Dynamic Programming:

- Dynamic Programming is applied to optimization problem
- Dynamic Programming solves the problem by combining the solution of sub problem.
- It is applicable when sub problems are dependent i.e., sub problems share the information.
- Dynamic programming solves every sub problem just once & save the outcomes in table.
- Dynamic programming algorithm can be broken into 4 steps:
  (a) Characterise the structure of optimal solution.
  (b) recursively define the value of an optimal soln
  (c) compute the value of optimal soln in a bottom-up fashion.
  (d) construct the optimal solution from computed information.

Matrix chain Multiplication:-

Consider 3 matrices of the order $A_{10\times100}$, $B_{100\times5}$, $C_{5\times50}$

$(AB)C = 10\times100\times5 + 10\times5\times50 = 7500$

$A(BC) = 100\times5\times50 + 10\times50\times100 = 75000$

We have seen that by changing the evaluation sequence of matrices, the number of multiplication also changes.

The number of multiplication should be minimum.

The matrix chain multiplication problem can be written as follows:-

"Given a chain of matrices $\langle A_1, A_2, A_3, \ldots, A_n \rangle$ of n matrices.
Matrix A_i have dimension $p_{i-1} \times p_i$.
In this problem, to find fully parenthesis the product of $A_1, A_2 \ldots A_n$ matrices in a way that minimize the number of scalar multiplication."

Step 1:- The structure of an optimal solution $A_i, A_{i+1}, \ldots, A_j$ dividing into 2 parts:-

$A_i, A_{i+1}, A_{i+2}, \ldots, A_k$ and $A_{k+1}, A_{k+2}, \ldots, A_j$ where $i \le k < j$

We will keep on dividing this until we get the single element:-
At each step the optimal value is calculated.

Step 2: Recursive solution:-

Let $m[i,j]$ denote the minimum number of scalar multiplications.

$$m[i,j] = \begin{cases} 0 & ; \text{if } i=j \\ \min_{i \le k < j} \{ m[i,k] + m[k+1,j] + p_{i-1} * p_k * p_j \} \end{cases}$$

Let $S[i,j]$ to store a value of k at which we can split the product of $A_i, A_{i+1}, \ldots A_j$ to obtain a optimal solution.

Step 3:- Compute the optimal value:-

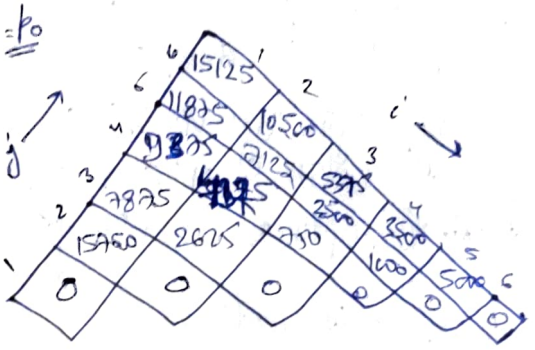
$p_0\ p_1\ p_2\ p_3\ p_4\ p_5\ p_6$
$\langle 30, 35, 15, 5, 10, 20, 25 \rangle$

Consider the following example:-

$n=6$;

$A_1 \rightarrow 30 \times 35$ ($p_0\ p_1$)
$A_2 \rightarrow 35 \times 15$ ($p_1\ p_2$)
$A_3 \rightarrow 15 \times 5$ ($p_2\ p_3$)
$A_4 \rightarrow 5 \times 10$ ($p_3\ p_4$)
$A_5 \rightarrow 10 \times 20$ ($p_4\ p_5$)
$A_6 \rightarrow 20 \times 25$ ($p_5\ p_6$)

$p_{i-1} = p_{i-1} = p_0$



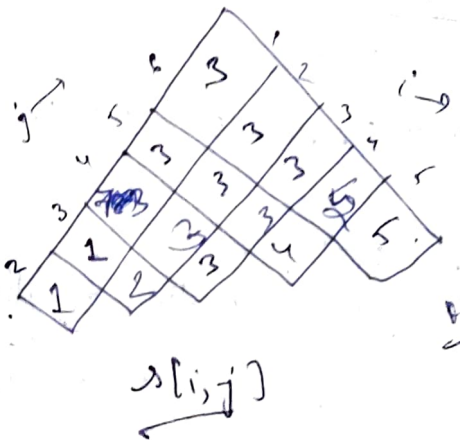
$m[i,j]$ $\quad i \le j < k$

$m[1,2]$: $i=1, j=2, k=1$

$= \min\{ m[1,1] + m[2,2] + p_0 * p_1 * p_2 \}$

$= \min\{0 + 0 + 30 \times 35 \times 15\}$

$= 15750$



$s[i,j]$

$k=2$ $m[2,3]$ $\quad i=2, j=3, k=2$

$m[2,3] = 2625$

$m[3,4] = 750$

$m[4,5] = 1000$

$m[5,6] = 5000$

$m[1,3]$ $\quad i=1, j=3, k=1,2$

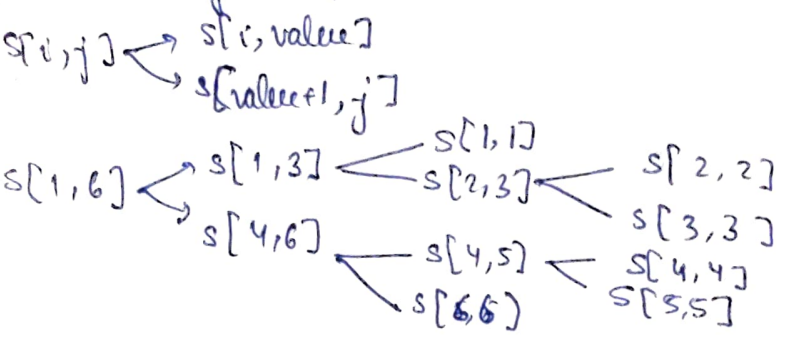
$k=1$:

$m[1,3] = \min\begin{cases} k=1 & m[1,1] + m[2,3] + p_0 \times p_1 \times p_3 \\ & 7875 \\ k=2 & m[1,2] + m[3,3] + p_0 * p_2 * p_3 \\ & 18000 \end{cases}$

25/09/16

The top most value will be the optimal solution

min^m no. of multiplication = 15125.



$((A_1(A_2A_3))((A_4A_5)A_6))$

Step 3: Algorithm

```
Matrix_Chain(p)
1.  n ← length[p] - 1
2.  for i ← 1 to n
3.  do m[i,i] ← 0
4.  for l ← 2 to n                    2 to 6
5.  do for i ← 1 to n-l+1             1 to 5
6.       do
7.            j ← i+l-1
8.            m[i,j] ← ∞
9.       for k ← i to j-1
10.       do calculate q ← m[i,k] + m[k+1,j] + p_{i-1} * p_k * p_j
11.       if (q < m[i,j])
12.       then q ← m[i,j]
13.            s[i,j] ← k
          return m,s
```

Time complexity = $\Theta(n^3)$.

Step 4:- Constructing an optimal solution.

```
optimal_solution(s,i,j)
1.  if (i=j)
2.  then print "Ai"
3.  else print "("
4.  optimal_solution(s,i, s[i,j])
5.  optimal_solution(s,s[i,j]+1, j)
6.  print ")".
```

10M

Longest Common Subsequence (LCS):-

In this problem, we are given 2 sequences, we have to find longest common subsequences of X & Y.

$X = \langle x_1, x_2, \cdots, x_m \rangle$

$Y = \langle y_1, y_2, \cdots, y_n \rangle$

Step 1: Optimal Structure of LCS:-

Let $X = \langle x_1, x_2, \cdots, x_m \rangle$ & $Y = \langle y_1, y_2, \cdots, y_n \rangle$ be two subsequence & Let $Z = \langle z_1, z_2, \cdots, z_k \rangle$ be any LCS of X & Y.

(i) If $x_m = y_n$, then $z_k = x_m = y_n$ and z_{k-1} is an LCS of X_{m-1} & Y_{n-1}.

(ii) If $x_m \neq y_n$, then $z_k \neq x_m$ this implies that Z is an LCS of X_{m-1} & Y_n.

(iii) If $x_m \neq y_n$, then $z_k \neq y_n$ then implies that Z is an LCS of X_m & Y_{n-1}.

Step 2: Recursive formula:-

Let $c[i,j]$ defined recursively,

$c[i,j] = 0$ if $i=0$ or $j=0$

$c[i,j] = c[i-1, j-1] + 1$ if $x_i = y_j$

$c[i,j] = \max\{ c[i-1,j], c[i,j-1] \}$ if $x_i \neq y_j$

Step 3:- Consider the following 2 sequences:-

$X = \{ A, B, C, B, D, A, B \}$

$Y = \{ B, D, C, A, B, A \}$

		A	B
	0	0	0
B	0	0	1
B	0	0	1

Make table of length $(X+1) * (Y+1)$

		B	D	C	A	B	A
	0	0	0	0	0	0	0
A	1	0	0	0	1	1	1
B	2	0	1	1	1	2	2
C	3	0	1	1	2	2	2
B	4	0	1	1	2	3	3
D	5	0	1	2	2	3	3
A	6	0	1	2	2	3	4
B	7	0	1	2	2	3	4

Longest common subsequence = B C A

Length of LCS = 4

B A

30/09/16

Step 3:- Computing the length of an LCS

at least show 2 row reduction.

```
LCS_length(X,Y)
1. m ← length[X]
2. n ← length[Y]
3. for i ← 0 to m
4. do c[i,0] ← 0          c[i][0]
5.    for j ← 0 to n
6.    do c[0,j] ← 0
7.    for i ← 1 to m
8.    do for j ← 1 to n
9.       do if (x_i = y_j)
             then c[i,j] ← c[i-1, j-1] + 1
                  b[i,j] ← "↖"
          else if c[i-1, j] ≥ c[i, j-1]
             then c[i,j] ← c[i-1, j]
                  b[i,j] ← "↑"
          else  c[i,j] ← c[i, j-1]
                b[i,j] ← "←"
   return b and c
```

the running time of this procedure is $O(mn)$.

Step 4:- Optimal_solution :-

```
LCS_Optimal_Solution(b, X, n, m)
1. if (n=0 or m=0)
2. then return 0
3.    else if (b[n,m] = "↖")
4.       then LCS_Optimal_Solution (b, X, n-1, m-1)
5.            print "x_n"
6.       else if (b[n,m] = "↑")
7.          then LCS_Optimal_Solution (b, X, n-1, m)
8.          else LCS_Optimal_Solution (b, X, n, m-1)
```

Time complexity of this is $O(m+n)$.

01/10/16

Strassen's Matrix Multiplication :-

Strassen's

If 2 matrices given of nxn order & $n = 2^k$; $k \geq 1$.

Consider the matrix $\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$

$C_{11} = A_{11} * B_{11} + A_{12} * B_{21}$
$C_{12} = A_{11} * B_{12} + A_{12} * B_{22}$
$C_{21} = A_{21} * B_{11} + A_{22} * B_{21}$
$C_{22} = A_{21} * B_{12} + A_{22} * B_{22}$

Time complexity is given as :-

$T(n) = 8\,T(\frac{n}{2}) + \theta(n^2)$

by master method,

$a = 8,\ b = 2,\ f(n) = n^2$

Calculate $n^{\log_b a} = n^{\log_2 8} = n^3$

case I :-

$f(n) = O(n^{(\log_b a - \epsilon)})$

$n^2 = O(n^{3-\epsilon})$

$\epsilon = 1$

case 1 holds,

$T(n) = \theta(n^{(\log_b a)}) = \underline{\theta(n^3)}$

In SMM, given some recursive formula, in this 2 matrices given of nxn & $n = 2^k$.

The formulae are :-

Consider: $\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$

$P = (A_{11} + A_{22})(B_{11} + B_{22})$
$Q = (A_{21} + A_{22}) B_{11}$
$R = (A_{11} (B_{12} - B_{22})$
$S = A_{22} (B_{21} - B_{11})$
$T = (A_{11} + A_{12}) B_{22}$
$U = (A_{21} - A_{11}) (B_{11} + B_{12})$
$V = (A_{12} - A_{22}) (B_{21} + B_{22})$

$C_{11} = P + S - T + V$
$C_{12} = R + T$
$C_{21} = Q + S$
$C_{22} = P + R - Q + U$

Use Strassen's MM to compute the matrix product of

$$\begin{pmatrix} 1 & 3 \\ 7 & 5 \end{pmatrix}\begin{pmatrix} 6 & 8 \\ 4 & 2 \end{pmatrix}$$

P = 48 ✓
Q = 72 ✓
R = 6 ✓
S = -10 ✓
T = 6
U = 64
V = -12

$C_{11} = 18$
$C_{12} = 14$
$C_{21} = 62$
$C_{22} = 66$

$$C\begin{bmatrix} 18 & 14 \\ 62 & 66 \end{bmatrix}$$

Time Complexity :-

Algorithm:

1. If n=1
2. then return A×B
3. else P ← strassen_multiplication $(A_{11}+A_{22}, B_{11}+B_{22}, \frac{n}{2})$
4. Q ← strassen_multiplication $(A_{21}+A_{22}, B_{11}, n/2)$
5. R ← " $(A_{11}, B_{12}-B_{22}, n/2)$
6. S ← " $(A_{22}, B_{21}-B_{11}, n/2)$
7. T ← " $(A_{11}+A_{12}, B_{22}, n/2)$
8. U ← " $(A_{21}-A_{11}, B_{11}+B_{12}, n/2)$
9. V ← " $(A_{12}-A_{22}, B_{21}+B_{22}, n/2)$
10. C_{11} ← P+S-T+V
11. C_{12} ← R+T
12. C_{21} ← Q+S
13. C_{22} ← P+R-Q+V

Data structure for disjoint sets :-

A disjoint set DS maintains a collection of disjoint dynamic sets.
Let $S = \{S_1, S_2, \ldots, S_k\}$.
Each set is identified by a representative which is the member of that set.
If we have two sets S_x & S_y.

Representative: $S_x = \{1, 2, 3, 4, 5\}$
$S_y = \{6, 7\}$

Then this sets are called disjoint sets, because there is no common element in both sets.

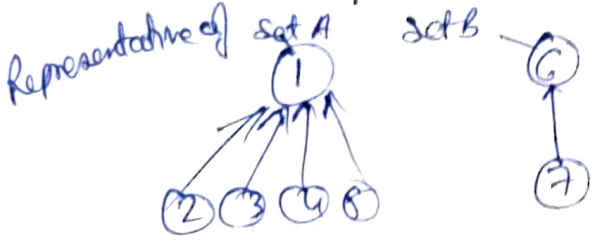
The first element is representative of set.

Tree representation of disjoint sets:-

Suppose sets A and B are disjoint set

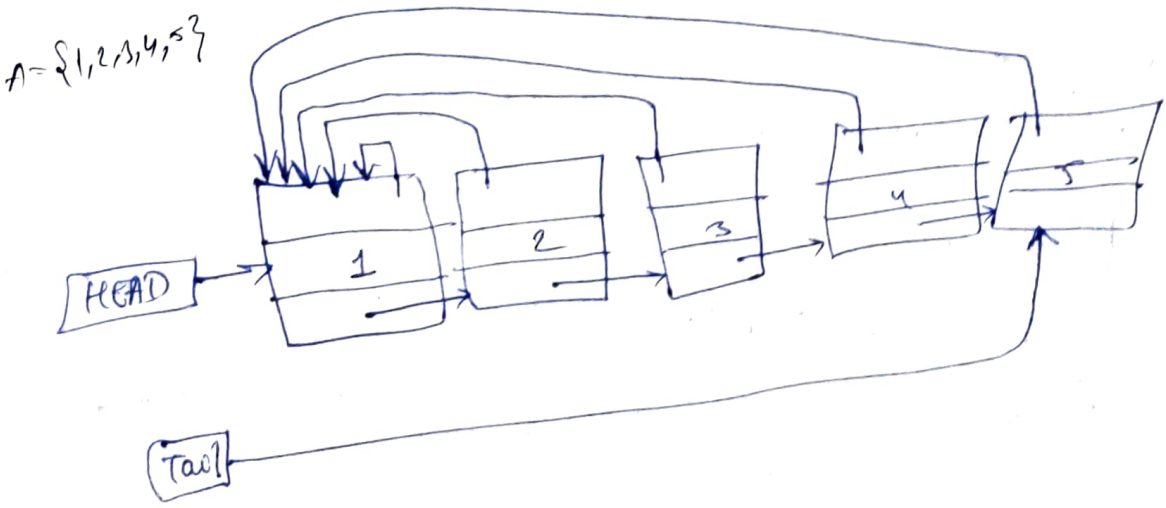
A = { 1, 2, 3, 4, 5 }

B = { 6, 7 }



Linked list representation of Disjoint Set :-

In this, first element is serve as representative.
Each element in the linked list contain a set member, a pointer to the next element set member.
A pointer back to the representative member.



Operations for disjoint Sets:-

1. Make-Set(S); It creates a new set, that member is pointed by x.

```
Make_SET(x)
1. p[x] ← x
2. rank[x] ← 0
```

2. Union(x,y): It contains all the elements of the sets Sx & Sy. The representatives is any member of x.

```
• LINK(FIND_SET(x) - FIND_SET(y))
• LINK(x,y)
1. if rank[x] > rank[y]
2. then p[y] ← x
3. else p[x] ← y
4. if rank[x] = rank[y]
   rank[y] + 1
```

3. FIND_SET(x) :- It returns pointer to the representative of set.

```
FIND_SET(x)
1. if x ≠ p[x]
2. then p[x] ← FIND_SET(p[x])
3. return p[x]
```

03/10/16

Spanning Tree :-

Spanning tree of graph is just a sub graph that contains all the vertices and is a tree.

minimum spanning tree (MST) :-

A min^m spanning tree whose weight is less than or equal to all other spanning tree.

Application :-

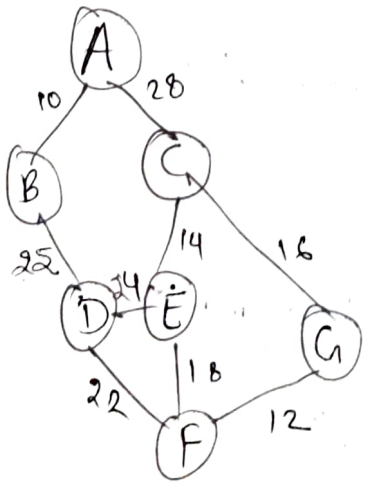
1. For finding the airline routes
2. In the design of network.

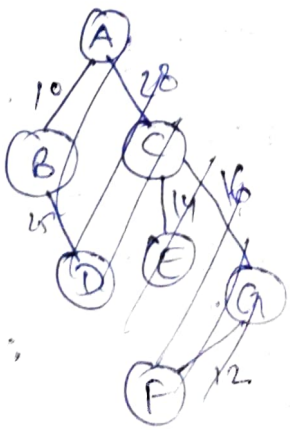
e.g.; **Kruskal's Algorithm :-**

It is an algorithm that finds min^m spanning tree for a connected weighted graph.

The time complexity of this algorithm is $O(E \lg E)$ or $O(E \lg V)$

Q. Find the MST for the graph :-

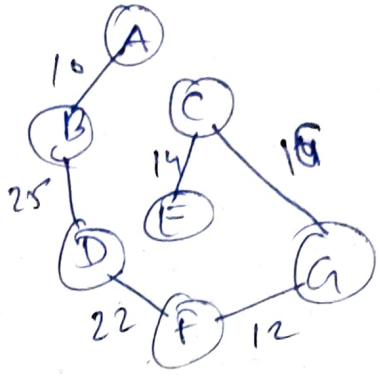




Solution:- Arrange all the edges in increasing order of weight:-

AB = 10 ✓
FG = 12 ✓
CE = 14 ✓
CG = 16 ✓
BF = 18 ✗
DF = 22 ✓
DE = 24 ✗
BD = 25 ✓
AC = 28 ✗

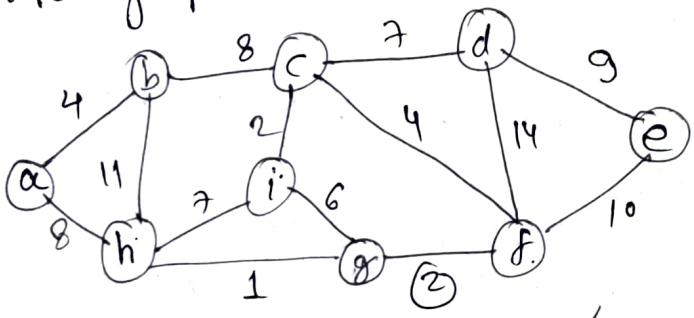
avoid cycling



cost = 99

NOTE:- for n vertices graph, in MST, (n-1) edges must be present.

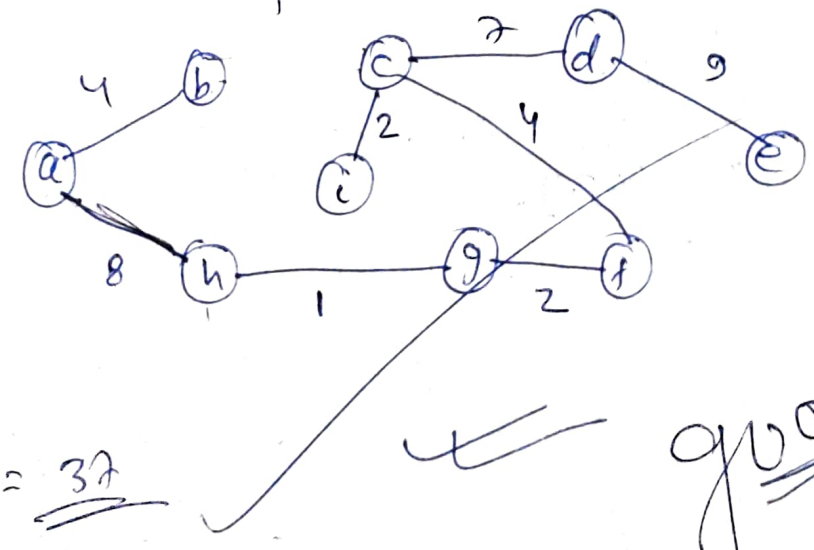
8. find the MST for the graph:-



arrange all the edges in increasing order of weight:-

gh = 1, ci = 2, gf = 2
ab = 4, cf = 4
✗ ig = 6, ✗ hi = 7, ad = 7, ah = 8, bc = 8, de = 9

fe = 10 ✓
bh = 11 ✗
df = 14



cost = 37

good.

Algorithm:

// G is a graph, w is an array of weight

```
MST_Kruskal(G,w).
1.  A ← φ
2.  for each vertex v ← V[G]
3.  do MAKE_SET(v)
4.  Sort the edges of E into increasing order of weight w
5.  for each edge (u,v) ∈ E taken in non-decreasing order of weight
6.  do if FIND_SET(u) ≠ FIND_SET(v)
7.     then A ← A ∪ {u,v}
8.          UNION(u,v).
9.          return A.
```

e.g. Prim's Algorithm:

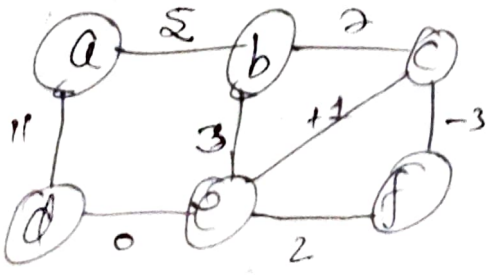
Like Kruskal Algorithm, prims algorithm is special case of generic MST.

In this, we initialize priority queue to contain all the vertices & the key of each vertex set to ∞ except the root vertex, whose key value is 0.

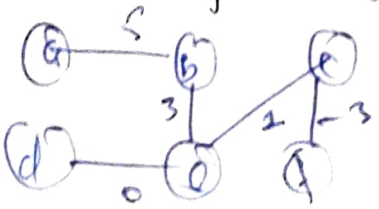
O(E + V log V)

```
MST_Prims(G, w, r) // r is root of spanning tree
1.  for each v ← V(G)
2.  do key[v] ← ∞
3.     π[v] ← NIL
4.     key[r] ← 0
5.     Q ← V[G]
6.  while Q ≠ φ
7.  do u ← EXTRACT_MIN(Q)
8.     for each v ← Adj[u].
9.     do if (v ∈ Q and w(u,v) < key[v])
10.       then π[v] ← u
11.            key[v] ← w(u,v)
```

Q1



Find the MST using Prims algorithm.



root vertex is a :-

Q | a | b | c | d | e | f |

0 ∞ ∞ ∞ ∞ ∞

while (Q ≠ ϕ) true
a ← extract_min(Q)
Adj{a} = {b, d}

| b | c | d | e | f |

5 ∞ 11 ∞ ∞

while Q ≠ ϕ true
b ← extract_min(Q)
Adj{b} = { c, e }

| c | d | e | f |

7 11 3 ∞

while Q ≠ ϕ true
e ← extract_min Q.
adj(e) = {d, c, f}

| c | d | f |

1 0 2

while Q ≠ ϕ true
d ← extract_min(Q)
adj{d} = { ϕ }

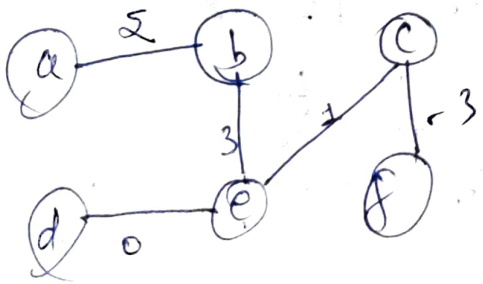
| c | f |

1 2

while Q ≠ ϕ true
c ← extract_min(Q)
adj(c) ← { f }

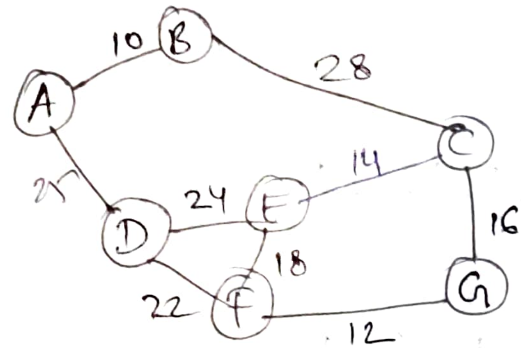
| f |

-3



cost = 6

Q. find MST using prims algorithm.



Q

A	B	C	D	E	F	G
0	∞	∞	∞	∞	∞	∞

while $(Q \neq \phi)$ true.
A ← extract min (Q)
adj (A) = { B, D }

B	C	D	E	F	G
10	∞	25	∞	∞	∞

while $(Q \neq \phi)$ true.
B ← extract_min (Q)
adj (B) = { C }

C	D	E	F	G
28	25	∞	∞	∞

while $(Q \neq \phi)$ true
D ← extract_min (Q)
adj (D) = { E, F }

C	E	F	G
28	24	22	∞

while $(Q \neq \phi)$ true
F ← extract_min (Q)
adj { F } = { E, G }

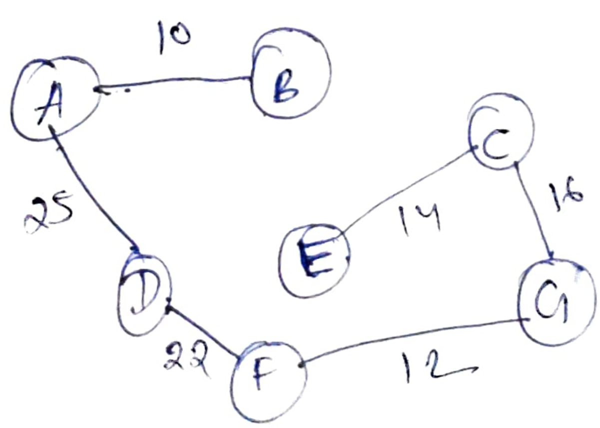
C	E	G
28	18	12

while $(Q \neq \phi)$
G ← Extract_min (Q)
adj (G) = { C }

C	E
16	18

while $(Q \neq \phi)$ C ← extract min (Q)
adj (C) = { E }

E
14



cost = 99

06/10/16

Dijkastra Algorithm :-

It solves the single source shortest path problem on a weighted graph in which all the edges have non-ive weights.

Given a graph, $G = \{V, E\}$, find the shortest path from a given source vertex to every other vertices.

DIJKSTR (G, w, s) — w: weight; s: source vertex of graph G

1. Initialize_single_source (G, s)
2. $S \leftarrow \phi$
3. $Q \leftarrow V[G]$
4. while $Q \neq \phi$
   - $u \leftarrow$ Extract_min (Q)
   - $S \leftarrow S \cup \{u\}$
   - for each vertex $v \leftarrow adj[u]$
   - do relax (u, v, w)

Initialise_single_source (G, s)

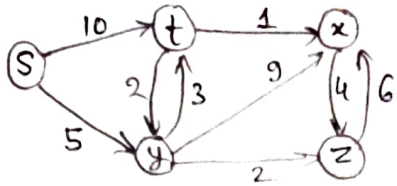
1. for each vertex $v \leftarrow V[G]$
2. do $d[v] \leftarrow \infty$
3. $\pi[v] \leftarrow NIL$
4. $d[s] \leftarrow 0$

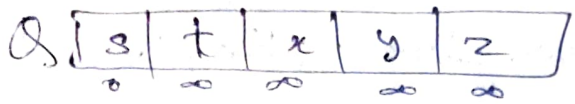
Relax (u, v, w)

1. if $d[v] > d[u] + w(u,v)$
2. then $d[v] \leftarrow d[u] + w(u,v)$
3. $\pi[v] \leftarrow u$.

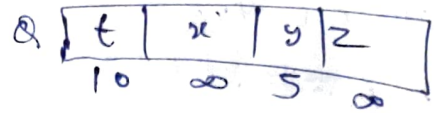
The running time of dijkstra's algorithm is lower than that Bellman for $O((V+E) \log V)$.

Q. Find the shortest path from the vertex s through other vertices using Dijkastra method.

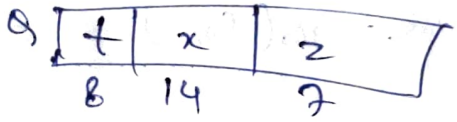




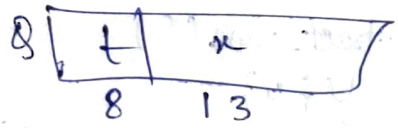
while Q≠0 is true.
s ← extract_min(Q).
adj(s) = {t, y}



while Q≠0 is true
y ← extract_min(Q)
adj(y) ← {t, x, z}



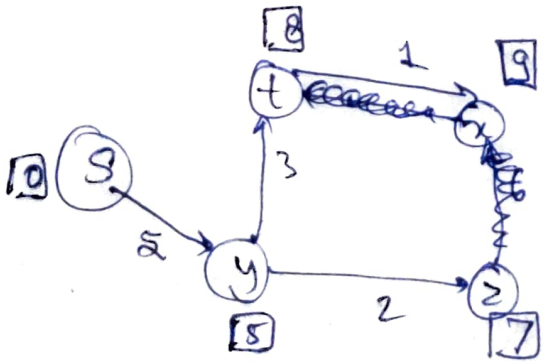
while Q≠0 is true
z ← extract_min(Q)
adj(z) = {x}



while Q≠0 is true
t ← extract_min(Q)
adj(t) = {x}

while Q≠0 is true
x ← extract_min(Q)

while Q≠0 is false.



Q Find LCS:-

$X = \langle 1,0,0,1,0,1,0,1 \rangle$

$Y = \langle 0,1,0,1,1,0,1,1,0 \rangle$

X \ Y	0	0	1	0	1	1	0	1	1	0
0	0	0	0	0	0	0	0	0	0	0
1	0	0↑	1↖	1←	1↖	1↖	1←	1↖	1↖	1←
0	0	1↖	1↑	2↖	2←	2←	2↖	2←	2←	2↖
0	0	1↖	1↑	2↖	2↑	2↑	3↖	3←	3←	3↖
1	0	1↑	2↖	2↑	3↖	3↖	3↑	4↖	4↖	4←
0	0	1↖	2↑	3↖	3↑	3↑	4↖	4↑	4↑	5↖
1	0	1↑	2↖	3↑	4↖	4↖	4↑	5↖	5↖	5↑
0	0	1↖	2↑	3↖	4↑	4↑	5↖	5↑	5↑	6↖
1	0	1↑	2↖	3↑	4↖	5↖	5↑	6↖	6↖	6↑

Length = 6

LCS = 1 0 0 1 1 0

07/10/16

Greedy Algorithm:-

A greedy algorithm always makes the choice that looks best at that moment i.e it makes locally optimal choice in the hope that it may lead to a globally optimal solution.

Optimization problems are solved by greedy algorithm. They do not give always the optimal solution but for many problem, they find the optimal solution.

NOTE:- In dynamic programming, we use bottom-up approach while in Greedy algorithm, we use top-down approach.

Huffman Algorithm / Huffman Coding:-

Huffman invented a Greedy algorithm that constructs an optimal prefix code called Huffman code. It is used for encoding & decoding in the n/w security.

A code is said to be prefix code if no code word is prefix of some other code word.

eg: a — 11
b — 10
c — 0
} not a prefix code

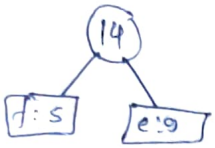
Q Consider the following alphabets {a, b, c, d, e, f} & the frequency of letters:-

f(a) = 42 | f(c) = 12 | f(e) = 9
f(b) = 13 | f(a) = 16 | f(d) = 5

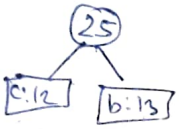
Find the huffman code for this data.

Solution:- Sort the letters increasing order of frequency:-

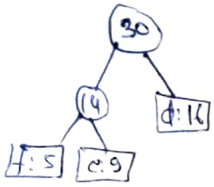
f:5 e:9 c:12 b:13 d:16 a:45



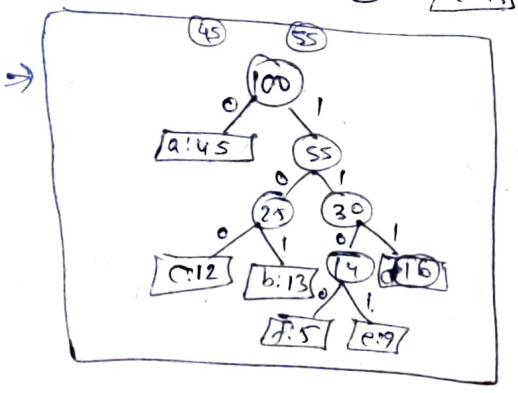
c:12 b:13 14 d:16 a:45

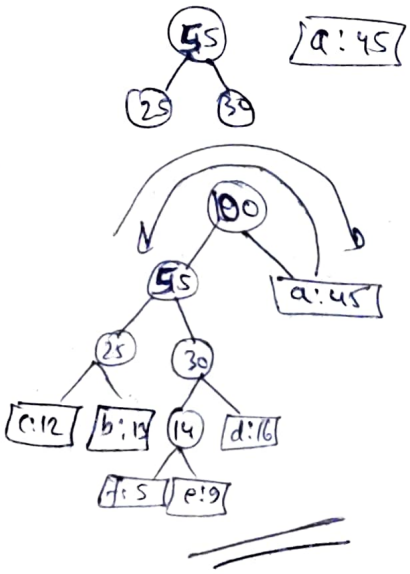


14 d:16 25 a:45



25 30 a:45





prefix code:
a = 0
b = 101
c = 100
d = 111
e = 1101
f = 1100

Let Q is min-heap of the values of the frequency array(c):-

Path length
= 1×45
+ 3×13
+ 3×12
+ 3×16
+ 3×14

Algorithm:-

Huffman(c)

1. n ← length[c]
2. Q ← c
3. for i ← 1 to n-1
4. do allocate a new node
5. x ← Extract_min(Q)
6. y ← Extract_min(Q)
7. left[z] ← x
8. right[z] ← y
9. f(z) ← f(x) + f(y)
10. insert(z, Q)

Time complexity = $\Theta(n \log n)$

14/10/16

Knapsack Problem :-

In this problem, a thief enters into a shop and find n items, the ith item has value p_i and weight w_i s (where w_i & p_i are integers).

Thief wants to take valuable loads but he can carry at most W weights. ($w_i \leq W$) W capacity of the knapsack

There are 2 types of knapsack problem :-

1. **Fractional knapsack problem :-**

In this knapsack problem, each item either must be taken or left and also the fraction of item can be taken.

2. **0/1 knapsack problem :-**

In this knapsack problem, item can be taken or left.

NOTE :- fractional knapsack problem is solved by greedy.
& 0/1 knapsack problem is solved by dynamic.

Eg: Consider the following instance of knapsack with 5 objects and capacity of knapsack W = 100. How to fill the object in the knapsack that maximum profit can be obtained.

w :	10	20	30	40	50
p :	20	30	66	40	60
profit density p/w :	2	1.5	2.2	1	1.2

soln :- Greedy to weight :-
from the remaining object, choose the object which has min weight i.e., objects are taken in order of increasing weights.

Total profit = $20 \times 1 + 30 \times 1 + 66 \times 1 + 40 \times 1$
= 156

solution vector = (1, 1, 1, 1, 0).

Greedy to profit :- from the remaining objects, choose the object which has maximum profit.

total profit = $66 \times 1 + 60 + 40 \times \frac{1}{2}$ = 146

solution vector = $(0, 0, 1, \frac{1}{2}, 1)$

iii) greedy to profit density :-

from the remaining object, choose the object which has max profit density.

$60 \times \frac{4}{5} = 48$ (12)

$\frac{40}{50}$ weight remain / total weight

total profit = $66 + 20 + 30 + 6 \times \frac{4}{5}$

$= 116 + 48$

$= 164$

solution vector = $(1, 1, 1, 0, \frac{4}{5})$

Maximum profit is gained by greedy to profit density.

Limitation of greedy algorithm :-

Greedy algorithm can be applied only to the fractional knapsack problem and it can't be applied to 0/1 knapsack problem.

eg:

$n = 3$, $W = 10$

w :	6	5	5
P :	8	5	5
profit density P/w :	1.33	1	1

total profit = $8 \times 1 + \frac{4}{5} \times 5$

$= 8 + 4$

$= 12$.

Hence, 0/1 knapsack problem can't be solved by greedy method.

Algorithm:-

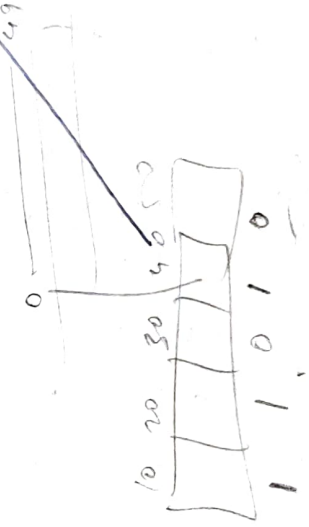
GreedyKnapsack(m,n)

// p[1..n] and w[1..n] contains profit & weight array respectively.

// n objects ordered such that $\frac{p[i]}{w[i]} \geq \frac{p[i+1]}{w[i+1]}$

// m is knapsack size and x[1..n] is solution vector

```
1.  for i ← 1 to n
2.  do x[i] ← 0.0
3.     U ← m
4.  for i ← 1 to n
5.  do if w[i] > W
       then break;
6.     x[i] ← 1.0
7.
8.     U ← U - w[i]
9.  if (i ≤ n)
10. then x[i] ← U / w[i]
```



15/10/16

fractional_knapsack(w, p, W)

```
1.  for i ← 1 to n
2.  do x[i] ← 0
3.  weight ← 0
4.  while weight < W and i < n
5.  do   i ← best remaining object according to greedy criteria
6.       if (weight + w[i] < W)
7.       then { x[i] ← 1.0
8.              weight ← weight + w[i]
               }
9.       else
10.        { x[i] ← (W - weight) / w[i]
11.        weight ← w[i] * x[i]
           weight +
           }
12.  return x
```

Analysis: If the items are already sorted in the decreasing order of $\frac{p_i}{w_i}$, then total time complexity is $O(n)$

0/1 knapsack problem with dynamic programming (DAA) :-

If n number of items is given and W is the capacity of knapsack.
Let c[i, W] denote the value of solution.
c[i, W] defined recursively as :-

$$c[i,W] = \begin{cases} 0 & \text{if } i=0 \text{ or } W=0 \\ c[i-1,W] & \text{if } w_i > W \\ \max\{p_i + c[i-1, W-w_i], c[i-1,W]\} & \text{if } w_i \le W \end{cases}$$

eg: Consider the following 0/1 knapsack problem in which no. of items n=5, whose profit is :-

$$p = \langle 20, 50, 30, 100, 60 \rangle$$

and weights $w = \langle 5, 40, 30, 20, 15 \rangle$ & $W = 50$.
Find the value of optimal soln (profit) & optimal soln (solution vector).

w_i :	5	40	30	20	15
p_i :	20	50	30	100	60
$\frac{p_i}{w_i}$:	4	1.25	1	5	4

$c[i,W] = c[5, 50]$

$= \max\{ 60 + \underbrace{c[4,35]}_{A\ 120}, \underbrace{c[4,50]}_{B\ 130} \} = 180$

A: $c[4,35] = \max\{ 100 + \underbrace{c[3,15]}_{a\ 20}, \underbrace{c[3,35]}_{b\ 50} \}$

$= 120$

a: $c[3,15] = c[2,15]$ ($w_3 > W$, i.e. 30 > 15)

$= c[1,15]$ ($w_2 > W$, i.e. 40 > 15)

$= \max\{ 20 + c[0,10], c[0,15] \}$ ($c[0,10] \to 0$, $c[0,15] \to 0$)

$= 20$

value of optimal soln = 180

optimal soln = $\langle 0, 0, 1, 1, 0 \rangle$ or $\langle 1, 1, 0, 1, 1 \rangle$

b: $C[3,35] = \max\{ 30 + \underbrace{C[2,5]}_{a_1}, \underbrace{C[2,35]}_{b_1} \}$ $\quad 30 \leq 35$

$= 50$

a_1: $C[2,5] = C[1,5]$

$= \max\{ 20 + C[0,0], C[0,5] \}$

$= 20$

b_1: $C[2,35] = C[1,35]$ $\quad w_i > W$, $40 > 35$

$= \max\{ 20 + C[0,30], C[0,35] \}$

$= 20$

B: $C[4,50] = \max\{ 100 + \underbrace{C[3,30]}_{a_2}, \underbrace{C[3,50]}_{b_2} \} = 130$ $\quad w_i \; W$, $20 \leq 50$

a_2: $C[3,30] = \max\{ 30 + C[2,0], \underbrace{C[2,30]}_{b_3} \} = 30$ $\quad (20)$

b_3: $C[2,30] = C[1,30]$ $\quad w_i > W$, $40 > 30$

$= \max\{ 20 + C[0,25], C[0,30] \}$

$= 20$

b_2: $C[3,50] = \max\{ 30 + \underbrace{C[2,20]}_{a_4}, \underbrace{C[2,50]}_{b_4} \} = 70$ $\quad w_i \leq W$, $30 \quad 50$ $\quad (20 \quad 70)$

a_4: $C[2,20] = C[1,20]$ $\quad w_i > W$

$= \max\{ 20 + C[0,15], C[0,20] \}$

$= 20$

b_4: $C[2,50] = \max\{ 50 + \underbrace{C[1,10]}_{a_5}, \underbrace{C[1,50]}_{b_5} \} = 70$

a_5: $C[1,10] = \max\{ 20 + C[0,5], C[0,10] \}$

$= 20$

b_5: $C[1,50] = \max\{ 20 + C[0,15], C[0,50] \}$

Algorithm:-

Dynamic_knapsack (v, w, n, W)

```
1. for i ← 0 to W
2. do  c[0,i] ← 0
3. for j ← 1 to n
4. do  c[j,0] ← 0
5. for i ← 1 to n
6.    do for j ← 1 to W
7.       do if ( j < wi )
8.          then c[i,j] ← c[i-1, j]
9.          else if ( vi + c[i-1, j-wi] ≥ c[i-1, j] )
10.              then c[i,j] ← vi + c[i-1, j-wi]
11.                   else c[i,j] ← c[i-1, j]
```

Running time of 0/1 knapsack = $\underline{O(nW)}$

17/10/16

Bellmanford Algorithm :-

This algorithm finds single source shortest path problem. It is more general case in which weights can be +ve or -ive.

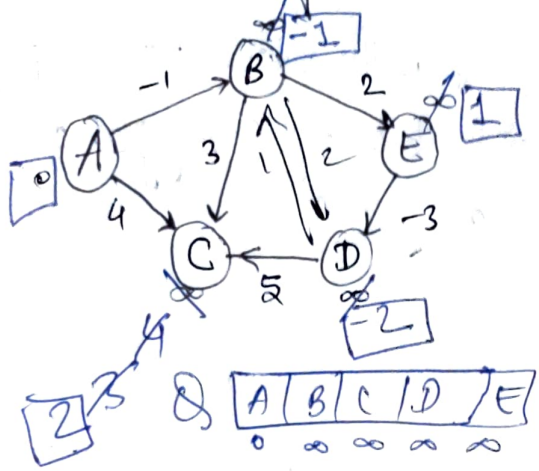
This algorithm returns a boolean value that indicate a -ive weight cycle exist or not. If there is such a cycle, then algorithm indicate that no solution exists.

```
Bellman-Ford (G, w, s)
1. initialise-single-source (G, s)
2. for i ← 1 to |V(G)| - 1
3. do for each edge (u,v) ∈ t(G)
4.       do relax (u, v, w)
5. for each edge (u,v) ∈ (G)
6. do if d[v] > d[u] + w(u,v)
7.    then return false
8. return true
```

```
initialise-single-source (G, s)
1. for each vertex v ∈ v[G]
2. do d[v] ← ∞
3.    π[v] ← nil
4. d[s] ← 0
```

```
Relax (u, v, w)
1. if d[v] > d[u] + w(u,v)
2. then d[v] ← d[u] + w(u,v)
3.      π[v] ← u
```

Q: Find the shortest path from vertex A to other vertices.



Relax edge AB:

$d[v] > d[u] + w(u,v)$

$\infty > 0 - 1$

Yes

$d[v] = -1$

Relax edge AC:

$d[v] > d[u] + w(u,v)$

$\infty > 0 + 4$

Yes

$d[v] = 4$

Relax edge BE:

$d[v] > d[u] + w(u,v)$

$\infty > -1 + 2$

Yes

$d[v] = +1$

Relax edge ED:

$\infty > 1 - 3$

Yes

$d[v] = -2$

Relax edge DC:

$4 > -2 + 5$

Yes.

$d[v] = 3$

Relax edge BC:

$3 > -1 + 3$

Yes.

$d[v] = 2$

Relax edge BD:

$-2 > -1 + 2$

false

Relax edge DB:

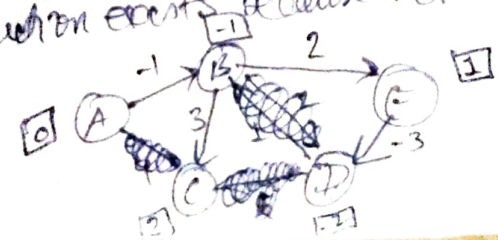
$-1 > -2 + 1$

false

→ Ans: $-1 > 0 - 1$ for all edges false.

false.

Here solution exists because returns true.

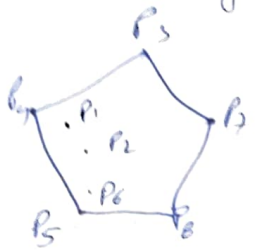


The algorithm return boolean value true. This means single source shortest path exists.

Convex Hull :-

The convex hull of a set Q of points is the smallest convex polygon P for which each points in Q either on the boundary of polygon or in its interior. We denote the convex-hull of Q as CH(Q).

e.g:



There are 2 algorithms that compute convex hull of set of n points.

1. Graham's Scan Algorithm:- which runs in $O(n \log n)$ times
2. Jarvis-March Algorithm:- which runs in $O(nh)$ times, where h is no. of vertices of CH(Q).

1. Graham's Scan Algorithm:-

It solves the convex-hull problem by maintaining a stack S. of candidate points.

```
GRAHAM'S_SCAN(S)
1. Let P0 be the point in Q with minimum y coordinate or leftmost such point
   in case of tie.
2. Let <P1, P2, ..., Pn> be the remaining points in Q sorted by angle in
   counter clockwise order around P0.
3.    PUSH(P0, S)
4.    PUSH(P1, S)
5.    PUSH(P2, S)
6.    for i <- 3 to n
7.    do while the angle formed by the points NEXT_TO_TOP(S)
8.          TOP(S) and Pi makes a non left turn.
            do POP(S).
9.    PUSH(Pi, S)
10.   Return S
```