty & \infty & \infty & \infty \\ \hline 0 & \infty & \infty & \infty & \infty \\ \hline \end{array} \\ \begin{array}{ccccc} \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\ 0 & x & x & x & 2 \end{array} \end{array} = M_9'$$

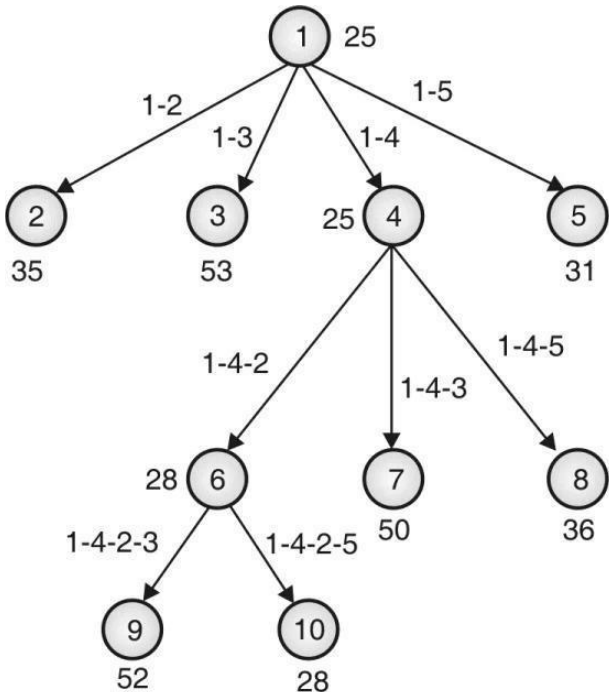
$$\therefore M_9' \Rightarrow \begin{array}{|c|c|c|c|c|} \hline \infty & \infty & \infty & \infty & \infty \\ \hline \infty & \infty & \infty & \infty & \infty \\ \hline 0 & \infty & \infty & \infty & 0 \\ \hline \infty & \infty & \infty & \infty & \infty \\ \hline 0 & \infty & \infty & \infty & \infty \\ \hline \end{array} = M_9$$

Costofnode9:
C(9)=C(6)+Reductioncost +M$_6$[2][3]
=28+11+2+11=52
Addedge2-5(Path1-4-2-5):
SetM$_6$[1][]= M$_6$[4][]=M$_6$[2][]=M$_6$[][5]=∞ Set M$_6$
[5][1] = ∞
Reduceresultantmatrixifrequired.

$$\therefore M_6 \Rightarrow \begin{array}{|c|c|c|c|c|} \hline \infty & \infty & \infty & \infty & \infty \\ \hline \infty & \infty & \infty & \infty & \infty \\ \hline 0 & \infty & \infty & \infty & \infty \\ \hline \infty & \infty & \infty & \infty & \infty \\ \hline \infty & \infty & 0 & \infty & \infty \\ \hline \end{array} = M_{10}$$

Costofnode10:
C(10)=C(6)+Reductioncost+M_6[2][5]
=28+0+0=28

Statespacetree

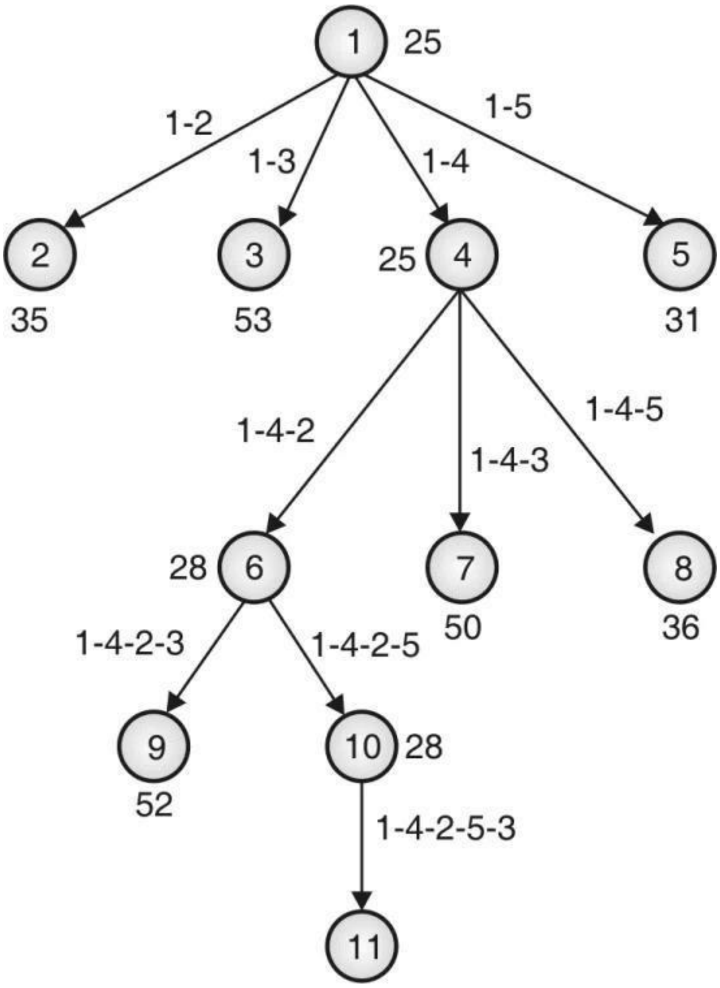


Addedge5-3(Path1-4-2-5-3):

$$\therefore\ M_{10} \Rightarrow \begin{array}{|c|c|c|c|c|}\hline \infty & \infty & \infty & \infty & \infty \\ \hline \infty & \infty & \infty & \infty & \infty \\ \hline \infty & \infty & \infty & \infty & \infty \\ \hline \infty & \infty & \infty & \infty & \infty \\ \hline \infty & \infty & \infty & \infty & \infty \\ \hline \end{array} = M_{11}$$

Costofnode11:
C(11)=C(10)+Reductioncost+M_{10}[5][3]
=28+0+0=28

Statespacetree:



Sowecanselectany oftheedge.Thusthefinalpathincludestheedges<3,1>,<5,3>,<1,4>,<4,2>, <2,5>,thatformsthe path1– 4–2 –5– 3–1.Thispathhascost of28.

UNIT5

TractableandIntractableProblems

Tractableproblemsrefertocomputationalproblemsthatcanbesolvedefficientlyusingalgorithms that can scale with the input size of the problem. In other words, the time required to solve a tractable problem increases at most polynomially with the input size.

Onthe otherhand,intractableproblemsarecomputationalproblemsforwhichnoknownalgorithm can solve them efficiently in the worst-case scenario. This means that the time required to solve an intractable problem grows exponentially or even faster with the input size.

Oneexampleofa tractableproblemis computingthesumofa list of nnumbers.The timerequired to solve this problem scales linearly with the input size, as each number can be added to a running total in constant time. Another example is computing the shortest path between two nodes in a graph,whichcanbesolvedefficientlyusingalgorithmslikeDijkstra'salgorithmortheA*algorithm.

In contrast, some well-known intractable problems include the traveling salesman problem, the knapsack problem, and the Boolean satisfiability problem. These problems are NP-hard, meaning that any problem in NP (the set of problems that can be solved in polynomial time using a non-deterministicTuringmachine)canbe reducedtotheminpolynomial time.Whileit ispossibletofind approximatesolutionstotheseproblems,thereisnoknownalgorithmthatcansolvethemexactlyin polynomial time.

In summary, tractable problems are those that can be solved efficiently with algorithms that scale wellwiththeinput size,whileintractableproblemsarethosethatcannotbesolvedefficiently inthe worst-case scenario.

ExamplesofTractableproblems

1. Sorting:Givenalistofnitems,thetaskistosorttheminascendingordescending order. Algorithms like QuickSort and MergeSort can solve this problem in O(n log n) time complexity.
2. Matrixmultiplication:GiventwomatricesAandB,thetaskistofindtheirproductC=AB. The best-known algorithm for matrix multiplication runs in O(n^2.37) time complexity, which is considered tractable for practical applications.
3. Shortest path in a graph: Given a graph G and two nodes s and t, the task is to find the shortestpathbetweensandt.AlgorithmslikeDijkstra'salgorithmandtheA* algorithmcan solvethisprobleminO(m+nlogn) timecomplexity,wheremis thenumberofedgesand n is the number of nodes in the graph.
4. Linearprogramming:Givenasystemoflinearconstraintsandalinearobjectivefunction,the task is to find the values of the variables that optimize the objective function subject to the constraints. Algorithms like the simplex method can solve this problem in polynomial time.
5. Graph coloring: Given an undirected graph G, the task is to assign a color to each node such thatno two adjacentnodeshavethesame color,using asfewcolorsas possible.The greedy algorithmcansolvethisprobleminO(n^2)time complexity,wherenisthenumberofnodes in the graph.

Theseproblemsare consideredtractablebecausealgorithmsexistthatcansolvetheminpolynomial time complexity, which means that the time required to solve them grows no faster than a polynomial function of the input size.

Examplesofintractableproblems

1. Travelingsalesmanproblem(TSP):Givenasetofcitiesandthedistancesbetweenthem,the taskis tofindtheshortestpossibleroutethatvisitseachcityexactlyonceandreturns tothe starting city. The best-known algorithms for solving the TSP have an exponential worst-case time complexity, which makes it intractable for large instances of the problem.
2. Knapsack problem:Given a setof items with weights and values, and a knapsackthat can carry amaximumweight,the taskis to find themostvaluable subsetofitemsthatcan be carriedbytheknapsack.TheknapsackproblemisalsoNP-hardand isintractableforlarge instances of the problem.
3. Boolean satisfiability problem (SAT): Given a boolean formula in conjunctive normal form (CNF),thetaskis todetermineif thereexistsanassignment oftruthvaluestothe variables thatmakestheformulatrue.TheSATproblemisoneofthemostwell-knownNP-complete problems, which means that any NP problem can be reduced to SAT in polynomial time.
4. Subsetsumproblem:Givenasetofintegersandatargetsum,thetaskistofindasubsetof the integers that sums up to the target sum. Like the knapsack problem, the subset sum problem is also intractable for large instances of the problem.
5. Graphisomorphismproblem:GiventwographsG1andG2,thetaskistodetermineifthere

1. Linearsearch:Givenalistofnitems,thetaskistofindaspecificiteminthe list.Thetime complexity of linear search is O(n), which is a polynomial function of the input size.

2. Bubble sort:Givenalistofnitems,thetaskistosorttheminascendingordescendingorder. The time complexity of bubble sort is O(n^2), which is also a polynomial function of theinput size.
3. Shortest path in a graph: Given a graph G and two nodes s and t, the task is to find the shortestpathbetweensandt.AlgorithmslikeDijkstra'salgorithmandtheA* algorithmcan solve this problem in O(m + n log n) time complexity, which is a polynomial function of the input size.
4. Maximum flow in a network: Given a network with a source node and a sink node, and capacities on the edges, the task is to find the maximum flow from the source to the sink. The Ford-Fulkerson algorithm can solve this problem in O(mf), where m is the number of edgesinthenetworkandfisthemaximumflow,whichisalsoapolynomialfunctionofthe input size.
5. Linearprogramming:Givenasystemoflinearconstraintsandalinearobjectivefunction,the task is to find the values of the variables that optimize the objective function subject to the constraints. Algorithms like the simplex method can solve this problem in polynomial time.

P(Polynomial)problems

P problems refer to problemswhere an algorithmwould take a polynomial amount of time tosolve,orwhereBig-Oisapolynomial(i.e.O(1),O(n),O(n^2),etc).Theseare problemsthat would be considered ‘easy’ to solve, and thus do not generally have immense run times.

NP(Non-deterministicPolynomial)Problems

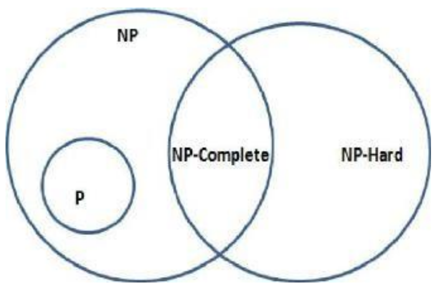
NPproblemswerealittleharderformetounderstand,but Ithinkthisiswhattheyare.In terms of solving a NP problem, the run-time would not be polynomial. It would be something like O(n!) or something much larger.

NP-HardProblems

A problem is classified as NP-Hard when an algorithm for solving it can be translated to solve*any*NPproblem.Thenwecansay,thisproblemis*at least*ashardasanyNPproblem, but it could be much harder or more complex.

NP-CompleteProblems

NP-CompleteproblemsareproblemsthatliveinboththeNPandNP-Hardclasses.This means that NP-Completeproblems can be verified in polynomial time and that any NP problem can be reduced to this problem in polynomial time.



BinPackingproblem

BinPackingprobleminvolvesassigningnitemsofdifferentweightsandbinseachofcapacity c to a bin such that number of total used bins is minimized. It may be assumed that all items have weights smaller than bin capacity.

Thefollowing4 algorithmsdependonthe orderoftheirinputs.Theypackthe itemgiven first and then move on to the next input or next item

1) NextFitalgorithm

The simplest approximate approach to the bin packing problem is the Next-Fit (NF) algorithm which is explained later in this article. The first item is assigned to bin 1. Items 2,...,narethenconsideredbyincreasingindices:eachitemisassignedtothe currentbin,if it fits; otherwise, it is assigned to a new bin, which becomes the current one.

VisualRepresentation

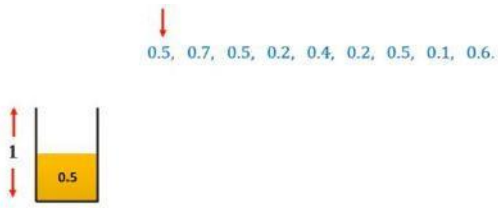
Letusconsiderthesameexampleasusedaboveandbinsofsize1

Assumingthesizesoftheitemsbe{0.5,0.7,0.5,0.2,0.4,0.2,0.5,0.1, 0.6}.

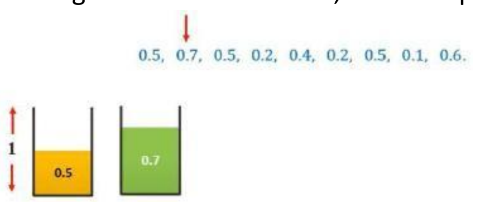
TheminimumnumberofbinsrequiredwouldbeCeil((TotalWeight)/(BinCapacity))= Celi(3.7/1) = 4 bins.

The Next fit solution (NF(I))for this instance I would be-

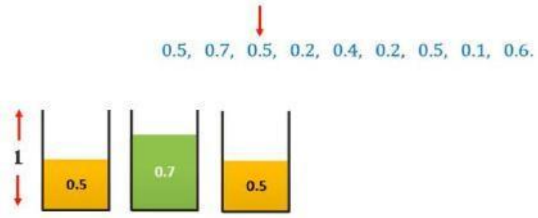
Considering0.5sizeditemfirst,wecanplaceitinthefirstbin



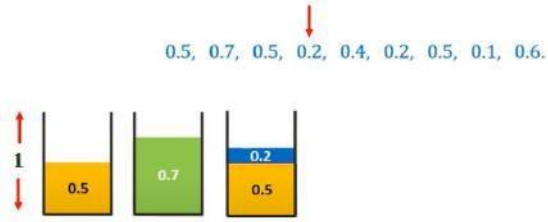
Movingontothe0.7sizeditem,wecannotplaceit inthefirstbin.Hence weplace itina new bin.



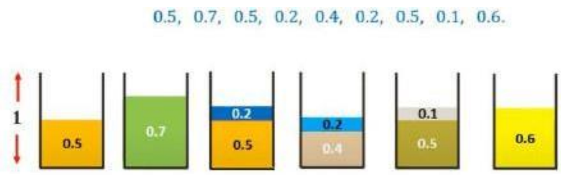
Movingontothe0.5sizeditem,wecannotplaceit inthecurrentbin.Henceweplaceit ina new bin.



Movingontothe0.2sizeditem,wecanplaceitinthecurrent(third bin)



Similarly,placingalltheotheritemsfollowingtheNext-Fitalgorithmweget-



Thusweneed6 binsasopposedtothe4 binsofthe optimalsolution.Thuswecanseethat this algorithm is not very efficient.

AnalyzingtheapproximationratioofNext-Fitalgorithm

ThetimecomplexityofthealgorithmisclearlyO(n).Itiseasytoprove that,foranyinstance I of BPP,the solution value NF(I) provided by the algorithm satisfies the bound

NF(I)<2z(I)

wherez(I)denotestheoptimalsolutionvalue.Furthermore,thereexistinstancesforwhich the ratio NF(I)/z(I) is arbitrarily close to 2, i.e. the worst-case approximation ratio of NF is r(NF) = 2.

Psuedocode

```
NEXTFIT(size[],n,c)
size[]isthearraycontaingthesizesofthe items,nisthenumberofitemsandcisthe capacity of the bin
{
   Initializeresult(Countofbins)andremainingcapacityincurrentbin. res = 0
   bin_rem=c
    Placeitemsonebyone
   for(inti=0;i <n;i++){
      //Ifthisitemcan'tfitincurrentbin if
      (size[i] > bin_rem) {
```

```
            Useanewbin
            res++
            bin_rem=c-size[i]
        }
        else
            bin_rem-=size[i];
    }
    returnres;
}
```

2) FirstFitalgorithm

A better algorithm, First-Fit (FF), considers the items according to increasing indicesandassignseachitemtothelowestindexedinitializedbinintowhichit fits; only when the current item cannot fit into any initialized bin, is a new bin introduced

VisualRepresentation

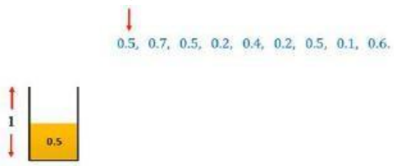
Letusconsiderthesameexampleasusedaboveandbinsofsize1

Assumingthesizesoftheitemsbe{0.5,0.7,0.5,0.2,0.4,0.2,0.5,0.1, 0.6}.

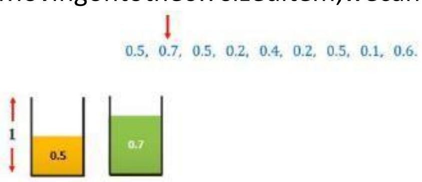
TheminimumnumberofbinsrequiredwouldbeCeil((TotalWeight)/(BinCapacity))= Celi(3.7/1) = 4 bins.

The First fit solution (FF(I))for this instance I would be-

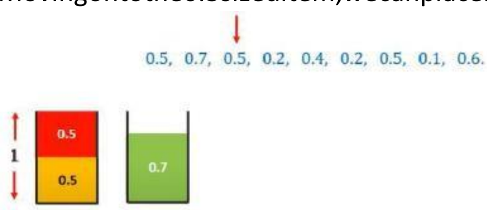
Considering0.5sizeditemfirst,wecanplaceitinthefirstbin



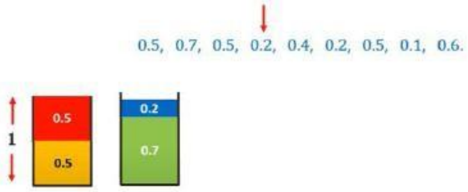
Movingontothe0.7sizeditem,wecannotplaceit inthefirstbin.Hence weplace itina new bin.



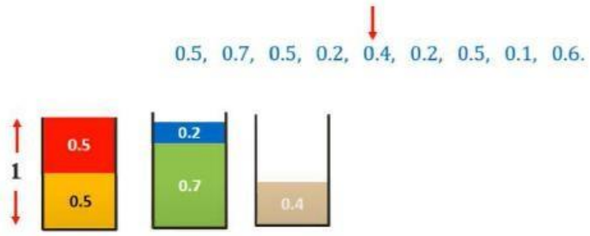
Movingontothe0.5sizeditem,wecanplaceitinthefirstbin.



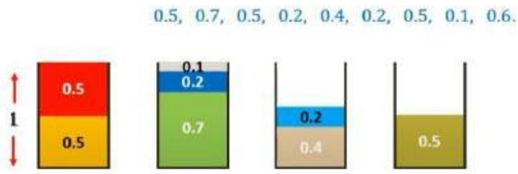
Movingontothe0.2sizeditem, wecanplaceit inthefirstbin, wecheckwiththesecondbin and we can place it there.



Movingontothe0.4sizeditem,wecannotplaceit inanyexistingbin. Henceweplaceit ina new bin.



Similarly,placingalltheotheritemsfollowingtheFirst-Fitalgorithmweget-



Thusweneed5 binsasopposedtothe4 binsofthe optimalsolutionbut ismuchmore efficient than Next-Fit algorithm.

AnalyzingtheapproximationratioofNext-Fitalgorithm

IfFF(I)istheFirst-fitimplementationforInstanceandz(I)isthemostoptimalsolution,then:

$$FF(I) \leq \frac{17}{10}\, z(I) + 2$$

for all instances I of BPP, and that there exist instances I, with $z(I)$ arbitrarily large, for which

$$FF(I) > \frac{17}{10}\, z(I) - 8.$$

Itcanbeseenthatthe FirstFitneverusesmorethan1.7*z(I)bins. SoFirst-Fitisbetterthan Next Fit in terms of upper bound on number of bins.

Psuedocode

FIRSTFIT(size[],n, c)
{
size[]isthearraycontaingthesizesofthe items,nisthenumberofitemsandcisthe capacity of the bin

/Initializeresult(Countofbins)

```
   res=0;
    Createanarraytostoreremainingspaceinbinstherecanbeatmostnbins bin_rem[n];

   Plae items one by one
   for(inti=0;i<n;i++){
      Findthefirstbinthatcanaccommodateweight[i] int j;
      for(j=0;j <res;j++){
         if (bin_rem[j] >= size[i]) {
            bin_rem[j]=bin_rem[j]-size[i];
            break;
         }
      }

      Ifnobincouldaccommodatesize[i] if
      (j == res) {
         bin_rem[res]=c-size[i];
         res++;
      }

   }
   returnres;
}
```

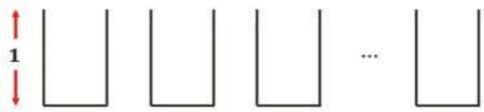
3) BestFitAlgorithm

The next algorithm, Best-Fit (BF), is obtained from FF by assigning the current itemtothefeasiblebin(ifany)havingthesmallestresidualcapacity(breaking ties in favor of the lowest indexed bin).

Simplyput,theideaistoplacesthenextiteminthe*tightest*spot.Thatis,put itinthe binso that the smallest empty space is left.

VisualRepresentation

Letusconsiderthesameexampleasusedaboveandbinsofsize1

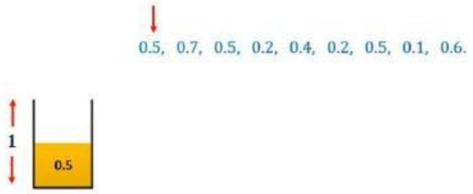


Assumingthesizesoftheitemsbe{0.5,0.7,0.5,0.2,0.4,0.2,0.5,0.1, 0.6}.

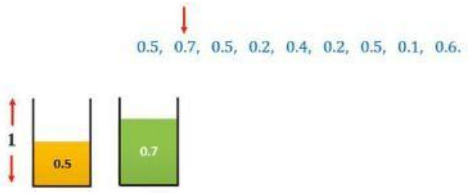
Theminimumnumberofbinsrequiredwouldbe Ceil((TotalWeight)/(BinCapacity))= Celi(3.7/1) = 4 bins.

TheFirstfitsolution(FF(I))forthisinstanceIwouldbe-

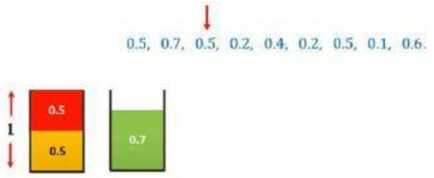
Considering0.5sizeditemfirst,wecanplaceitinthefirstbin



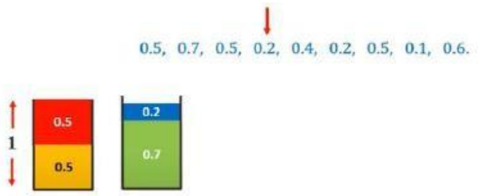
Movingontothe0.7sizeditem,wecannotplaceit inthefirstbin.Hence weplace itina new bin.



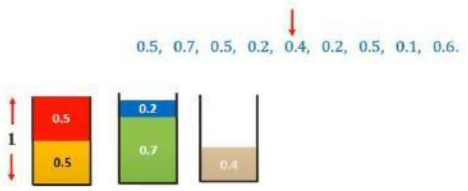
Movingontothe0.5sizeditem,wecanplaceitinthefirstbin tightly.



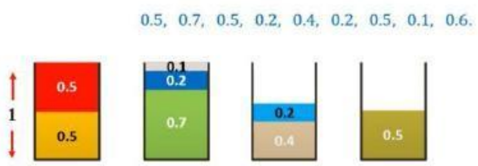
Movingontothe0.2sizeditem,wecannotplaceit inthefirstbin butwecanplace itin second bin tightly.



Movingontothe0.4sizeditem,wecannotplaceit inanyexistingbin. Henceweplaceit ina new bin.



Similarly,placingalltheotheritemsfollowingtheFirst-Fitalgorithmweget-



Thusweneed5 binsasopposedtothe4 binsofthe optimalsolutionbut ismuchmore efficient than Next-Fit algorithm.

AnalyzingtheapproximationratioofBest-Fitalgorithm

ItcanbenotedthatBest-Fit(BF),isobtainedfromFFbyassigningthecurrentitemtothe feasible bin (if any) having the smallest residual capacity (breaking ties in favour of the lowest indexed bin). BF satisfies the same worst-case bounds as FF

AnalysisOfupper-boundofBest-Fitalgorithm

Ifz(I)istheoptimalnumberofbins,thenBestFitneverusesmorethan2*z(I)-2bins. So Best Fit is same as Next Fit in terms of upper bound on number of bins.

Psuedocode

```
BESTFIT(size[],n,c)
{
size[]isthearraycontaingthesizesofthe items,nisthenumberofitemsandcisthe capacity of the
bin
  Initializeresult(Countofbins) res
  = 0;

  Createanarraytostoreremainingspaceinbinstherecanbeat mostnbins
  bin_rem[n];

  Placeitemsonebyone
  for(inti=0;i <n;i++){

    Findthebestbinthatcanaccommodateweight[i] int j;

    Initializeminimumspaceleftandindexofbestbin int
    min = c + 1, bi = 0;

    for(j=0;j <res;j++){
      if(bin_rem[j]>=size[i]&&bin_rem[j]-size[i]<min){ bi = j;
        min=bin_rem[j]-size[i];
      }
    }

    Ifnobincouldaccommodateweight[i],createanewbin if
    (min == c + 1) {
      bin_rem[res]=c-size[i];
      res++;
    }
    else
      Assigntheitemtobestbin
      bin_rem[bi] -= size[i];
  }
```

```
   returnres;
}
```

Intheofflineversion,wehaveallitemsat ourdisposalsincethestartoftheexecution.The natural solution is to sort the array fromlargest to smallest, and then apply the algorithms discussed henceforth.

NOTE: Intheonlineprogramswehavegiventhe inputsupfront forsimplicitybut itcanalso work interactively

Letuslookatthevariousofflinealgorithms

1) FirstFitDecreasing

Wefirst sortthe arrayofitemsindecreasingsizeby weight andapply first-fitalgorithmas discussed above

Algorithm

- Readtheinputsofitems
- Sortthearrayofitemsindecreasingorderbytheirsizes
- ApplyFirst-Fitalgorithm

VisualRepresentation

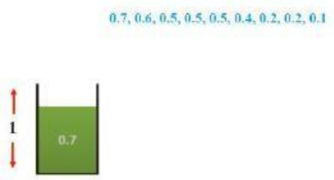
Letusconsiderthesameexampleasusedaboveandbinsofsize1

Assumingthesizesoftheitemsbe{0.5,0.7,0.5,0.2,0.4,0.2,0.5,0.1, 0.6}.

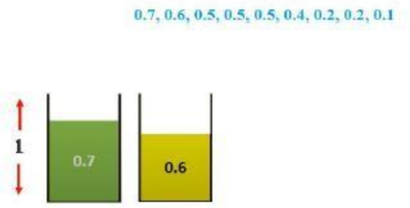
Sortingthemweget{0.7,0.6,0.5,0.5,0.5,0.4,0.2,0.2,0.1}

TheFirstfitDecreasingsolutionwould be-

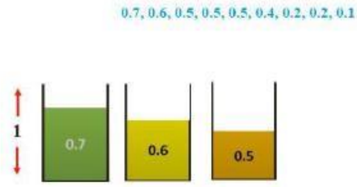
Wewillstartwith0.7andplaceitinthefirst bin



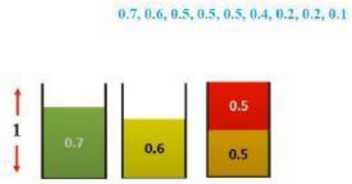
Wethenselect0.6sizeditem.Wecannotplaceitinbin1.So,weplaceitinbin2



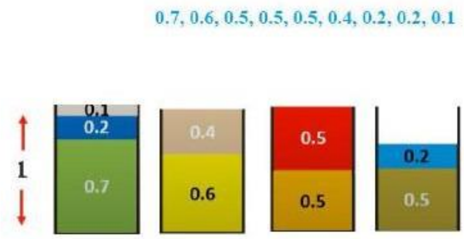
Wethenselect0.5sizeditem.Wecannotplaceitinanyexisting.So,weplaceitinbin3



Wethenselect0.5sizeditem.Wecanplace itinbin3



Doingthesameforallitems,we get.



Thusonly4binsarerequiredwhichisthesameastheoptimalsolution.

2) BestFitDecreasing

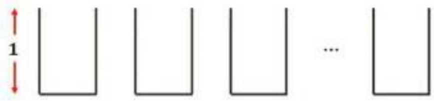
WefirstsortthearrayofitemsindecreasingsizebyweightandapplyBest-fitalgorithmas discussed above

Algorithm

- Readtheinputsofitems
- Sortthearrayofitemsindecreasingorderbytheirsizes
- ApplyNext-Fitalgorithm

VisualRepresentation

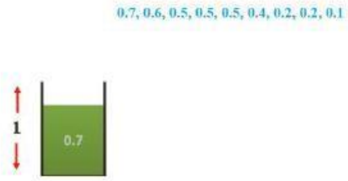
Letusconsiderthesameexampleasusedaboveandbinsofsize1



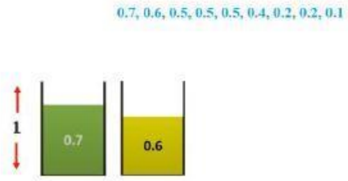
Assumingthesizesoftheitemsbe{0.5,0.7,0.5,0.2,0.4,0.2,0.5,0.1, 0.6}.
Sortingthemweget{0.7,0.6,0.5,0.5,0.5,0.4,0.2,0.2,0.1}

TheBestfitDecreasingsolutionwouldbe-

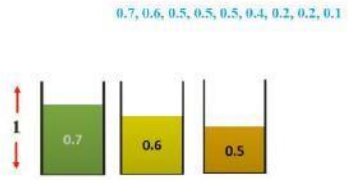
Wewillstartwith0.7andplaceitinthefirst bin



Wethenselect0.6sizeditem.Wecannotplaceitinbin1.So,weplaceitinbin2

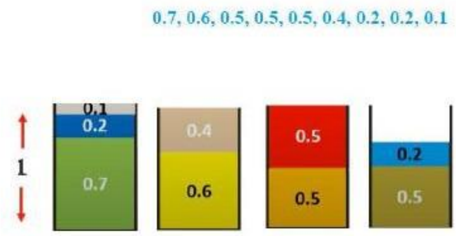


Wethenselect0.5sizeditem.Wecannotplaceitinanyexisting.So,weplaceitinbin3



Wethenselect0.5sizeditem.Wecanplace itinbin3

Doingthesameforallitems,we get.



Thusonly4binsarerequiredwhichisthesameastheoptimalsolution.

ApproximationAlgorithmsfortheTravelingSalesmanProblem

WesolvedthetravelingsalesmanproblembyexhaustivesearchinSection3.4,mentioned its decision version as one of the most well-known *NP*-complete problems in Section 11.3, and saw how its instances canbe solved by a branch-and-bound algorithm in Section 12.2. Here, we consider several approximation algorithms, a small sample of dozens of such algorithms suggested over the years for this famous problem.

But first let us answer the question of whether we should hope to find a polynomial-time approximation algorithm with a finite performance ratio on all instances of the traveling salesmanproblem.Asthefollowingtheorem[Sah76]shows,the answerturnsouttobeno, unless ***P*** = ***N P*** .

THEOREM1 If***P***!=*NP*,thereexistsno***c***-approximationalgorithmforthetravelingsalesman problem, i.e., there exists no polynomial-time approximation algorithm for this problem so that for all instances

$$f(s_a) \le cf(s^*)$$

for some constant c.

Nearest-neighbouralgorithm

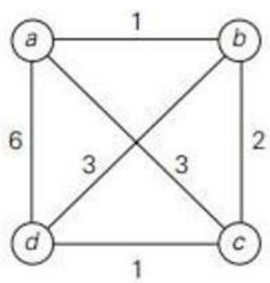
Thefollowingwell-knowngreedyalgorithmisbasedonthe***nearest-neighbor***heuristic: always go next to the nearest unvisited city.

Step1 Chooseanarbitrarycityasthestart.

Step 2 Repeatthe followingoperationuntilallthecitieshavebeenvisited:gotothe unvisited city nearest the one visited last (ties can be broken arbitrarily).

Step3 Returntothestartingcity.

EXAMPLE1 Fortheinstancerepresentedbythe graphinFigure 12.10,with***a***asthestarting vertex, the nearest-neighbor algorithm yields the tour (Hamiltonian circuit)s_a:***a***− ***b***−***c*** −***d***−***a***oflength10.



Theoptimalsolution,ascanbeeasilycheckedbyexhaustivesearch,isthe tours^*: ***a***−***b***−***d*** −***c***−***a***oflength8.Thus,theaccuracyratioofthisapproximationis

$$r(s_a) = \frac{f(s_a)}{f(s^*)} = \frac{10}{8} = 1.25$$

(i.e., tour s_a is 25% longer than the optimal tour s^*).

Unfortunately,exceptforitssimplicity,notmanygoodthingscanbesaidaboutthenearest-neighbor algorithm. In particular, nothing can be said in general about the accuracy of solutions obtained by this algorithm because it can force us to traverse a very long edge on the last leg of the tour.Indeed, if we change the weight of edge ***(a, d)*** from6 to an arbitrary large number $w \geq 6$ in Example 1, the algorithm will still yield the tour $a - b - c - d - a$ of length $4 + w$, and the optimal solution will still be $a - b - d - c - a$ of length 8. Hence,

$$r(s_a) = \frac{f(s_a)}{f(s^*)} = \frac{4 + w}{8},$$

whichcanbemadeaslarge aswewishby choosinganappropriatelylargevalueof$w$. Hence, $R_A = \infty$ for this algorithm (as it should be according to Theorem 1).

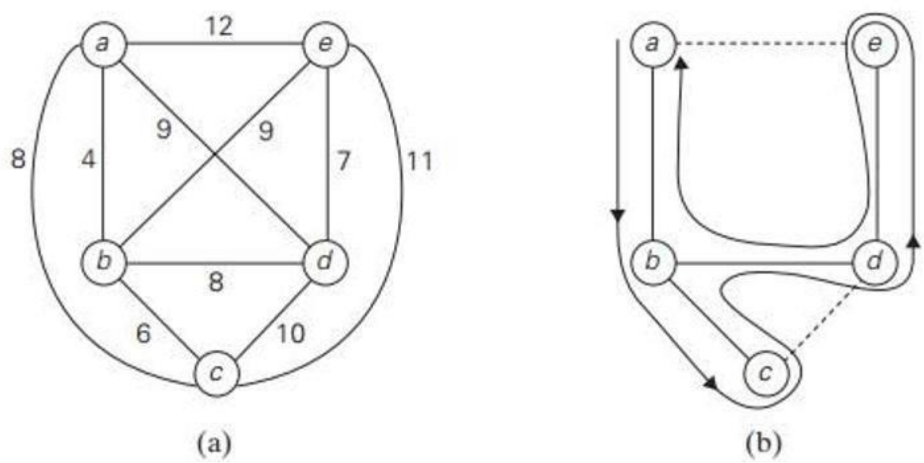
Twice-around-the-treealgorithm

Step1 Constructaminimumspanningtreeofthegraphcorrespondingtoagiveninstanceof the traveling salesman problem.

Step 2 Startingatanarbitraryvertex,performawalkaroundtheminimumspanning tree recording all the vertices passed by. (This can be done by a DFS traversal.)

Step3 ScanthevertexlistobtainedinStep2andeliminatefromit allrepeatedoccurrences of the same vertex except the starting one at the end of the list. (This step is equivalent to making shortcuts in the walk.) The vertices remaining on the list will form a Hamiltonian circuit, which is the output of the algorithm.

EXAMPLE 2 Let us apply this algorithm to the graph in Figure 12.11a. The minimum spanningtreeofthisgraphismadeupofedges***(a,b),(b,c)***,***(b, d)***,and***(d, e).***Atwice-



around-the-treewalkthatstartsandendsat*a*is

a,b,c,b,d,e,d,b,a.

Eliminatingthesecond***b***(ashortcutfrom***c***to***d***),the second***d***,andthethird***b***(ashortcut from ***e*** to ***a***) yields the Hamiltonian circuit

a,b,c,d,e,a

oflength39.

ThetourobtainedinExample2isnotoptimal.Althoughthatinstanceissmallenoughtofind an optimal solution by either exhaustive search or branch-and-bound, we refrained from doing so to reiterate a general point. As a rule, we do not know what the length of an optimaltouractually is,and thereforewecannotcomputetheaccuracyratio $f(s_a)/f(s^*)$. For the twice-around-the-tree algorithm, we can at least estimate it above, provided the graphis Euclidean.

Fermat'sLittleTheorem:

Ifnisaprimenumber,thenforeverya,1<a<n-1,

$a^{n-1} \equiv 1 (\mathrm{mod}\, n)$ OR

$a^{n-1} \% n = 1$

Example:Since 5isprime,$2^4 \equiv 1(\mathrm{mod}\,5)$ [or $2^4 \% 5 = 1$],

$3^4 \equiv 1(\mathrm{mod}\,5)$ and $4^4 \equiv 1(\mathrm{mod}\,5)$

Since7isprime,$2^6 \equiv 1(\mathrm{mod}\,7)$,

$3^6 \equiv 1(\mathrm{mod}\,7)$, $4^6 \equiv 1(\mathrm{mod}\,7)$

$5^6 \equiv 1(\mathrm{mod}\,7)$ and $6^6 \equiv 1(\mathrm{mod}\,7)$

Algorithm

1) Repeatfollowingktimes:

   a) Pickarandomlyinthe range[2,n-2]

   b) Ifgcd(a,n)≠1,thenreturn false

   c) Ifa^{n-1}≢1(modn),thenreturnfalse

2) Returntrue[probablyprime].

Unlikemergesort,we don’tneedtomerge thetwosortedarrays.ThusQuicksortrequires lesser auxiliary space than Merge Sort, which is why it is often preferred to Merge Sort. UsingarandomlygeneratedpivotwecanfurtherimprovethetimecomplexityofQuickSort.

<u>Algorithmforrandompivoting</u>

partition(arr[],lo,hi)

```
  pivot=arr[hi]
  i = lo     //placeforswapping
  for j := lo to hi – 1 do
     if arr[j] <= pivot then
        swaparr[i]witharr[j] i
        = i + 1
  swaparr[i]witharr[hi] return
  i
partition_r(arr[],lo,hi)
  r=RandomNumberfromlotohi Swap
  arr[r] and arr[hi]
  returnpartition(arr,lo,hi)
quicksort(arr[], lo, hi)
  iflo<hi
     p=partition_r(arr,lo,hi)
     quicksort(arr, lo , p-1)
     quicksort(arr, p+1, hi)
```

Findingkthsmallestelement

ProblemDescription: GivenanarrayA[]ofnelementsandapositiveintegerK,findtheKth smallest element in the array. It is given that all array elements are distinct.

ForExample:

Input :A[]={10,3,6,9,2,4,15,23},K=4

Output:6

Input:A[]={5,-8,10,37,101,2,9},K=6

Output:37

Quick-Select:Approachsimilartoquicksort

Thisapproachissimilartothe quicksortalgorithmwhereweusethepartitionontheinput array recursively. But unlike quicksort, which processes both sides of the array recursively, this algorithm works on only one side of the partition. We recur for either the left or right side according to the position of pivot.

SolutionSteps

1. PartitionthearrayA[left..right]intotwosubarraysA[left..pos]andA[pos+1..right]such that each element of A[left .. pos] is less than each element of A[pos + 1 .. right].
2. ComputesthenumberofelementsinthesubarrayA[left..pos]i.e.count=pos-left+1
3. if(count==K),thenA[pos]istheKthsmallestelement.
4. OtherwisedeterminesinwhichofthetwosubarraysA[left..pos-1]andA[pos+1 ..right] the Kth smallest element lies.

- If(count>K)thenthedesiredelementliesontheleftsideofthe partition

- If (count < K), then the desired element lies on the right side of the partition. Since we alreadyknowivaluesthataresmallerthanthekthsmallestelementofA[left..right],the desired element is the (K - count)th smallest element of A[pos + 1 .. right].
- Basecaseisthescenarioofsingleelementarrayi.eleft==right.returnA[left]orA[right].

Pseudo-Code

```
//Originalvalueforleft=0andright=n-1
intkthSmallest(intA[],intleft,intright,intK)
{
    if(left== right)
       returnA[left]
  intpos=partition(A,left,right)
  count = pos - left + 1
  if(count==K)
      returnA[pos]
    elseif(count>K)
       returnkthSmallest(A,left,pos-1,K)
  else
       returnkthSmallest(A,pos+1,right,K-i)
}

intpartition(intA[],intl,intr)
{
  intx=A[r]
  inti=l-1
  for (j=ltor-1)
  {
    if(A[j]<= x)
    {
      i = i + 1
      swap(A[i],A[j])
    }
  }
  swap(A[i+1],A[r])
  returni+1
}
```

ComplexityAnalysis

TimeComplexity:Theworst-case timecomplexityforthisalgorithmisO(n^2),but itcanbe improved if we choose the pivot element randomly. If we randomly select the pivot, the expected time complexity would be linear, **O(n).**