CMR COLLEGE OF ENGINEERING & TECHNOLOGY

(UGC Autonomous)

Approved by AICTE & Permanently Affiliated to JNTUH, Hyderabad

Kandlakoya, Medchal Road, Hyderabad -501401

Department of Computer Science and Engineering

DAA LAB Manual

Week-1:
1. Implement greedy algorithm for job sequencing with deadlines.
2. Implement greedy algorithm for Huffman codes.
Week-2:
1. Implements Prim's algorithm to generate minimum cost spanning tree.
2. Implements Kruskal's algorithm to generate minimum cost spanning tree
Week-3:
1. Implement Floyd's algorithm for the all pairs shortest path problem.
2. Implement Dijkstra's algorithm for the Single source shortest path problem
Week-4:
1. Implement Dynamic Programming algorithm for the 0/1 Knapsack problem.
2. Implement Dynamic Programming algorithm for the longest common subsequence.
Week-5:
1. Implement Dynamic Programming algorithm for the Optimal Binary Search Tree Problem.
Week-6:
1. Implement Dynamic Programming algorithm for the TSP Problem.
Week-7:
1. Implement Dynamic Programming algorithm for the matrix chain multiplication Problem.
Week-8:
1. Implement backtracking algorithm for the N-queens problem.

Week-9:
1. Implement the backtracking algorithm for the sum of subsets problem.
Week10:Implement the back tracking algorithm for Hamiltonian circuits problem.
Week -11: Implement LC branch and bound algorithm for the TSP Problem
Week-12: Implement LC branch and bound algorithm for the Knapsack problem

Week1:

1. **Implement greedy algorithm for job sequencing with deadlines**

   Problem Explain:

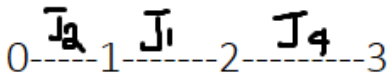
Explanation: Is an application of Greedy method in which we have 'n' jobs, each job associated with a profits P>=0 and deadline d>=0 .for any job 'I' the profit is earn if the job is completed by its deadline. To complete the job one has to process the job on a machine for I unit of time. Only one machine is available for processing jobs. The feasible solution for this job is a subset of jobs such that each job jobs in this subsets can be completed by its deadline. The value of feasible solution's' is the sum of profits of the jobs in 'j'. Therefore an optimal solution is a feasible solution with max profit.

Calculate job sequence using greedy method where n=5 (J1,J2,J3,J4,j5)=(20,15,10,5,1)and (d1,d2,d3,d4,d5)=(2,2,1,3,3) calculate the optimal solution.

Sol:

Object	J1	J2	J3	J4	J5
Profits	20	15	10	5	1
Deadline	2	2	1	3	3

Maxdeadline =3



Job consider	Slot assigned	solution	profits
-	-	0	-
J2	[0,1]	J2	20
J1	[0,1][1,2]	J2J1	20+15=35
J3not considered	[0,1][1,2]	J2J1	20+15=35
J4	[0,1][1,2][2,3]	J1J2J4	20+15+5=40
J5 not considered	[0,1][1,2][2,3]	J1J2J4	20+15+5=40

The maximum profit gained after executing 3 jobs out of 5 is 40

Execut the the following job sequence with the given deadlines and earn profits.

Jobs : J1 J2 J3 J4
Profits : 100 10 15 27
Deadline: 2 1 2 1

Feasible sol	processing seq	value
(1,2)	2,1	10+100=110
(1,3)	1,3 or 3,1	100+15=115
(1,4)	4,1	27+100=127
(2,3)	2,3	10+15=25
(3,4)	4,3	27+15=42
1	1	100
2	2	10
3		
4	3	15
	4	27

Maximu prof =127

```
Algorithm Greedyjob(d,j,n)
{
J={1}
for  i=2 to n do
{
 if(all jobs in JU{i} can be completed by the deadline)
  then
    J=JU{i}
}
}
```

2. Implement greedy algorithm for Huffman codes.

Problem Explanation:

Huffman codes are used in the communication theory and in the field of data compression. For communication to happen the related symbols or message must be encoded using any coding techniques. This task is carried out by an encoder. The encoder assigns a unique address to all the symbols and then a message is transmitted across the channel as a series of 0's and 1's, the message is received across the channel and decoded using the decoder block. The decoder message is received by the receiver. Therefore, the code plays an important role in communication. And this code is named after David Huffman who designed this code in 1951. Here the size of the data is reduced. And is of two types namely.

1. Fixed length coding and
2. Variable length coding:

Suppose if I have the following data

A B B C D B C C D A A B B E E E B E A B

Where total 20 characters each holding 8 bits hence it requires 20*8 = 160 bits, so without any compression to transfer the data which takes 160 bits.

1. Fixed length coding: we have 5 characters and each character has its own ASCII value and which takes 8 bits these are as follows :

A----------65

B----------66

C----------67

D----------68

E----------69

If we have 2 characters then we can represent in 2^2 combinations but here we have 5 characters so we can represent it in 2^3 combinations that is

000 --A

001-- B

010--C

011--D

100--E

101

110

111

Therefore in fixed length the total data required is 5*8+5*3+20*3=115 bits .

2. Variable length code:

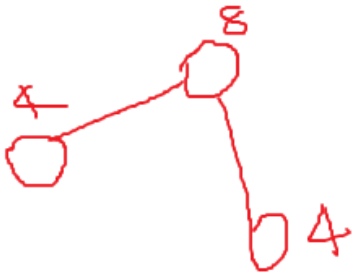
Character	Frequency	Value
A	4	01
B	7	11
C	3	101
D	2	100
E	4	00

So the data required space is 4*2+7*2+3*3+2*3+4*2 = 45 * 5*8+12=97bits.

Therefore total 97 bits are required in variable length. And now these frequencies are represented in increasing order in the queue that is 2 3 4 4 7 then take two minimum values and construct the tree where the right node is greater than left node and root is addition of these two nodes then insert the resultant node into the queue and repeat it until all nodes are over in the queue.

2 3 4 4 7 The min elements are 2 and 3 then the tree is

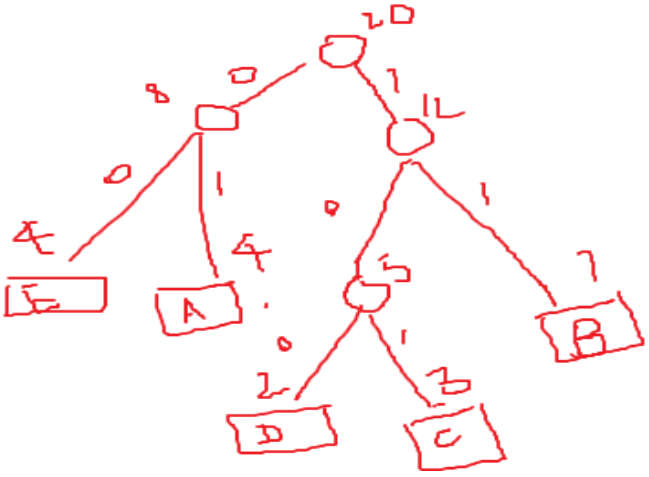
Now the queue is 5 4 4 7 after sorting 4 4 5 7 again construct the new node



Now the elements are 8 5 7 then the
increasing order is 5 7 8 repeat same as above

Now the elements are 12 and 8 then sort that 8 12 do the same as above

Final tree is as follows



The left of root is represented with 0 and its right is with 1

So E Is from root node to child node is 00

A Is from root node to child node is 01

D Is from root node to child node is 100

C Is from root node to child node is 101

B Is from root node to child node is 11

So while sending the data from source to destination user can send table shown above or tree for the decoding encoded data

Note: No code is prefix of another code. (Which means different characters should not contain same code).

Algorithm Huffman code( )

```
While (min_queue.length > 1) {
    z = new node
    z.left = min_queue.extract( )
    z.right = min_queue.extract( )
    z.freq = z.left.freq + z.right.freq
    min_queue. insert(z)
}
  return min_queue.extract
```

Week-2:

1. **Implements Prim's algorithm to generate minimum cost spanning tree.**

Problem Explanation:

Prim's Algorithm is a famous greedy algorithm.And is vertex based algorithm.

It is used for finding the Minimum Spanning Tree (MST) of a given graph.

Step-01: Randomly choose any vertex.

The vertex connecting to the edge having least weight is usually selected.

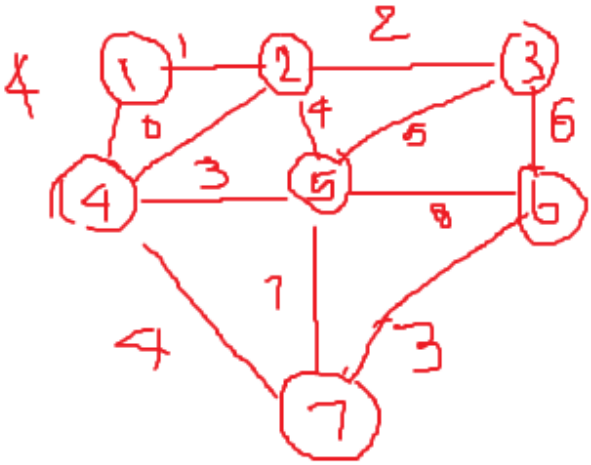
Step-02: Find all the edges that connect the tree to new vertices.

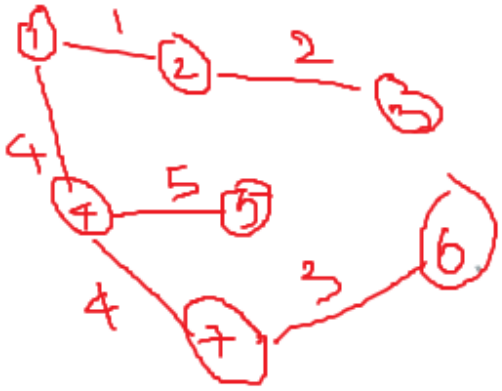
Find the least weight edge among those edges and include it in the existing tree.

If including that edge creates a cycle, then reject that edge and look for the next least weight edge.

Step-03: Keep repeating step-02 until all the vertices are included and Minimum Spanning Tree (MST) is obtained.

Construct the minimum spanning tree (MST) for the given graph using Prim's Algorithm-





Initial	{u,v}	B
	-	{1}
	{1,2}	{1,2}
	{2,3}	{1,2,3}
	{1,4}	{1,2,3,4}
	{4,5}	{1,2,3,4,5}
	{4,7}	{1,2,3,4,5,7}
	{7,6}	{1,2,3,4,5,6,7}

Minimum cost is=1+2+4+5+4+3=19

Algorith Prims(Q=<N,A>: graph Set of edges

{

T←Φ

B←{an arbitrary member of N)

While(B notequal N) do

{
Find P={u,v} of min length such that u € B and v €(N-B)
T←T U { e }
B←B U {v}
}
return T
}

2. Implements Kruskal's algorithm to generate minimum cost spanning tree

Problem Explanation:

Step-01:

Sort all the edges from low weight to high weight.

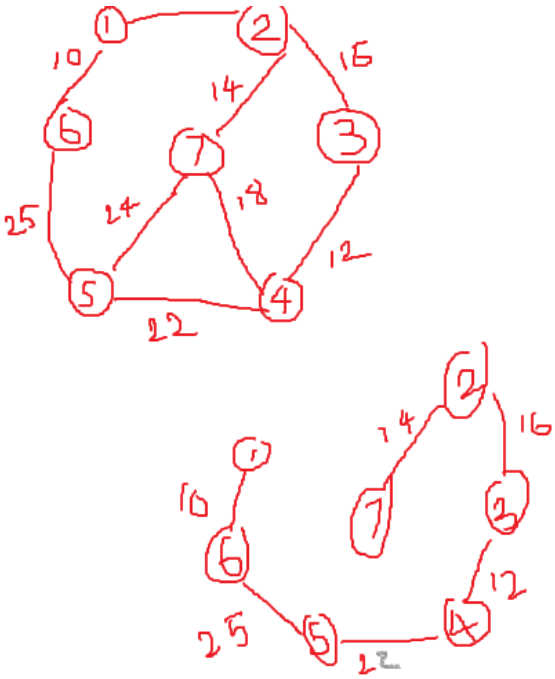
Step-02:

Take the edge with the lowest weight and use it to connect the vertices of graph.

If adding an edge creates a cycle, then reject that edge and go for the next least weight edge.

Step-03:

Keep adding edges until all the vertices are connected and a Minimum Spanning Tree (MST) is obtained.



Edge	copmarision
	1 . 2 3 4 5 6 7
{1,6}	{1,6} 2 3 4 5 7
{3,4}	{1,6}, {3,4} 2 5 7
{2,7}	{1,6} {3 ,4 }, { 2 ,7 } 5
{2,3}	{1, 6} , { 2 , 3 ,4 , 7 } 5
{4 ,5 }	{ 1 ,6 } {2 3, 4, 5 , 7 }
{5, 6}	{ 1, 2 ,3 , 4 , 5, ,6 , 7 }

The minimu cost is =10+25+22+12+16+14=99

Algorithm kruskals((G=<N,A> graph):setofedges

//initialization

{

N← no of nodes in N

T← Φ

repeat

e= {u,v}

u comp = find(u)

v comp= find(v)

if(u comp not= v comp)then

merge(u comp, v comp)

T←T U {e}

Until T contain(n-1) edges

return T

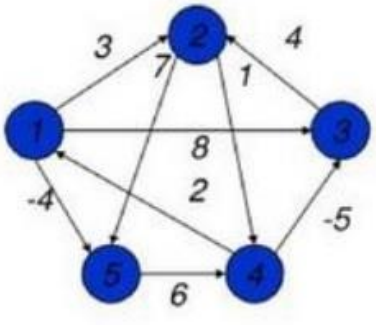
}

Week-3:

1. Implement Floyd's algorithm for the all pairs shortest path problem.

   Problem Explanation:

The all pair shortest path algorithm is also known as Floyd-Warshall algorithm is used to find all pair shortest path problem from a given weighted graph. As a result of this algorithm, it will generate a matrix, which will represent the minimum distance from any node to all other nodes in the graph.



$$\begin{bmatrix} 0 & 1 & -3 & 2 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{bmatrix}$$

At first the output matrix is same as given cost matrix of the graph. After that the output matrix will be updated with all vertices k as the intermediate vertex.

The time complexity of this algorithm is O(V3), here V is the number of vertices in the graph.

Input − The cost matrix of the graph.

0 3 6 ∞ ∞ ∞ ∞

3 0 2 1 ∞ ∞ ∞

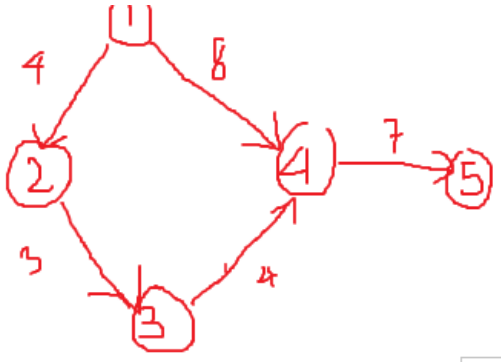
6 2 0 1 4 2 ∞
∞ 1 1 0 2 ∞ 4
∞ ∞ 4 2 0 2 1
∞ ∞ 2 ∞ 2 0 1
∞ ∞ ∞ 4 1 1 0
Output − Matrix of all pair shortest path.
0 3 4 5 6 7 7
3 0 2 1 3 4 4
4 2 0 1 3 2 3
5 1 1 0 2 3 3
6 3 3 2 0 2 1
7 4 2 3 2 0 1
7 4 3 3 1 1 0

Algorithm

```
{
  for k := 0 to n, do
    for i := 0 to n, do
      for j := 0 to n, do
        if cost[i,k] + cost[k,j] < cost[i,j], then
          cost[i,j] := cost[i,k] + cost[k,j]
        done
      done
    done
    display the current cost matrix
}
```

2.Implement Dijkstra's algorithm for the Single source shortest path problem
Problem Explanation:

Dijkstra Algorithm is a very famous greedy algorithm.
It is used for solving the single source shortest path problem.
It computes the shortest path from one particular source node to all other remaining nodes of the graph



Iterator	S	Visited	Distance				
-	-	-	∝	∝	∝	∝	∝
1	{1}	1	0	4	∝	8	∝
2	{1 , 2}	2	0	4	7	8	∝
3	{1 , 2 , 3}	3	0	4	7	8	∝
4	{1 ,2, 3,4}	4	0	4	7	8	15

Week-4:

1. Implement Dynamic Programming algorithm for the 0/1 Knapsack problem.

Problem Explanation:

Step-01:

- Draw a table say 'T' with (n+1) number of rows and (w+1) number of columns.
- Fill all the boxes of 0^{th} row and 0^{th} column with zeroes as shown-

	0	1	2	3	.....	W
0	0	0	0	0	.....	0
1	0					
2	0					
.....	.....					
n	0					

T-Table

Step-02:

Start filling the table row wise top to bottom from left to right.

Use the following formula-

$$T(i, j) = \max \{ T(i-1, j), value_i + T(i-1, j - weight_i) \}$$

Here, T(i , j) = maximum value of the selected items if we can take items 1 to i and have weight restrictions of j.

- This step leads to completely filling the table.
- Then, value of the last box represents the maximum possible value that can be put into the knapsack.

Step-03:

To identify the items that must be put into the knapsack to obtain that maximum profit,

- Consider the last column of the table.
- Start scanning the entries from bottom to top.
- On encountering an entry whose value is not same as the value stored in the entry immediately above it, mark the row label of that entry.
- After all the entries are scanned, the marked labels represent the items that must be put into the knapsack.

2. Implement Dynamic Programming algorithm for the longest common subsequence.

Problem Explanation:

A subsequence of a given sequence is just the given sequence with some elements left out. LCS Problem Statement: Given two sequences, find the length of longest subsequence present in both of them. A subsequence is a sequence that appears in the same relative order, but not necessarily contiguous. For example, “abc”, “abg”, “bdf”, “aeg”, ‘”acefg”, .. etc are subsequences of “abcdefg”.

Examples:
1) Consider the input strings “AGGTAB” and “GXTXAYB”. Last characters match for the strings. So length of LCS can be written as:

L(“AGGTAB”, “GXTXAYB”) = 1 + L(“AGGTA”, “GXTXAY”)

	A	G	G	T	A	B
G	-	-	4	-	-	-
X	-	-	-	-	-	-
T	-	-	-	3	-	-
X	-	-	-	-	-	-
A	-	-	-	-	2	-
Y	-	-	-	-	-	-
B	-	-	-	-	-	1

Algorithm

```
int lcs( char *X, char *Y, int m, int n )
{
if (m == 0 || n == 0)
return 0;
if (X[m-1] == Y[n-1])
return 1 + lcs(X, Y, m-1, n-1);
else
return max(lcs(X, Y, m, n-1), lcs(X, Y, m-1, n));
}
```

Week-5:

1. Implement Dynamic Programming algorithm for the Optimal Binary Search Tree Problem.

Problem Eplanation:

1.Calculate optimal binary search tree using dynamic programming n=4 (a1,a2,a3,a4)=(do, if ,int, while)

(p1,p2,p3,p4)=(3,3,1,1) and (q0,q1,q2,q3,q4)= (2,3,1,1,1,) where 0<=p<=4.

Sol:

0	1	2	3	4
W00=2	W11=3	W22=1	W33=1	W44=1

C00=0 R00=0	C11=0 R11=0	C22=0 R22=0	C33=0 R33=0	C44=0 R44=0
W01=8 C01=8 R01=1	W12=7 C12=7 R12=2	W23=3 C23=3 R23=3	W34=3 C34=3 R34=4	
W02=12 C02=19 R02=1	W13=9 C13=12 R13=2	W24=5 C24=8 R24=3 0r 4		
W03=14 C03=25 R03=2	W14=11 C14=19 R14=2			
W04=16 C04=32 R04=2				

W00=q[0]=2 w11=q[1]=3 w22=q[2]=1 w33=q[3]=1 w44=q[4]=1

C00=0 c11=0 c22=0 c33=0 c44=0

R00=0 r11=0 r22=0 r33=0 r44=0

$$W[i,j]=p[j]+q[j]+w[I,j-1]$$

$$C[i,j]= \min_{i<k<=j} \{ C[i,k-1]+C[k,j]\} +W[i,j]$$

$$w[0,1]=p[1]+q[1]+w[0,0]$$
$$=3+3+2$$
$$=8$$

C[0,1]= min { c[0,0]+c[1,1]} + w[0,1]

k=1

=min{0+0}+ 8
=0+8
=8
r[0,1]=1

w[1,2]=p[2]+q[2]+w[1,1]

=3+1+3
=7
C[1,2]= min { c[1,1]+c[2,2]} + w[1,2]
k=2

=0+0+7
=7
r[1,2]=2

w[2,3]=p[3]+q[3]+w[2 , 2]

=1 + 1+ 1
=3
C[2,3]= min { c [2 ,2]+c[3, 3]} + w[2, 3]
k=3

= { 0 + 0}+3
=3
r[2,3]= 3

w[3,4]=p[4]+q[4]+w[3 , 3]

=1 + 1+ 1
=3
C[3,4]= min { c [3 ,3]+c[4, 4]} + w[3 , 4]
k=4

= { 0 + 0}+3
=3
r[3,4]= 4

w[0,2]=p[2]+q[2]+w[0 , 1]

=3 + 1 + 8
=12
C[0,2]= min { c [0 ,0]+c[1,2]} + w[0 , 2]
k=1,2 { , C[0,1]+C[2, 2] }

{ 0 + 3}

C[0,2]= min +
{ 8 + 0 } 12

= 7 +12
=19

r[0 , 2]= 1

w[1,3]=p[3]+q[3]+w[1 , 2]

= 1 + 1 +7
=9
C[1,3]= min { c [1 ,1]+c[2 ,3]]} + w[1, 3]

k=2,3 { , C[1,2]+C[3, 3] }

=min { 0 + 3}
{7 + 0} + 9

= min{ 3 ,7} + 9
=3+9
=12
r[1,3]= 2

w[2,4]=p[4]+q[4]+w[2 , 3]

= 1 + 1 +3

=5

C[2,4]= min c [2 ,2]+c[3 ,4] + w[2, 4]
k=3,4 , C[2,3]+C[4, 4]

C[2,4]= min 0+ 3
5
3 + 0

=min{3 + 5}

=8

r[2,4]= 3 0r 4

w[0,3]=p[3]+q[3]+w[0 , 2]

= 1+1+12

=14

C[0,3]= min C [0,0]+c[1 ,3] + w[0, 3]
k=1,2 ,3 , C[0,1]+C[2, 3]
C[0,2]+C[3,3]

C[0,3]= 0 + 12
8+ 3 14
19+0

=min{11 +14}

= 25

=r[0,3]= 2

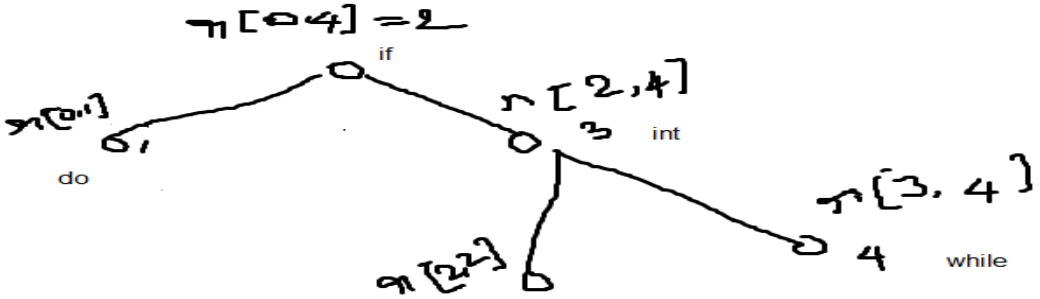
w[1,4]=p[4]+q[4]+w[1 , 3]

= 1 + 1 +9
=11

C[1,4]= min C [1,1]+c[2 ,4] + w[1, 4]
k=2,3,4 , C[1,2]+C[3, 4]
C[1,3]+C[4,4]

{ 0 + 8,
7+ 3,
12+ 0} +11

= 19
r[1,4]= 2
wher do , if , int and while considered as 1 , 2 , 3 and 4 respectively



Therefore the cost of OBST at root 2=32.
Algorithm OBST(p,q,n)
{
for i=0 to n-1 do
{
w[i,i]=q[i],r[I,i]=0 ,c[I,i]=0
w[i,i+1]=q[i]+q[i+1]+p[i+1]

```
r[i,i+1=i+1
c[i,i+1]=q[i]+q[i+1]+p[i+1]
}
   w[n,n]=q[n],r[n,n]=0,c[n,n]=0
 for m=2 to n do
    for i =0 to n-m do
{
     j=i+m

      w[i,j]=w[i,j-1]+p[j]+q[j]
k=Find(c,r,i,j);
c[i,j]=w[i,j]+c[i,k-1]+c[k,j]
r[i,j]=k
}
Write (c[0,n],w[0,n],r[0,n])
}
Algorithm Find(c ,r,i,j)
{
min=α
   for m=r[I,j-1] to r[i+1,j] do
     iIf (c[i,m-1]+c[m,j])<min  then
          {
          min=c[i,m-1]+c[m,j]
         l=m
        }
 return l
}
```

Week-6:

1. Implement Dynamic Programming algorithm for the TSP Problem.
   Problem Statement:
   In the TSP, given a set of cities and the distance between each pair of cities, a salesman needs to choose the shortest path to visit every city exactly once and return to the city from where he started.

Algorithm 1: Dynamic Approach for TSP

```
Data: s: starting point; N: a subset of input cities; dist():
      distance among the cities
Result: Cost : TSP result
Visited[N] = 0;
Cost = 0;
Procedure TSP(N, s)
    Visited[s] = 1;
    if |N| = 2 and k ≠ s then
        Cost(N, k) =dist(s, k);
        Return Cost;
    else
        for j ∈ N do
            for i ∈ N and visited[i] = 0 do
                if j ≠ i and j ≠ s then
                    Cost(N, j) = min ( TSP(N − {i}, j) + dist(j, i))
                    Visited[j] = 1;
                end
            end
        end
    end
    Return Cost;
end
```

Week7:

Implement Dynamic Programming for the matrix multiplication problem**.**

Problem Explanation:

Given a sequence of matrices, find the most efficient way to multiply these matrices together. The problem is not actually to perform the multiplications, but merely to decide in which order to perform the multiplications.
We have many options to multiply a chain of matrices because matrix multiplication is associative. In other words, no matter how we parenthesize the product, the result will be the same. For example, if we had four matrices A, B, C, and D, we would have:
(ABC)D = (AB)(CD) = A(BCD) = ...

However, the order in which we parenthesize the product affects the number of simple arithmetic operations needed to compute the product, or the efficiency. For example, suppose A is a 10 × 30 matrix, B is a 30 × 5 matrix, and C is a 5 × 60 matrix. Then,
(AB)C = (10×30×5) + (10×5×60) = 1500 + 3000 = 4500 operations
A(BC) = (30×5×60) + (10×30×60) = 9000 + 18000 = 27000 operations.
Clearly the first parenthesization requires less number of operations.
Given an array p[] which represents the chain of matrices such that the ith matrix Ai is of dimension p[i-1] x p[i]. We need to write a function MatrixChainOrder() that should return the minimum number of multiplications needed to multiply the chain.
Input: p[] = {40, 20, 30, 10, 30}
Output: 26000
There are 4 matrices of dimensions 40x20, 20x30, 30x10 and 10x30.
Let the input 4 matrices be A, B, C and D. The minimum number of multiplications are obtained by putting parenthesis in following way
(A(BC))D --> 20*30*10 + 40*20*10 + 40*10*30

A1	A2	A3	A4
3X2	2X4	4X2	2X5
d1 d2	d2 d3	d3 d4	d4 d5

Sol: take the matrix with 4x4 order and fill it values using the following

0	24	28	58
	0	16	36
		0	40
			0

$$C[i,j]=\min_{i\le k<j}\{ C[i,k]+C[k+1,j] + d_{i-1} \times d_k \times d_j\}$$

$$C[1,2]=\min_{k=1}\{ c[1,1]+C[2,2]+d0xd1xd2\}$$

$$=\min\{ 0 + 0 +3 \times 2 \times 4\}$$

$$=\min\{24\}=24$$

$$C[2,3]=\min_{k=2}\{ C[2,2] +C[3,3] + d1 \times d2 \times d3$$

= min {0 + 0 +2 x 4 x2}

=min{ 16}=16

C[3,4] =min { C[3,3] + C [4,4]+d2 x d3 x d4

k=3

= min{ 0 + 0 + 4 x 2 x 5}

=min{ 40}=40

C[1,3] =min C [1,1]+C[2,3]+d0 x d1 x d3}

k=1

k=2{ C[1,2] + C[3 ,3] + d0 x d2 xd3

= min{ 0+16 + 3 x 2 x 2 }

24 +0 + 3 x 4 x 2

= Min {28

48}=28

$$C[2,4] = \min_{k=2,3} \left\{ \begin{array}{l} C[2,2] + C[3,4] + d1 \times d2 \times d4 \\ C[2,3] + c[4,4] + d1 \times d3 \times d4 \end{array} \right.$$

$$\min \left\{ \begin{array}{l} 0 + 40 + 2 \times 4 \times 5 \\ 16 + 0 \quad 2x \quad 2 \times 5 \end{array} \right.$$

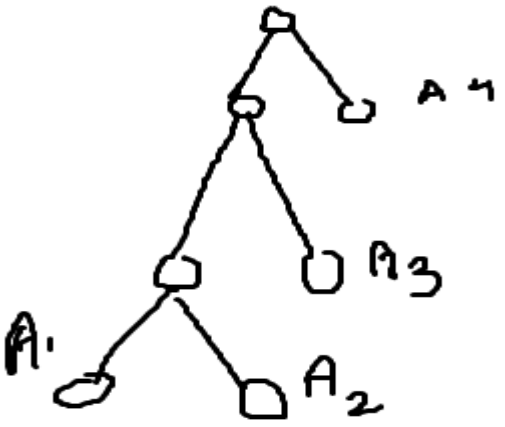
$$\min = \left\{ \begin{array}{l} 80 \\ 36 \end{array} \right\} = 36$$

$$C[1,4] = \min_{k=1,2,3} \left\{ \begin{array}{l} C[1,1] + C[2,4] + d0 \times d1 \times d4 \\ C[1,2] + C[3,4] + d0 \times d2 \times d4 \\ C[1,3] + C[4,4] + d0 \times d3 \times d4 \end{array} \right.$$

$$\min = \left\{ \begin{array}{l} 0 + 36 + 3 \times 2 \times 5 \\ 24 + 40 + 3 \times 4 \times 5 \\ 28 + 0 + 3 \times 2 \times 5 \end{array} \right\} = \left\{ \begin{array}{l} 66 \\ 124 \\ 58 \end{array} \right\} = 58$$

The optimal cost is 58.

BINARY TREE IS:



Week8:

1. Implement backtracking algorithm for the N-queens problem.

Problem Explanation:

N - Queens problem is to place n - queens in such a manner on an n x n chessboard that no queens attack each other by being in the same row, column or diagonal.

For 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 each other. In such a conditional each queen must be placed on a different row, i.e., we put queen "i" on row "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 a dead 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 another possible solution, the whole method is repeated for all partial solutions. The other solutions for 4 - queens problems is (3, 1, 4, 2) i.e.

		Q1	
Q2			
			Q3
	Q4		

Algorithm Place(k,i)

{

For j=1 to k-1 do

If((x[j]=i) or (Abs(x[j]-i)=Abs(j-k)))

Then return false

return true

}

Algorithm Nqueens(k,n)

{

for i= 1 to n do

```
  {
    if Place(k,i) then
     {   x[k]=i
        if (k=n) then write (x[1:n])
          ellse NQueens(k+1,n)
}
}
}
```

Week9:

Implement the backtracking algorithm for the sum of subsets problem

Problem Explanation:

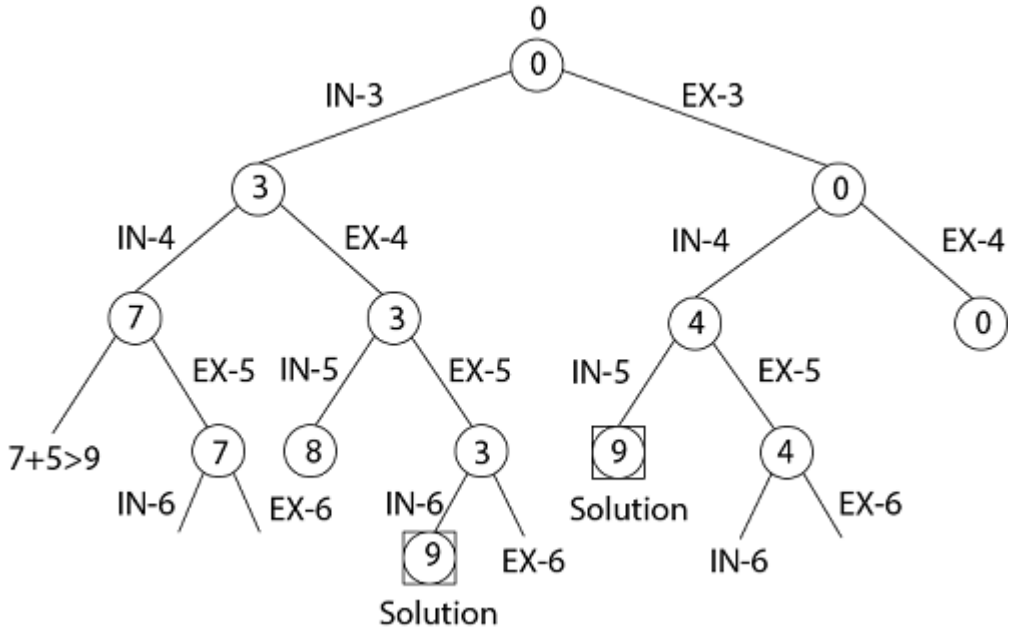
1. Start with an empty set
2. Add the next element from the list to the set
3. If the subset is having sum M, then stop with that subset as solution.
4. If the subset is not feasible or if we have reached the end of the set, then backtrack through the subset until we find the most suitable value.
5. If the subset is feasible (sum of seubset < M) then go to step 2.
6. If we have visited all the elements without finding a suitable subset and if no backtracking is possible then stop without solution.

Example: Given a set S = (3, 4, 5, 6) and M =9. Obtain the subset sum using Backtracking approach.

Solution:

Initially S = (3, 4, 5, 6) and X =9.

The implicit binary tree for the subset sum problem is shown as fig:




```
Algorithm   SumOfSub(s,k,r)
 {
    X[k]=1
     if (s+w[k]=m) then write(x[1:k])
     Else if (s+w[k]+w[k+1]<=m)
               Then SumOfSub(s+w[k],k+1,r-w[k])
    if ((s+r-w[k]>=m) and (s+w[k+1]<=m)) then
    {
        X[k]=0
       SumOfSub(s,k+1,r-w[k])
   }
}
```

Week10:

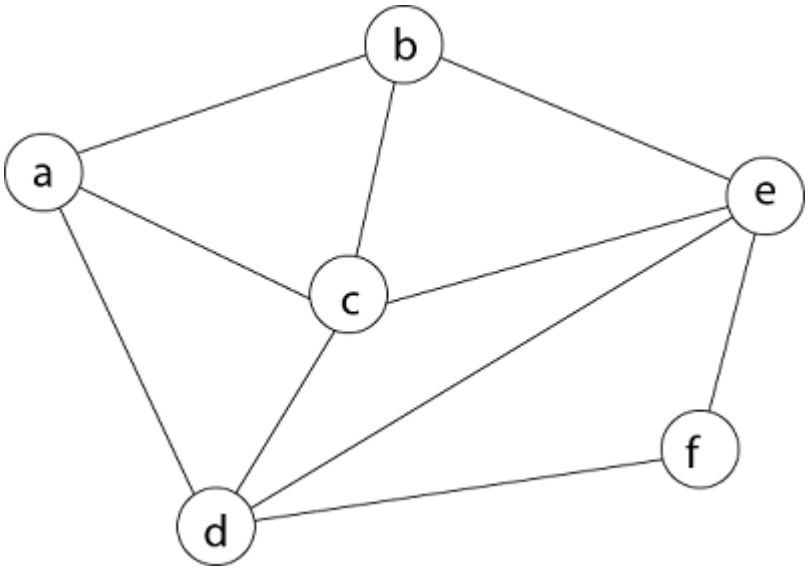
Implement the back tracking algorithm for Hamiltonian circuits problem

Problem Explanation:

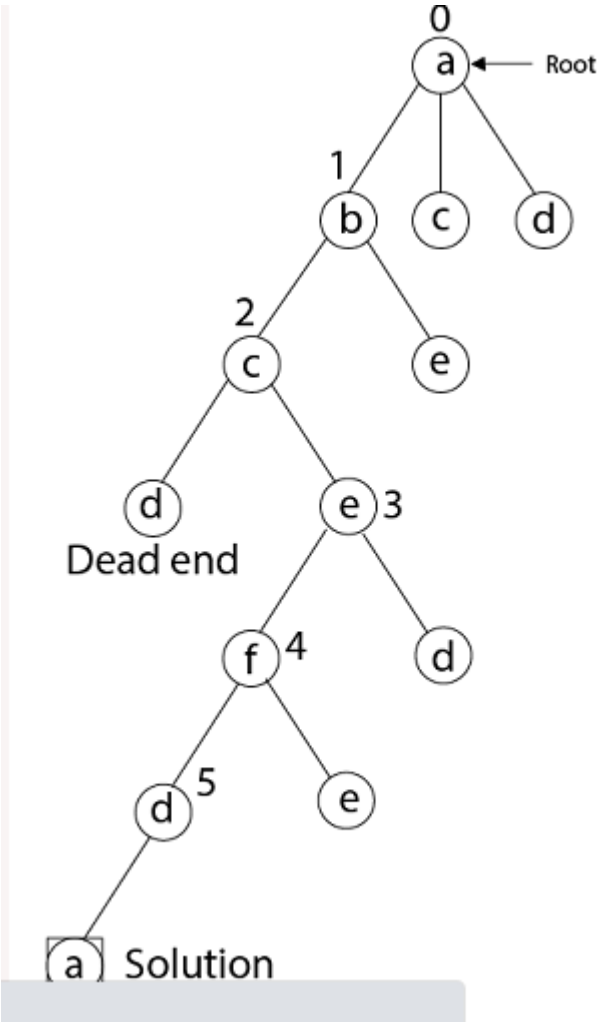
Given a graph G = (V, E) we have to find the Hamiltonian Circuit using Backtracking approach. We start our search from any arbitrary vertex say 'a.' This vertex 'a' becomes the root of our

implicit tree. The first element of our partial solution is the first intermediate vertex of the Hamiltonian Cycle that is to be constructed. The next adjacent vertex is selected by alphabetical order. If at any stage any arbitrary vertex makes a cycle with any vertex other than vertex 'a' then we say that **dead end** is reached. In this case, we backtrack one step, and again the search begins by selecting another vertex and backtrack the element from the partial; solution must be removed. The search using backtracking is successful if a Hamiltonian Cycle is obtained.

Example: Consider a graph G = (V, E) shown in fig. we have to find a Hamiltonian circuit using Backtracking method.



Final diagram:

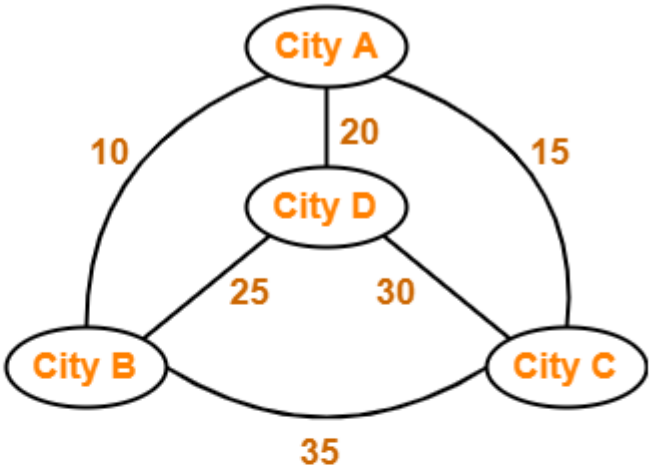


Week -11:

Implement LC branch and bound algorithm for the TSP Problem

Problem Explanation:

A salesman has to visit every city exactly once.
He has to come back to the city from where he starts his journey.
What is the shortest possible route that the salesman must follow to complete his tour?



Travelling Salesman Problem

If salesman starting city is A, then a TSP tour in the graph is-

A → B → D → C → A

Cost of the tour

= 10 + 25 + 30 + 15

= **80 units**

Week-12:

Implement LC branch and bound algorithm for the Knapsack problem

Problem Explanation:

Input: N = 4, C = 15, V[]= {10, 10, 12, 18}, W[]= {2, 4, 6, 9}
Output:
Items taken into the knapsack are
1 1 0 1
Maximum profit is 38
Explanation:
1 in the output indicates that the item is included in the knapsack while 0 indicates that the item is excluded.
Since the maximum possible cost allowed is 15, the ways to select items are:

(1 1 0 1) -> Cost = 2 + 4 + 9 = 15, Profit = 10 + 10 + 18 = 38.
(0 0 1 1) -> Cost = 6 + 9 = 15, Profit = 12 + 18 = 30
(1 1 1 0) -> Cost = 2 + 4 + 6 = 12, Profit = 32
Hence, maximum profit possible within a cost of 15 is 38.
Input: N = 4, C = 21, V[]= {18, 20, 14, 18}, W[]= {6, 3, 5, 9}
Output:
Items taken into the knapsack are
1 1 0 1
Maximum profit is 56
Explanation:
Cost = 6 + 3 + 9 = 18
Profit = 18 + 20 + 18 = 56