Explain general greedy method

The general greedy method can be applied to variety of problems, most problem contain n inputs & requires the constraints. Any subset that satisfies these constraints is called feasible solution. The feasible soln is used to either maximizes or minimizes the objective function. A feasible solution that does this is called an optimal solution. It considering one input at a time.

The algorithm for the greedy method is as follow, It is top down approach.

The moto of this method is find out required optimal solution.

```
1. Algorithm Greedy (a,n)
2. // a[1:n] contains the n inputs.
3. {
4.    solution := ϕ; // Initialize the solution
5.    for i=1 to n do
6.    {
7.       x := select (a);
8.       if feasible (solution, x) then
9.       solution := union (solution, x);
10.   }
11.   return solution;
12. }
```

Explain knapsack problem & job sequencing with deadli in details.

Knapsack problems are solved by using the general greedy method, In knapsack problem we have given n objects & a knapsack or bag. where object i having the weight wi & the bag having the capacity of m. objective of this method is to find out maximum profit with the capacity of that bag.

If a ~~fraction~~ function x_i; $0 \le x_i \le 1$ then profit is calculated by $p_i x_i$; the objective is to find maximum of total profit.

According to all above data the knapsack problem can be solved as,

$$\text{maximum} \sum_{1 \le i \le n} p_i x_i \text{ are subject to}$$

$$\sum_{1 \le i \le n} w_i x_i \le m \;\&$$

$$0 \le x_i \le 1 \;, \; 1 \le i \le n$$

Where weight & profit are pasitive numbers.

A feasible solution or filling is any set $(x_1, x_2 \ldots x_n)$ are satisfy above equation,

example: In a problem

$n=3$

$m=20$

$(P_1, P_2, P_3) = (25, 24, 15) \rightarrow P_i$

$(w_1, w_2, w_3) = (10, 18, 15) \rightarrow w_i$

We have to find feasible solution i.e (x_1, x_2, x_3)

$\sum w_i$

1) $10/10, 10/18, 0/15 = (1, 5/9, 0)$

2) $10/10, 0, 10/15 = (1, 0, 2/3)$

3) $0, 18/18, 2/15 = (0, 1, 2/15)$

4) $2/10, 18/18, 0/15 = (1/5, 1, 0)$

5) $0/10, 5/18, 15/15 = (0, 5/18, 1)$

6) $5/10, 0, 15/15 = (1/2, 0, 1)$

Now according to above filling we find the weight & profit as follow.

$\sum p_i x_i$

① $25 \times 1 + 24 \times \frac{5}{9} = 25 + \frac{120}{9}$

$= 38.33$

② $25 \times 1 + 24 \times 0 + 2/3 \times 15$

$= 35.00$

③ $25 \times 0 + 1 \times 24 + 2/5 \times 15$

$= 28.00$

④ $2/5 \times 25 + 24 \times 1 = 34.00$

$\sum w_i x_i$

① $1 \times 10 + 0 \times 18 + 15 \times 2/3$

$= 20$

② $1 \times 10 + 0 \times 18 + 2/3 \times 15$

$= 20$

③ $1 \times 10 + 5/9 \times 18 + 0 \times 15$

$= 20$

④ $\frac{1}{5} \times 10 + 1 \times 18 + 0 \times 15$

$= 20$

⑤ $5/18 \times 24 + 1 \times 15 = 21.33$ $0 \times 10 + 5/18 \times 18 + 1 \times 15 = 20$

⑥ $12.5 + 15 = 27.5$ $1/2 \times 10 + 0 \times 18 + 1 \times 15 = 20$

from all over data the 38.33 is optimal solution hence it is maximum profit.

Algorithm for greedy strategies for the knapsack problem

```
Algorithm Greedy knapsack (m,n)
// p[1:n] and w[1:n] contains the profit &
// weight respectively of n object ordered
// such that p[i]/w[i] >= p[i+1]/w[i+1].
// m is the knapsack size &
// x[1:n] is the solution vector.
{
  for i:=1 to n do x[i]:= 0.0 ; //initialize x;
  U:=m
  for i:=1 to n do
  {
    If (w[i]>U) then break;
    x[i] := 1.0; U:= U-w[i] ;
  }
if (i<=n) then x[i]:= U/w[i];
}
```

Job sequencing with deadlines

We are given a set of n jobs, Associated with job i is an integer deadlines $d_i \geq 0$ & profit $P_i > 0$. for any job i the profit P_i is earned iff the job is completed by its deadlines.

To complete a job, one has to process the job on a machine for one time (unit time). Only one machine is available for processing jobs, The feasible solution for this problem is a subset J of jobs such that each job in this subset can be completed by it's deadlines. The value of feasible solution J is the sum of the profit of the job in J or $\sum_{i \in J} P_i$, An optimal solution is a feasible solution with maximum value.

Algorithm:

```
Algorithm JS(d,j,n)
/* d[i] ≥ 1; 1≤i≤n are the deadlines, n≥1
The jobs are ordered such that P[1] ≥ P[2] ..... ≥ P[n].
J[i] is the ith job in optimal solution. 1≤i≤k.
Also at termination d[J[i]] ≤ d[J[i+1]], 1≤i<k */
{
    d[0] = J[0] = 0 // Initialize
    J[1] = 1;       // Include job 1
    k = 1;
    for i = 2 to n do
    {
        // Consider jobs in nondecreasing order of
P[i] find
        // Position for i & check feasibility of insertion
        r := k;
        while ((d[J[r]] > d[i]) & (d[J[r]] ≠ r))
```

```
    do r=r-1;
    if ((d[J[r]]≤d[i]) & (d[i]>r) then
    {
      //Insert i into J[]
      for q=k to (r+1) step-1 do J[q+1] := J[q];
      J[r+1]; i; k=k+1;
    }
  }
  return k;
}
```

Example:

n=4 // No of jobs

$(P_1, P_2, P_3, P_4) = (100, 10, 15, 27)$ // Profit

$(d_1, d_2, d_3, d_4) = (2, 1, 2, 1)$ // Deadlines

Now by the given formula,

No	feasiable solution	Processing sequence	value $\sum_{i \in J} P_i$
1	(1,2)	2,1	170
2	(1,3)	(1,3) or (3,1)	115
3	(1,4)	(4,1)	127
4	(2,3)	2,3	25
5	(3,4)	(4,3)	42
6	(1)	(1)	100
7	(2)	(2)	10
8	(3)	(3)	15
9	(4)	(4)	27

In above only jobs 1, 4 are processes & hence the value is 127 which is the optimal solution.

These jobs must be processed in the ordered job4 followed by job1. Thus the processing time of job4 is zero & that of job1 is completed at time 2.

Job sequencing with deadlines

finding maximum profit for given set of profit & deadlines.

Designing a multiprogramming operating system.

① Example:

Solve the problem by job sequencing with deadlines.

No of objects = 4

profit = (100, 10, 15, 27)

Deadline = (2, 1, 2, 1)

According to profit arrange the objects.

object	Profit	Deadline
O_1	100	2
O_4	27	1
O_3	15	2
O_2	10	1

No of jobs to be selected = Maximum deadline.

We have to select = 2 jobs.

job O_4 & O_2 having same deadlines select O_4 with maximum profit = 27

from O_1 & O_3 select O_1 = 100

Total profit = 127, optimal solution (1,001)

Q2) Solve job sequence with deadline problem using greedy approach for, n=5

$(P_1, P_2, \ldots\ldots P_5) = (45, 15, 20, 7, 65)$

$(d_1, d_2, \ldots\ldots d_5) = (1, 3, 2, 1, 2)$

object	Profit	deadline
O_1	45	1
O_2	15	3
O_3	20	2
O_4	7	1
O_5	65	2

If we arrange given problem with profit:

O	P	D
O_5	65	2
O_1	45	1
O_3	20	2
O_2	15	3
O_4	7	1

As we given maximum deadline as 3.
we have to select 3 object.

1) One from deadline 1
2) " " 2
3) " " 3

from deadline 1 = 45 (O_1)
" " 2 = 65 (O_5)
from deadline 3 = 15 (O_2)

125

Profit = 125, feasible, optimal soln = (1,1,0,0,1)

find the maximum profit such a way that knapsack problem.

$n=3$, $m=25$

$(P_1, P_2, P_3) = (10, 20, 30)$

$(w_1, w_2, w_3) = (12, 13, 9)$

Now we have know,

$m=25$, $n=3$ maximum weight,

So feasible solution will be;

1) (12, 13, 0)
2) (12, 4, 9)
3) (3, 13, 9)
4) (12, 13, 0)
5) (12, 4, 9)
6) (3, 13, 9)

(12, 13, 0)
(12, 4, 9)
(3, 13, 9)

Now feasible solution will be,

(1, 1, 0) ; (1, 4/13, 1), (1/4, 1, 1)

Now we know that,

Profit = $\sum P_i x_i$

weight = $\sum w_i x_i$

1) $10\times1 + 20\times1 + 30\times0$
$= 30$

$1\times12 + 1\times13 + 0\times9$
$= 25$

2) $10\times1 + 20\times 4/13 + 30\times1$
$= 10 + 30 + 6.1$
$= 46.1$

$1\times12 + 4/13\times13 + 1\times9$
$= 25$

3) $10\times 1/4 + 20\times1 + 30\times1$
$2.5 + 50$
$= 52.5$

$12\times 1/4 + 13\times1 + 9\times1$
$= 25$

Maximum profit 52.5, hence it is optimal solution

Q3) Solve the knapsack problem using n=4.

$(w_1, w_2, w_3, w_4) = (10, 15, 6, 9)$

$(P_1, P_2, P_3, P_4) = (2, 5, 8, 1)$

Capacity of knapsack is m = 25.

object	Profit	weight	P/w
O_1	2	10	0.20
O_2	5	15	0.33
O_3	8	6	1.33
O_4	1	9	0.11

If we arrange P/w ratio in decending.

object	P/w
O_3	1.33
O_2	0.33
O_1	0.20
O_4	0.11

Now we know that bag containing maximum 25.

By taking profit & weight ratio into consideration

O_3 object => profit = 8 weight = 6

upto O_2 object => profit = 8+5 weight = 6+15 = 21

Now we want 4 items to fullfill the take 4 items upto from O_1

$8 + 5 + 0.2 \times 8$

$= 13.8$

Total maximum profit = 13.8

knapsack capacity m = 25

optimal solution $(4/10, 1, 1, 0)$

④ find optimal solution of knapsack.

$n=6$, $c=20$

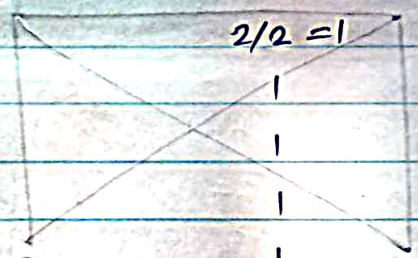
$(P_1, P_2, \ldots\ldots P_6) = (12, 5, 15, 7, 6, 18)$

$(w_1, w_2, \ldots\ldots w_6) = (2, 3, 5, 7, 1, 5)$

object	Profit	weight	P/w
O_1	$P_1 = 12$	2	6
O_2	$P_2 = 5$	3	1.67
O_3	$P_3 = 15$	5	3
O_4	$P_4 = 7$	7	1
O_5	$P_5 = 6$	1	6
O_6	$P_6 = 18$	5	3.6

If we arrange object in sequence of decreasing order then,

		W	Taken Object = 20	
O_1	6	2	2	2/2 = 1
O_5	6	1	1	1
O_6	3.6	5	5	1
O_3	3	5	5	1
O_2	1.67	3	3	1
O_4	1	7	7-3 = 4	4/7
			$\Sigma = 20$	



Profit $= 6\times2 + 6\times1 + 3.6\times5 + 3\times5 + 1.67\times3 + 1\times4$

$= 12 + 6 + 18 + 15 + 5 + 4$

Profit $= 60$

optimal solution = $(1, 1, 1, 4/7, 1, 1)$

Minimum Cast Spanning Tree:

1) No. of vertices in graph equals to no. of vertices in tree.

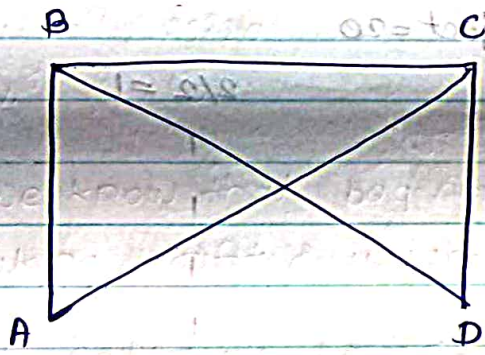
2) The edges in tree should be laser than graph.

3) All vertices should be connected.

4) No. cycle is found.

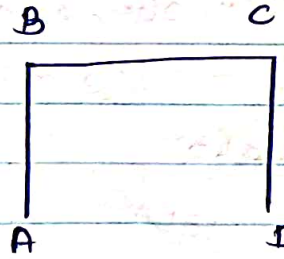
5) If graph contain n vertices then tree should contain atleast (n-1) edges, should be undirected graph.

eg:

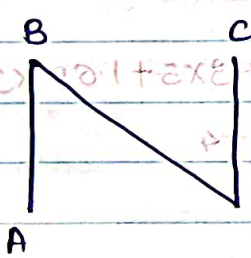


* some possible spanning tree

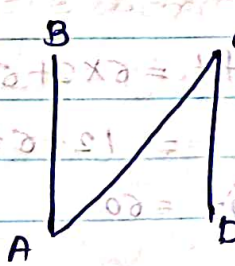
1)



2)



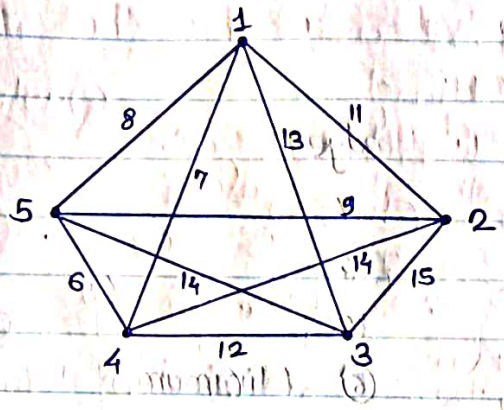
3)



Primes Algorithm :

1) Initially, node is selected & processes.
2) final Adjecency matrix.

eg: find Minimal spanning tree of graph.

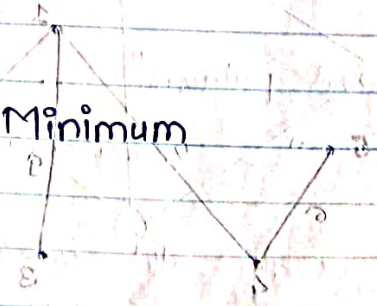


Now adjacency matrix [5X5]

$$M = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} \infty & 11 & 13 & 7 & 8 \\ 11 & \infty & 15 & 14 & 18 \\ 9 & 15 & \infty & 12 & 14 \\ 7 & 14 & 12 & \infty & 6 \\ 8 & 13 & 14 & 6 & \infty \end{bmatrix} \end{array}$$

1) Select Node 1

path from	edges
1 - 2	13
1 - 3	9
1 - 4	(7) Minimum
1 - 5	8



2) Select Node 2 :

path from	edge
2 - 1	(11) Minimum
2 - 3	15
2 - 4	14
2 - 5	13

Select Node 3:

path from	edge	
3 - 1	⑨	Minimum
3 - 2	15	
3 - 4	12	
3 - 5	14	

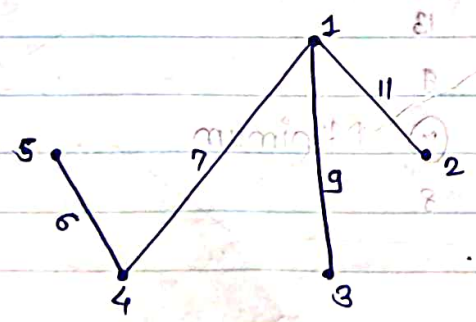
Select Node 4:

path from	edge	
4 - 1	7	
4 - 2	14	
4 - 3	12	
4 - 5	⑥	Minimum

~~Select~~ Select Node 6:

path from	edge	
5-1	⑧	
5-2	13	But all nodes are covered
5-3	14	
5-4	6	X selected

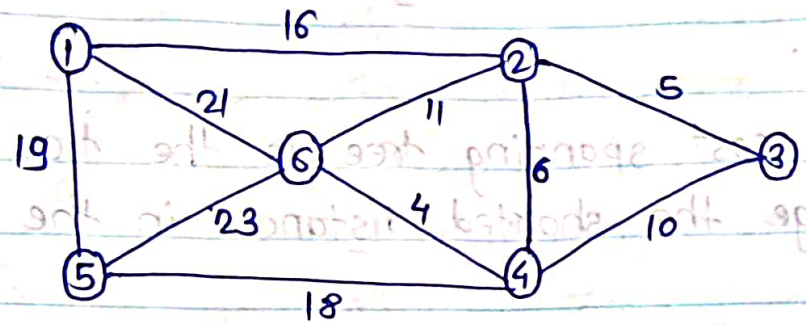
Spanning Tree



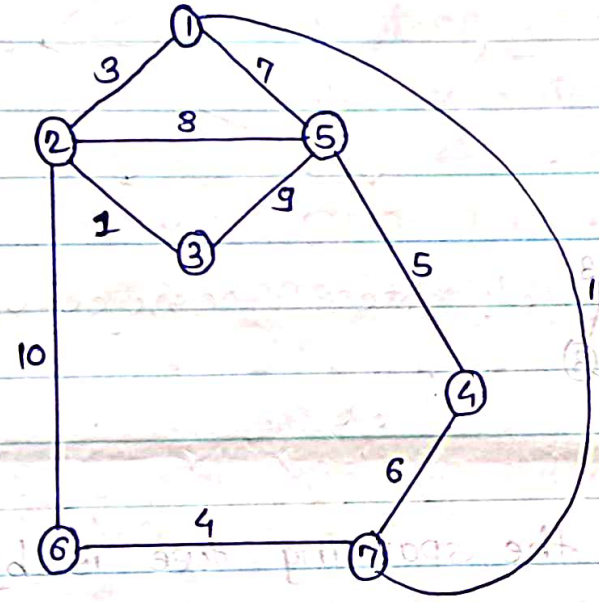
=33

example:

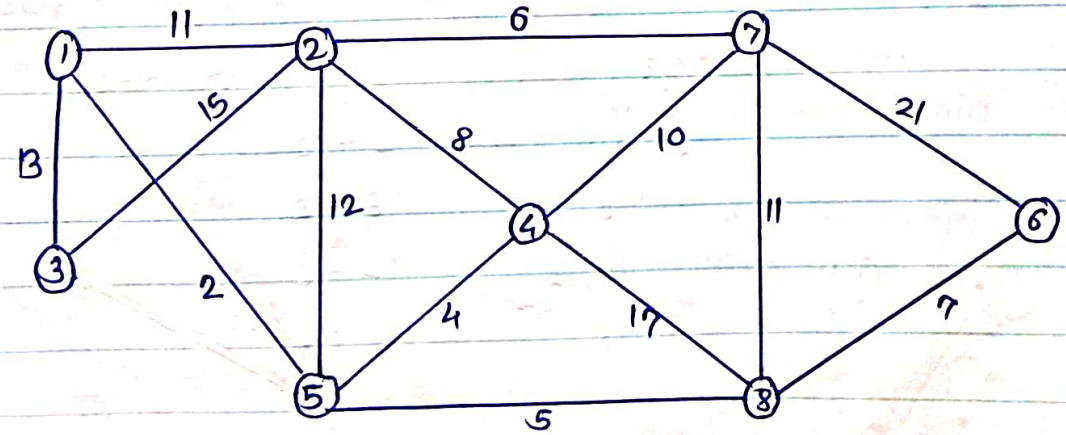
①


16
21
11
5
19
6
23
4
10
18

②


3
7
8
9
1
5
11
10
6
4

③

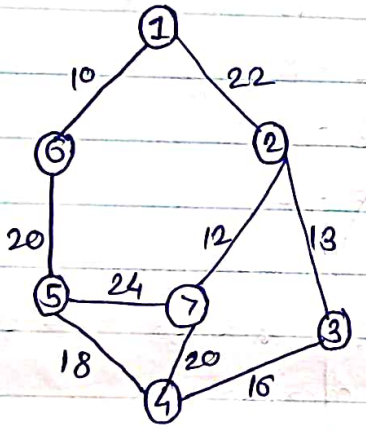

11
6
15
8
10
21
B
12
11
2
4
17
7
5

What is minimum cast spanning tree & explain with kruskal algorithm.

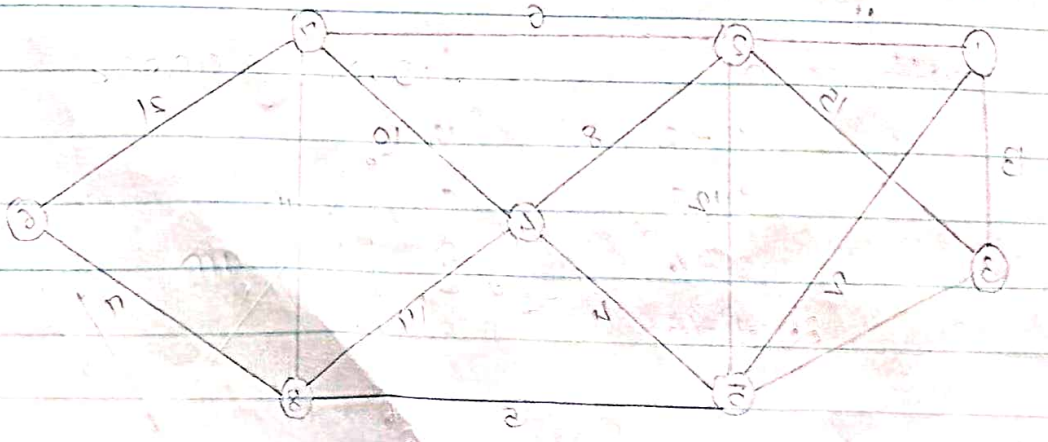
The minimum cast spanning tree is the tree by using it we can manage the shorted distance in the given nodes.

According to the krouskals algorithm the smallest edge is used for the tree.

Let us consider the following tree.



from above diagram the spanning tree is by krouskals algorithm as follows.

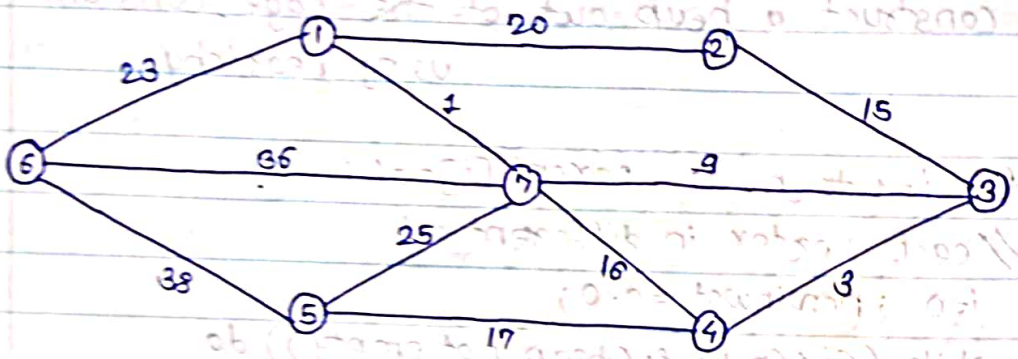


Algorithm for kruskals

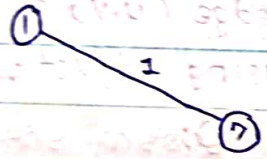
```
kruskal (E, cost, n, t)
{
 construct a heap out of the edge constant
                              using hearichy
for i=1 to n do  parent[i]=-1;
//each vertex in different set
 i=0 ; minwst =0.0;
while ((i<(n-1) & (heap not empty)) do
{
Delete a minimum wt edge (u,v) & from
 the heap & breaphy using adjust;
 j= find (u) ; k;= find (v);
  if (j≠k) then
  {
    i=i+1;
    t[i,1]=u ; t[i,1]=v;
    minwt:= minwst + wst [u,v];
    union (j,k);
  }
 }
  if (i≠n-1) then write ("No spanning tree");
  else return mincost;
}
```

Kruskal's algorithm:

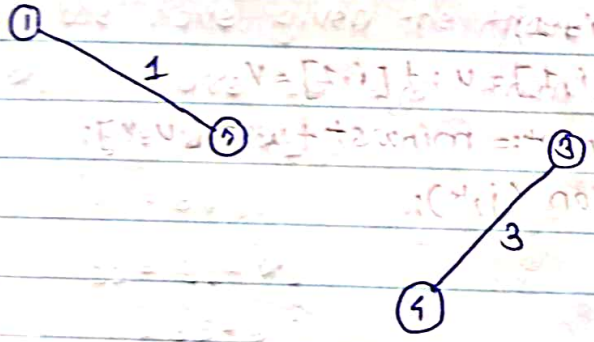
finding the lightest weight & go in increasing order without cycle.



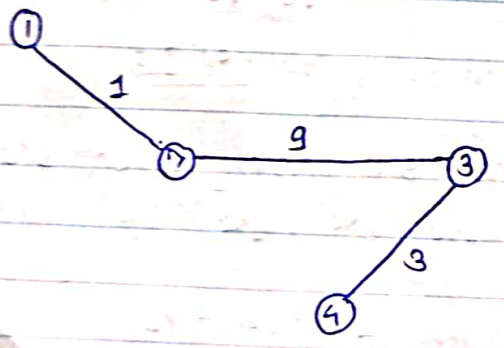
step ①



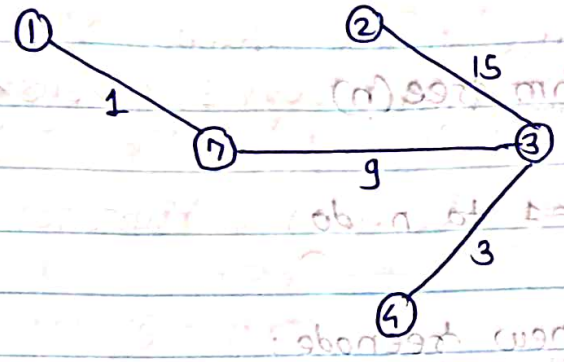
step ②



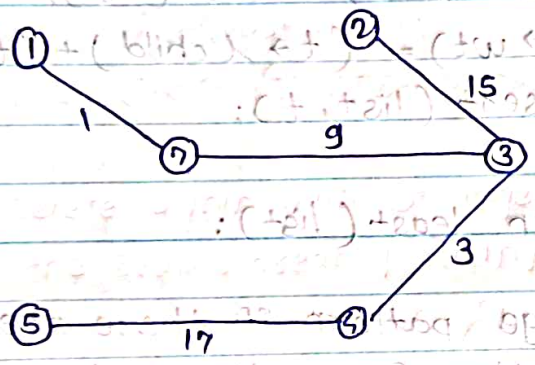
step ③



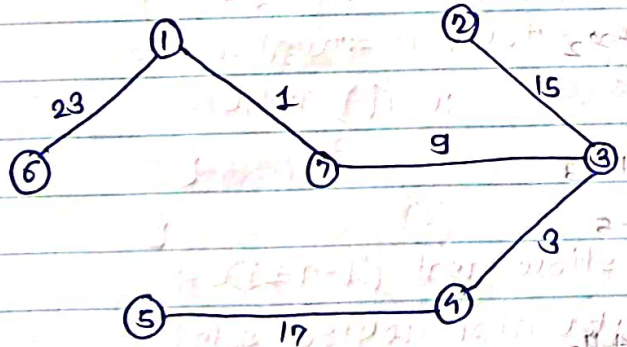
step ④



step ⑤



step ⑥



Minimum Cost ⇒ 1+3+9+15+17+23

= 68

Explain optimal merge pattern and huffmanns code.

Algorithm:

```
Algorithm tree(n)
//
 for i=1 to n do
 {
   t= new treenode;
   (t → l child) = least(list);
   (t → r child) = least (list);
   (t → wt) = (t→ l child) + (t→ r child)→wt));
    Insert (list, t);
 }
  return least (list);
```

In the merge pattern if there are two inputs then is no any problem for sorting but if merge than 2 Nodes are given then we have to merge such problem.

Let we have given the three inputs in such a way that x_1, x_2, x_3 are 20, 15, 5 then the,

$y_1 = x_1 + x_2$

$= 35$

$y_2 = y_1 + x_3$

$= 35 + 5$

$= 40$

$y_3 = y_1 + y_2$

$= 75$

If we have given the merge files as,

x_1, x_2, x_3, x_4 then

$y_1 = x_1 + x_2$

$y_2 = y_1 + x_3$

$y_2 + x_4 =$ sorted files

example:

$(x_1, x_2, x_3, x_4, x_5) = (20, 30, 10, 5, 30)$

Now By optimal merge pattern from above example we choose the minimum value such as,

$$y_1 = x_3 + x_4$$
$$= 10 + 5$$
$$= 15$$

$$y_2 = y_1 + x_1$$
$$= 35$$

Similarly $y_3 = x_2 + x_5$
$$= 30 + 30$$
$$= 60$$

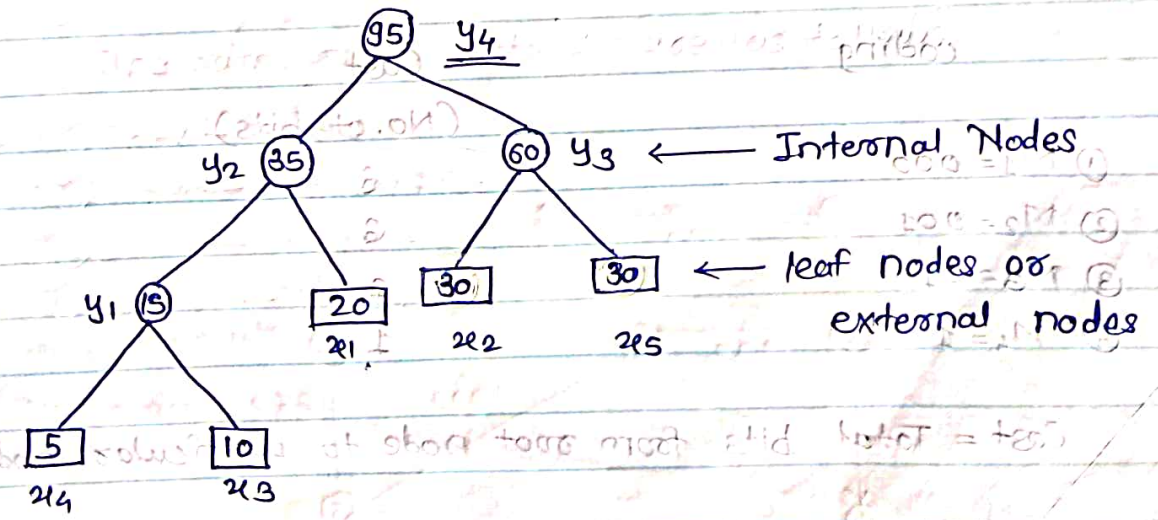
Hence,

$$y_4 = y_2 + y_3$$
$$= 35 + 60$$

$$\boxed{y_4 = 95}$$

Hence it is the optimal merge pattern of given problem

Tree is as follows



* Huffman's Code:

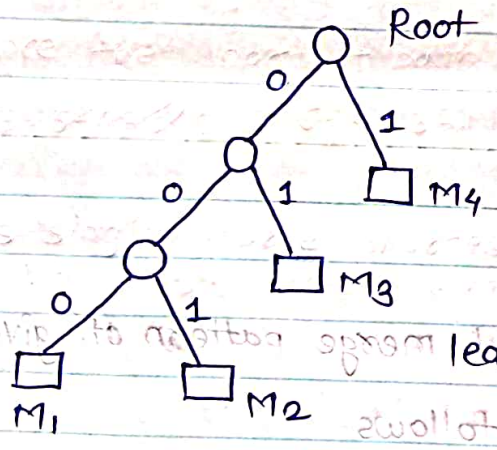
It depend upon frequency of occurance If this code are received at the received side then decoded by decode tree, decode tree is binary tree in which external nodes are represents the messages.

Eg: M_1, M_2, M_3, M_4

In huffmanns code the left branch is represented by 0 bit while Right is represented by 1 bit

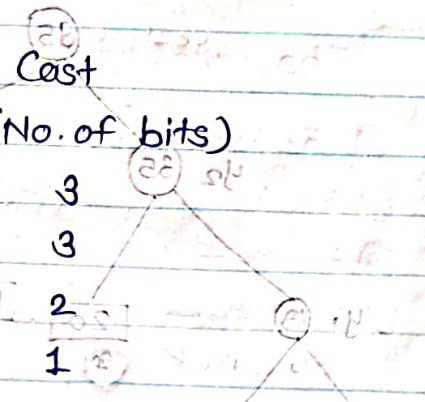
Huffmans code containing the distance from the nodes of external node.

The distance is represented by the total no. of 0 or 1 bit from each ~~eqv~~ sequencing node.



Coding

① $M_1 = 000$
② $M_2 = 001$
③ $M_3 = 01$
④ $M_4 = 1$



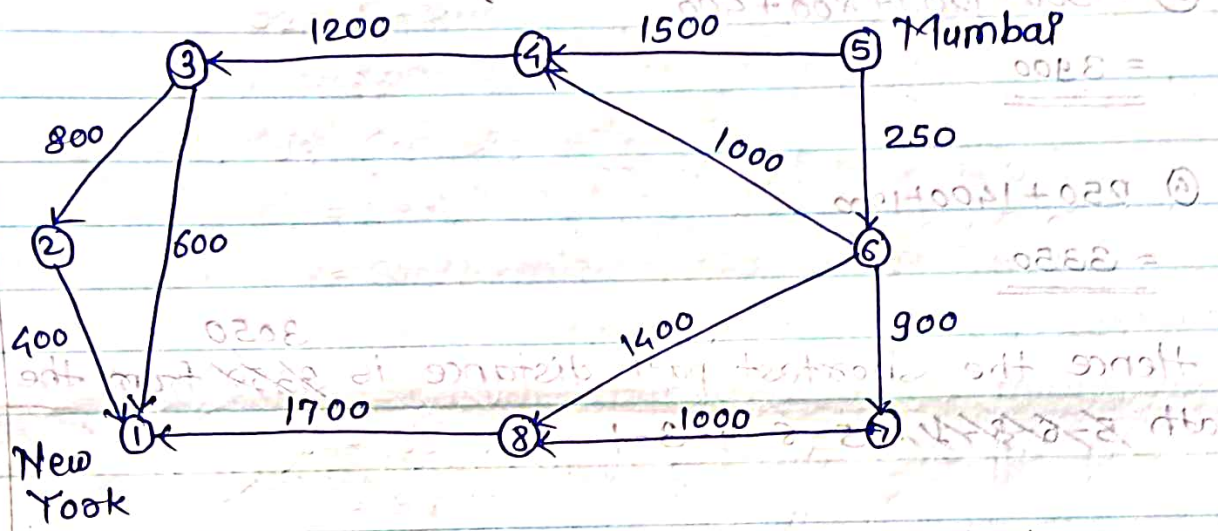
Cost = Total bits from root node to particular node.

Explain single source shortest path.

The single source shortest path is used as it's name. It used for the measuring the distances from any two station.

In the diagram we have given the nodes which represents the name of the city & edge shows the distance between that two cities.

example:



from the above Diagram we have to calculate the shortest path from point 5 to 1 i.e we have to find distance from mumbai to NewYork.

The paths from 5 to 1 are as follows.

① 5-4-3-1.
② 5-4-3-2-1.
③ 5-6-7-8-1.
④ 5-6-8-1.
⑤ 5-6-4-3-2-1
⑥ 5-6-4-3-1

We having only 6 paths from 5 to 1 hence the values are,

① 1500 + 1200 + 600
= 3300

⑤ 250 + 1000 + 1200 + 800 + 400
= 3650

③ 250 + 900 + 1000 + 1700
= 3850

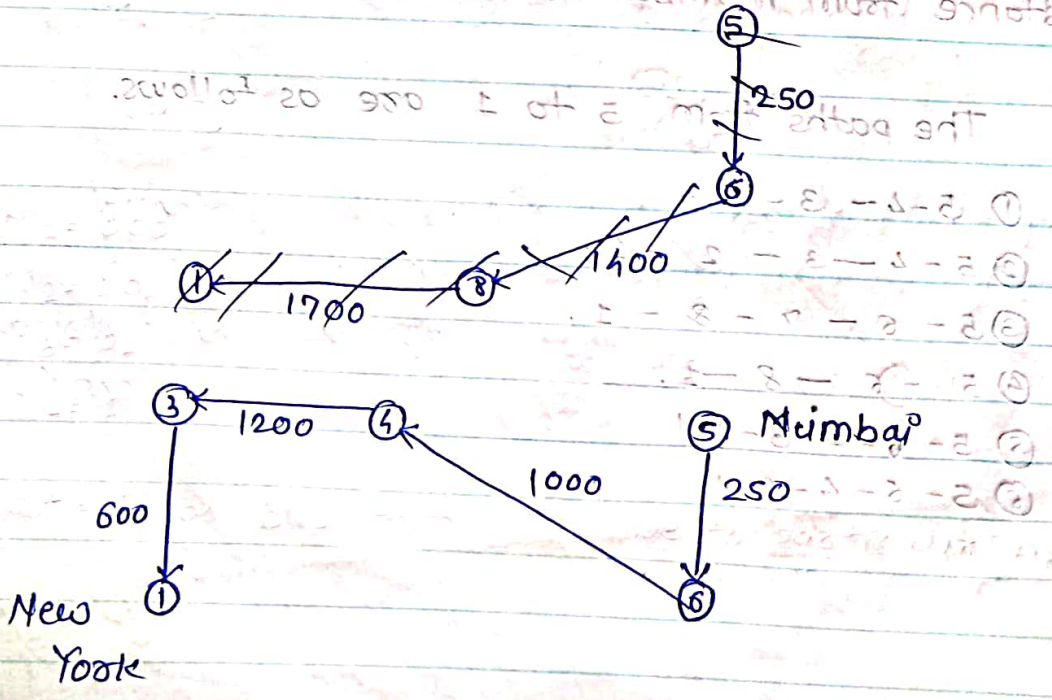
⑥ 250 + 1000 + 1200 + 600
= 3050

② 1500 + 1200 + 800 + 400
= 3900

④ 250 + 1400 + 1700
= 3350

Hence the shortest path distance is ~~3350~~ 3050 from the path ~~5-6-8-1~~ 5-6-4-3-1

It is single source shortest path from mumbai to New York.



```
Algorithm shortest path (v, cost, dist, n)
{
    for i=1 to n do;
    {
        // Initialize S
        S[i] = false;
        dist[i] = cost[v,i];
    }
    S[v] = true;
    dist[v] = 0.0;
    for num = 2 to n-1 do
    {
        // Determine n-1 paths from v
        choose u from among those vertices not
        in S such that dist[u] is minimum;

        S[u] := true // put u in S.
        for (each w| adjacent to u with S[w] = false) do
        // update distances
        if (dist[w] > dist[u] + cost[u,w]) then
            dist[w] = dist[u] + cost[u,w];
    }
}
```

* Optimal storage on Tapes:

Algorithm :

```
Algorithm store (n, m)
  // n = number of program
  // m = number of tapes
  {
    j = 0;
    for i = 1 to n do
    {
      write(" append program", i,
      " to permutation for tape", j);
      j = (j+1) mod m;
    }
  }
```

If there are n programs that are to be stored on a computer tape of length L, Associated with each program I is length l_i, $1 \le i \le n$.

The all programs could be stored on tape iff sum of length of the program is at most L.

We assume that however a program is to be retrived from this tape the tape is initially positioned at the front. Hence if programs are stored in the order.

$$I = i_1, i_2, \ldots\ldots\ldots i_n.$$

The time t_j required to retrive the program i_j, If all programs are retrived equally then excepted mean retrived time MRT.

In MRT the taps are stored depend upon permutation.

While minimizing MRT we should minimize the d[I).

Example:

let n=3 & $(l_1, l_2, l_3) = (5, 10, 3)$

Now n=3

possible ordeerings = n! = 6

possible ordeerings I will be,

(1,2,3) , (1,3,2) , (2, 1,3), (2, 3, 1)
(3, 1, 2) , (3, 2,1)

ordeerings	d(I)	
(1, 2, 3)	5+5+10+5+3+10	= 38
(1, 3, 2)	5+5+3+5+10+3	= 31
(2, 1, 3)	10+10+5+10+3+5	= 43
(2, 3, 1)	10+10+3+10+5+3	= 41
(3, 2, 1)	3+3+10+3+5+10	= 34
(3, 1, 2)	3+3+5+3+10+5	= 29