# 3.GREEDY METHOD

# Topics:-

1.Greedy methodology

2.Applications

3.Knapsack Problem

4.Minimum cost spanning trees

5.Single source shortest path problem

6.Huffman trees

# GREEDY METHODOLOGY:-

Every problem have some constraints and objective functions

## OBJECTIVE FUNCTION:

It is an attempt to express a business goal (or) a function that is desired to be maximized or minimized.

Any subset that specifies the given constraints is called feasible solution.

## FEASIBLE SOLUTION:

If the given problem has ‘n’ inputs then the subset of inputs satisfies the constraints of particular problem is called feasible solution.

## OPTIMAL SOLUTION:

A feasible solution that either maximizes or minimizes the objective function is called optimal solution.

In greedy method,we works in stages.

At each stage,we take one input at a time and make a decision, either it gives optimal solution or not.

A decision made in one stage cannot be changed in later stages.

i.e,there is no backtracking.

## ALGORITHM:

Algorithm greedy(a,n) || a[1:n] contains ‘n’ inputs

{

Solution:= ¶; || initialize the solution

For i:=1 to n do

{

x :=select(a);

if feasible(solution,x)then

solution:=union(solution,x)

}

return solution

# KNAPSACK PROBLEM:-

We have 'n' objects and a knapsack or bag.Each object has weight 'Wi' and profit 'Pi' and knapsack has capacity 'm'.

Objective is filling of Knapsack that maximises the total profit earned.So the problem can be stated as maximise

∑PiXi subject to ∑ WiXi <= m

1<=i<=n 1<=i<=n and 0<=Xi<=1 , 1 <= i <= n To Compute maximum profit, we take some solution factor i.e Xi.

If object directly placed Xi = 1 (If Enough space is available)

Otherwise Xi=0.

If object does not fit in the knapsack but some amount of space is available ,

Xi = Remaining Space/Actual Weight of Object. ALGORITHM :

Algorithm knapsack(p,w,n)

//p[1:n] & w[1:n] are profits and weights of objects such that

Pi/Wi > P(i+1)/W(i+1) > ......

{

for i := 1 to n do

x[i] := 0 (u -> capacity of bag)

m = u;

for i := 1 to n do {

if(w[i] > m) break;

x[i] := 1;

m = u - w[i];

}

if(i <= n) then

x[i] = m / w[i] ;

}

\*Time Complexity is O(n).

EXAMPLE:

1. No. of objects n = 3, m = Capacity of knapsack=20

(p1,p2,p3) = (25,24,15)

(w1,w2,w3)=(18,15,10)

Fill the bag with Maximum Profit using Knapsack greedy method

sol:

Given,

n=3,m=20

Our main aim is to fill the bag with Maximum Profit

Case 1: (Maximum Profit)

First we place Maximum profit object

p1 = 25

w1=18

x1=1

After placing First Object

M = 20 - 18 = 2

p2 = 24

w2 = 15

x2 = 2/15

There is no space available => x3 = 0

Maximize

∑PiXi => p1x1 + p2x2 + p3x3

1<=i<=n =>25(1)+24(2/15)+15(0)

=>25+3.2

=>28.2

Case 2: (Minimum Weight)

Among the above three onjects W3 weight is minimum

w3=10

M=20-10=10

x3=1 [Fitted]

Next object is w2=15

x2 = 10/15

Bag is Full

So,x1=0

Here,∑ Pixi = p1x1 + p2x2 + p3x3

1<=i<=n

=> 25(0)+24(10/15)+15(1)

=>0+16+15

=>31

Case 3:

Maximum Profit per unit weight:

p1/w1 = 25/18 = 1.4

p2/w2 = 24/15 = 1.6

p3/w3 = 15/10 = 1.5

We place an item in bag whose profit per weight (p/w) is maximum i.e. p2/w2 is maximum so that x2=1

M= 20-15=5

x3 = 5/10=0.5

x1=0

=>25(0)+24(1)+15(0.5)

=>24+7.5

=>31.5

Therefore,

The Maximum profitable feasible solution that maximises profit is 31.5

Therefore, The optimal solution is (0,1,1/2)

Time Complexity is O(n).

# MINIMUM COST SPANNING TREE:

## TREE:

**It is a connected undirected acyclic graph.let G = ( V , E) be an**

**connected graph , then the subgraph t = (V , E') of G is a spanning tree iff 't' is**

**a tree.**

## WEIGHTED GRAPH:

**A collection of vertices , edges and also weights on the**

**edges then the graph is said to be weighted graph.**

## SPANNING TREE:

**Any tree consisting of all vertices of a graph , then it is**

**called a spanning tree.**

## MINIMUM COST SPANNING TREE:

**A spanning tree with minimum cost is**

**called minimum cost spanning tree.**

**In case of complete graph the possible number of spanning trees are n^(n – 2).**

**To find the minimum cost spaning tree we will use two standard algorithms.**

* **prims Algorithm**
* **krushkal's Algorithm**

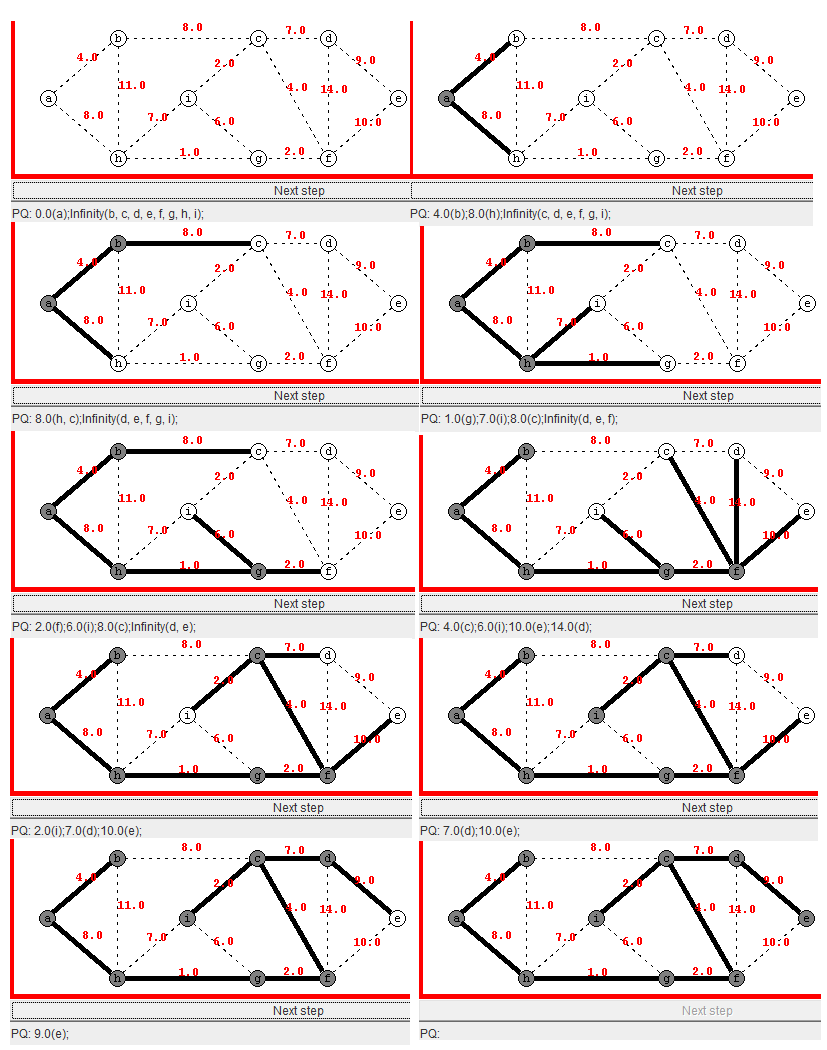
## 1.)PRIMS ALGORITHM:

**Step 1: Take minimum cost edges from all the edges**

**Step 2: Take another edge having minimum weight among those vertices which**

**are adjacent to previous edge.**

**Step 3: Repeat the above process untill spanning tree contains (n – 1) edges.**

**EXAMPLE:**

**Total cost = 4.0 + 8.0 + 1.0 + 2.0 + 2.0 + 11.0 + 7.0 + 4.0 = 39**

**ALGORITHM PRIMS ( G ):**

**//INPUT: A weighted connected graph G = <V , E>**

**//OUTPUT: T minimum spanning tree of G**

**{**

**V = { V0}**

**T = { } // empty graph**

**for i = 1 to n – 2 {**

**choose nearest neighbour Vj of V that is adjacent to Vi**

**Vi belongs to V and for which edge e i j = (Vi ,Vj ) does not form a cycle with members of T.**

**V = V union { Vj }**

**T = T union { e i j }**

**}**

**return T;**

**}**

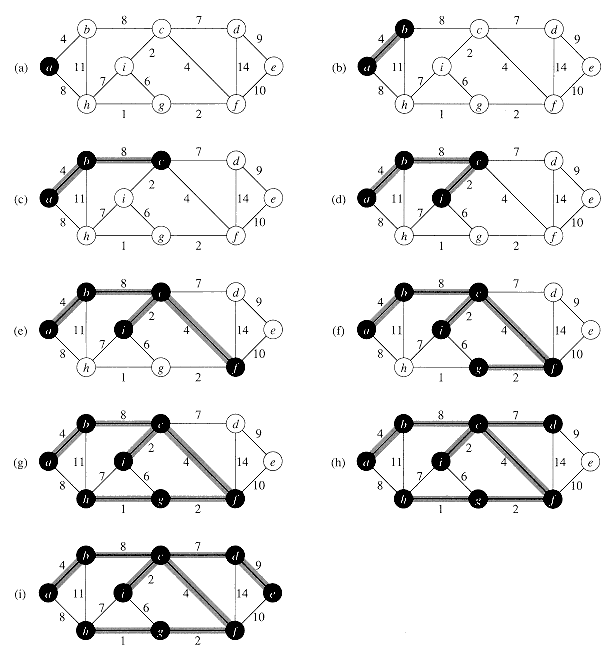
## 2.)KRUSHKAL'S ALGORITHM:

**Step 1: Take minimum cost edges from all the edges**

**Step 2: Take minimum weight edge among all remaining (n – 1 ) edges**

**Step 3: Repeat the above process untill spanning tree contains (n – 1) edges.**

**EXAMPLE:**



**Total cost = 4 + 8 + 7 + 9 + 2 + 4 + 1 + 2 = 37**

**ALGORITHM PRIMS ( G ):**

**//INPUT: A weighted connected graph G = <V , E>**

**//OUTPUT: T minimum spanning tree of G**

**{**

**T = { } // empty graph**

**for i = 1 to n – 2 do {**

**e := any edge in G with smallest weight that does not form a cycle when added to T**

**T := T union { e }**

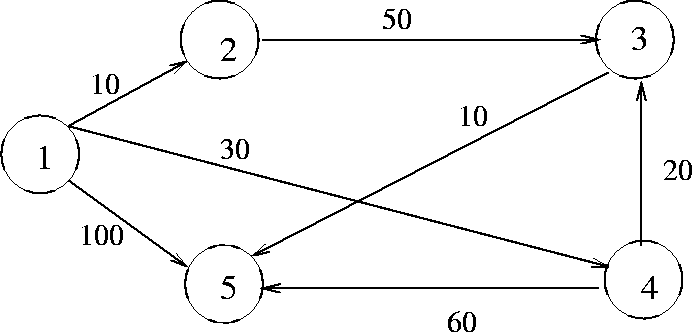
**}**

**return T;**

**}**

## SINGLE SOURCE SHORTEST PATH PROBLEM

Graphs can be used to represent distance between states or country with vertices representing cities and edge representing sections of highway. The edges can be assign weights which may be either distances between two cities connected by edge or average time to drive along that section of highway.



Start from source vertex

->Set s[1] = true;

{1,2} = 10;

{1,3} = ∞;

{1,4} = 30;

{1,5} = 100;

Now shortest distance from vertex 1 is10 i.e{1,2}

and set[2] = true;

{1,2,3} = 60;

{1,2,4} = ∞;

{1,2,5} =∞;

Now shortest distance from vertex 2 is 50 i.e{1,2,3} and set[3] = true;

{1,2,3,4} = ∞;

{1,2,3,5} = 70;

Now shortest distance from vertex 3 is 10 i.e{1,2,3,5}

and selected vertex is 5;

set s[5] = true;

destination vertex 5 is achieved through shortest path 1-2-3-5 with path length 70

ANALYSIS :- single source shortest path O(N^2) times

## ALGORITHM:-

Algorithm shortest path(v,cost,dist,n){

For i:= 1 to n do{

s[i] := false;

dist[i] := cost[v,i];

}

s[v] = true;

dist[v] =0.0;

for num:= 2 to n do{

// determine n-1 paths from v

Choose u from among those vertices not in s(not visible)

Such that dist[u] is minimum

dist[u] := min{dist[i]}

s[u]:= true;

for (each w adjacent to u with s[w] = false) do

if(dist[w]>dist[u]+cost[u,w]) then

dist[w] = dist[u]+cost[u,w];

}

}

# HUFFMAN TREES:

Huffman codes are used in communication theory in the field of data compression for a communication to be happened related symbols must be encoded using any coding technique this task is carried by an encoder .

The encoder assigns a unique addresses to all the symbols and the a message is transmitted across the channel as series of zeroe’s and one’s.

* The message is received and decoded by a decoder at the receiver side
* Coding plays an important role in the coomunication
* The most popular codes that are used in communication theories are ASCII,Binary green etc.,
* The Huffman code is a popular code as well as a variable address code
* It assigns a code to symbols based on important of the code
* Huffman codes are constructed using Greedy Approach

## PROCEDURE:

1. List the symbols and stored them according to there frequency in ascending order.
2. Pick two symbols having last frequency and create a new node by adding probabilities of two symbols and label new node with it.
3. Step 2 is repeated until all symbols are computed.
4. For every internal nodes start assigning values i.e 0 for left child and 1 for right child.
5. For every internal node construct codeword by tracking code from root node to leaf node.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| SYMBOL | A | B | C | D |
| FREQUENCY | 16 | 12 | 3 | 4 |

SYMBOLS ARE SORTED IN ASCENDING ODER AS FOLLOWS

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| SYMBOL | C | D | B | A |
| FREQUENCY | 3 | 4 | 12 | 16 |

1. Apply Step 2.

* NOTE: Pick the two atleast frequencies C,D the

Combine both the frequency 7.

## EXAMPLE:

NOW THE LIST OF FREQUENCIES 7,12,16.

C D

Again

B

C D

Now the list of frequencies available are 16,19

Code of each symbol can traced from root to leaf node as follows:

A = 10

B = 11

C = 100

D = 101

Analysis - O(nlogn) - Time Complexity