GREEDY METHOD

- Greedy method is the most straightforward designed technique.
- As the name suggest they are short sighted in their approach taking decision on the basis of the information immediately at the hand without worrying about the effect these decision may have in the future.

DEFINITION:

- A problem with N inputs will have some constraints .any subsets that satisfy these constraints are called a feasible solution.
- A feasible solution that either maximize can minimize a given objectives function is called an optimal solution.

Control algorithm for Greedy Method:

```
1.Algorithm Greedy (a,n)
2.//a[1:n] contain the 'n' inputs
3. {
4.solution =0;//Initialise 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.}
```

* The function select an input from a[] and removes it. The select input value is assigned to X.

- Feasible is a Boolean value function that determines whether X can be included into the solution vector.
- The function Union combines X with The solution and updates the objective function.
- The function Greedy describes the essential way that a greedy algorithm will once a particular problem is chosen ands the function subset, feasible & union are properly implemented.

Example

- Suppose we have in a country the following coins are available :

Dollars(100 cents)
Quarters(25 cents)
Dimes(10 cents)
Nickel(5 Cents)
Pennies(1 cent)

- Our aim is paying a given amount to a customer using the smallest possible number of coins.
- For example if we must pay 276 cents possible solution then,

→ 1 doll+7 q+ 1 pen➔9 coins
→ 2 doll +3Q +1 pen➔6 coins
→ 2 doll+7dim+1 nic +1 pen➔11 coins.

KNAPSACK PROBLEM

- we are given n objects and knapsack or bag with capacity M object I has a weight Wi where I varies from 1 to N.

- The problem is we have to fill the bag with the help of N objects and the resulting profit has to be maximum.

- Formally the problem can be stated as

  Maximize xipi subject to XiWi<=M
  Where Xi is the fraction of object and it lies between 0 to 1.

- There are so many ways to solve this problem, which will give many feasible solution for which we have to find the optimal solution.

- But in this algorithm, it will generate only one solution which is going to be feasible as well as optimal.

- First, we find the profit & weight rates of each and every object and sort it according to the descending order of the ratios.

- Select an object with highest p/w ratio and check whether its height is lesser than the capacity of the bag.

- If so place 1 unit of the first object and decrement .the capacity of the bag by the weight of the object you have placed.

- Repeat the above steps until the capacity of the bag becomes less than the weight of the object you have selected .in this case place a fraction of the object and come out of the loop.

- Whenever you selected.

ALGORITHM:

1.Algorityhm Greedy knapsack (m,n)
2//P[1:n] and the w[1:n]contain the profit

```
3.// & weight res'.of the n object ordered.
4.//such that p[i]/w[i] >=p[i+1]/W[i+1]
5.//n is the Knapsack size and x[1:n] is the solution vertex.
6.{
7.for I=1 to n do a[I]=0.0;
8.U=n;
9.For I=1 to n do
10.{
11.if (w[i]>u)then break;
13.x[i]=1.0;U=U-w[i]
14.}
15.if(i<=n)then x[i]=U/w[i];
16.}
```

Example:

Capacity=20
N=3 ,M=20
Wi=18,15,10
Pi=25,24,15

Pi/Wi=25/18=1.36,24/15=1.6,15/10=1.5

Descending Order ➔ Pi/Wi➔1.6 1.5 1.36
Pi = 24 15 25
Wi = 15 10 18
Xi = 1 5/10 0
PiXi=1*24+0.5*15➔31.5

The optimal solution is ➔31.5

X1	***X2***	***X3***	***WiXi***	***PiXi***
½	1/3	¼	16.6	24.25
1	2/5	0	20	18.2
0	2/3	1	20	31
0	1	½	20	31.5

Of these feasible solution Solution 4 yield the Max profit .As we shall soon see this solution is optimal for the given problem instance.

JOB SCHEDULING WITH DEAD LINES

The problem is the number of jobs, their profit and deadlines will be given and we have to find a sequence of job, which will be completed within its deadlines, and it should yield a maximum profit.

Points To remember:

- To complete a job, one has to process the job or a action for one unit of time.
- Only one machine is available for processing jobs.
- A feasible solution for this problem is a subset of j of jobs such that each job in this subject can be completed by this deadline.
- If we select a job at that time ,

→Since one job can be processed in a single m/c. The other job has to be in its waiting state until the job is completed and the machine becomes free.

→So the waiting time and the processing time should be less than or equal to the dead line of the job.

ALGORITHM:

```
Algorithm JS(d,j,n)
//The job are ordered such that p[1]>p[2]…>p[n]
//j[i]  is  the ith job in the optimal solution
// Also at terminal d [ J[ i]<=d[ J {i+1],1<i<k
 {
  d[0]= J[0]=0;
J[1]=1;
K=1;
For I =1 to n do
{ // consider jobs in non increasing  order of P[I];find the position for I and check feasibility insertion
r=k;
while((d[J[r]]>d[i] )and
  (d[J[r]] ≠  r)do r =r-1;
if (d[J[r]]<d[I])and (d[I]>r))then
{
for q=k to (r+1) step –1 do J [q+1]=j[q]
J[r+1]=i;
K=k+1;
}
}
return k;
}
```

Example :

1. n=5 (P1,P2,…P5)=(20,15,10,5,1)
   (d1,d2….d3)=(2,2,1,3,3)

Feasible solution ***Processing Sequence*** ***Value***

(1)	(1)	20
(2)	(2)	15
(3)	(3)	10
(4)	(4)	5
(5)	(5)	1
(1,2)	(2,1)	35
(1,3)	(3,1)	30
(1,4)	(1,4)	25
(1,5)	(1,5)	21
(2,3)	(3,2)	25
(2,4)	(2,4)	20
(2,5)	(2,5)	16
(1,2,3)	(3,2,1)	45
(1,2,4)	(1,2,4)	40

The Solution 13 is optimal

2. n=4 (P1,P2,…P4)=(100,10,15,27)
(d1,d2….d4)=(2,1,2,1)

Feasible solution	***Processing Sequence***	***Value***
(1,2)	(2,1)	110
(1,3)	(1,3)	115
(1,4)	(4,1)	127
(2,3)	(9,3)	25
(2,4)	(4,2)	37
(3,4)	(4,3)	42
(1)	(1)	100
(2)	(2)	10
(3)	(3)	15
(4)	(4)	27

The solution 3 is optimal.

MINIMUM SPANNING TREE

- Let G(V,E) be an undirected connected graph with vertices ‘v’ and edge ‘E’.
- A sub-graph t=(V,E’) of the G is a Spanning tree of G iff ‘t’ is a tree.3
- The problem is to generate a graph G’= (V,E) where ‘E’ is the subset of E,G’ is a Minimum spanning tree.
- Each and every edge will contain the given non-negative length .connect all the nodes with edge present in set E’ and weight has to be minimum.

NOTE:

- We have to visit all the nodes.
- The subset tree (i.e) any connected graph with 'N' vertices must have at least N-1 edges and also it does not form a cycle.

Definition:

- A spanning tree of a graph is an undirected tree consisting of only those edge that are necessary to connect all the vertices in the original graph.
- A Spanning tree has a property that for any pair of vertices there exist only one path between them and the insertion of an edge to a spanning tree form a unique cycle.

Application of the spanning tree:

1. Analysis of electrical circuit.
2. Shortest route problems.

Minimum cost spanning tree:

- The cost of a spanning tree is the sum of cost of the edges in that trees.
- There are 2 method to determine a minimum cost spanning tree are

1. Kruskal's Algorithm
2. Prom's Algorithm.

KRUSKAL'S ALGORITHM:

In kruskal's algorithm the selection function chooses edges in increasing order of length without worrying too much about their connection to previously chosen edges, except that never to form a cycle. The result is a forest of trees that grows until all the trees in a forest (all the components) merge in a single tree.

- In this algorithm, a minimum cost-spanning tree 'T' is built edge by edge.
- Edge are considered for inclusion in 'T' in increasing order of their cost.

    - An edge is included in 'T' if it doesn't form a cycle with edge already in T.
    - To find the minimum cost spanning tree the edge are inserted to tree in increasing order of their cost

Algorithm:

```
Algorithm kruskal(E,cost,n,t)
//E→set of edges in G has 'n' vertices.
//cost[u,v]→cost of edge (u,v).t→set of edge in minimum cost spanning tree
// the first cost is returned.
{
```

```
for i=1 to n do parent[I]=-1;
I=0;mincost=0.0;
While((I<n-1)and (heap not empty)) do
{
j=find(n);
k=find(v);
if(j not equal k) than
{
i=i+1
t[i,1]=u;
t[i,2]=v;
mincost=mincost+cost[u,v];
union(j,k);
    }
  }
if(i notequal n-1) then write("No spanning tree")
else return minimum cost;
}
```

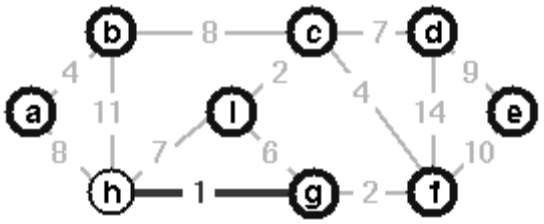
Analysis

- The time complexity of minimum cost spanning tree algorithm in worst case is O(|E|log|E|),

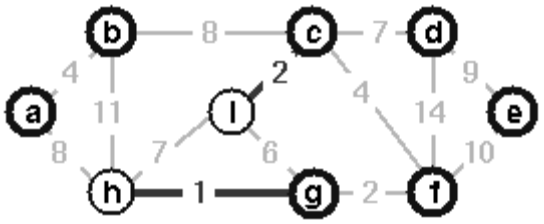
→where E is the edge set of G.

Example: Step by Step operation of Kurskal algorithm.

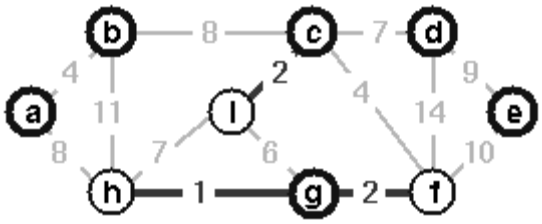
Step 1. In the graph, the Edge(g, h) is shortest. Either vertex g or vertex h could be representative. Lets choose vertex g arbitrarily.



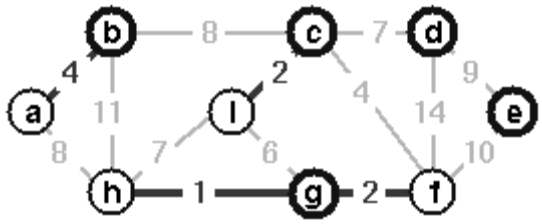
Step 2. The edge (c, i) creates the second tree. Choose vertex c as representative for second tree.



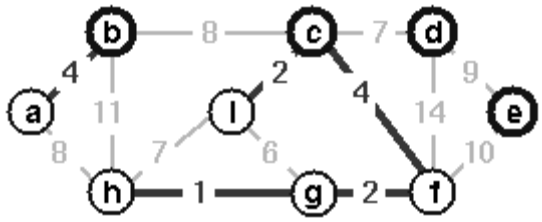
Step 3. Edge (g, g) is the next shortest edge. Add this edge and choose vertex g as representative.



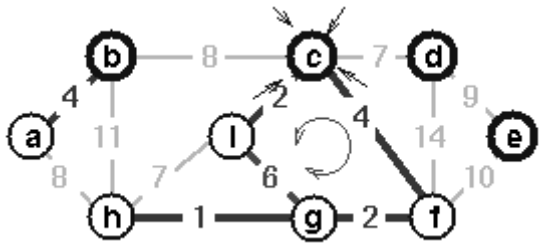
Step 4. Edge (a, b) creates a third tree.



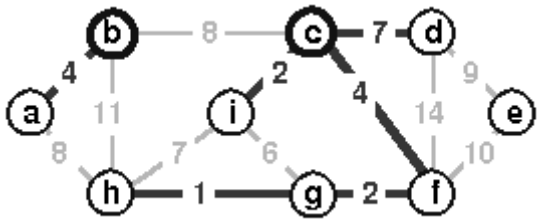
Step 5. Add edge (c, f) and merge two trees. Vertex c is chosen as the representative.



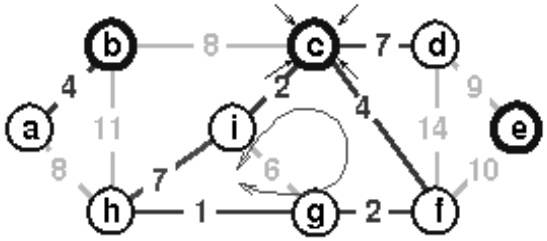
Step 6. Edge (g, i) is the next next cheapest, but if we add this edge a cycle would be created. Vertex c is the representative of both.



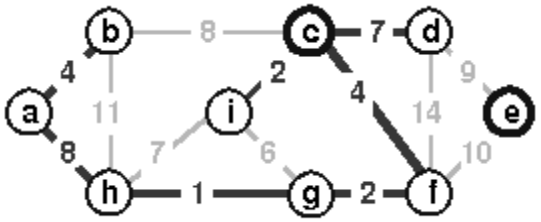
Step 7. Instead, add edge (c, d).



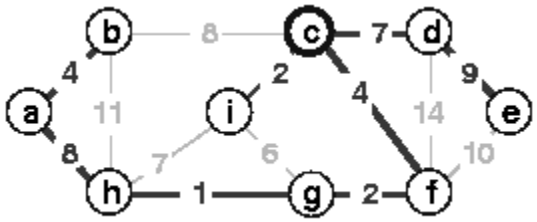
Step 8. If we add edge (h, i), edge(h, i) would make a cycle.



Step 9. Instead of adding edge (h, i) add edge (a, h).

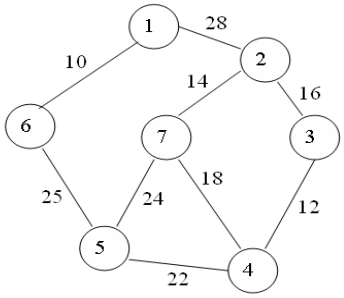


Step 10. Again, if we add edge (b, c), it would create a cycle. Add edge (d, e) instead to complete the spanning tree. In this spanning tree all trees joined and vertex c is a sole representative.



PRIM'S ALGORITHM

Start from an arbitrary vertex (root). At each stage, add a new branch (edge) to the tree already constructed; the algorithm halts when all the vertices in the graph have been reached.



Algorithm prims(e,cost,n,t)

{

Let (k,*l*) be an edge of minimum cost in E;

Mincost :=cost[k,*l*];

T[1,1]:=k; t[1,2]:=*l*;

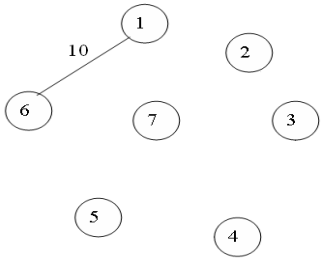
For I:=1 to n do

If (cost[i,*l*]<cost[i,k]) then near[i]:=*l*;
Else near[i]:=k;
Near[k]:=near[l]:=0;
For i:=2 to n-1 do
{
Let j be an index such that near[j]≠0 and
Cost[j,near[j]] is minimum;
T[i,1]:=j; t[i,2]:=near[j];
Mincost:=mincost+ Cost[j,near[j]];
Near[j]:=0;
For k:=0 to n do
If near((near[k]≠0) and (Cost[k,near[k]]>cost[k,j])) then
Near[k]:=j;
}
Return mincost;

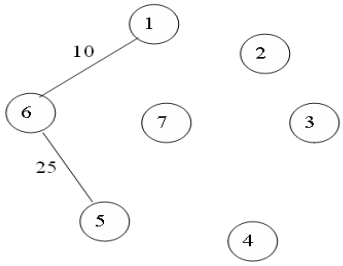
}

- The prims algorithm will start with a tree that includes only a minimum cost edge of G.
- Then, edges are added to the tree one by one. the next edge (i,j) to be added in such that I is a vertex included in the tree, j is a vertex not yet included, and cost of (i,j), cost[i,j] is minimum among all the edges.
- The working of prims will be explained by following diagram

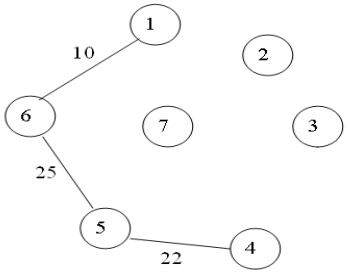
Step 1:



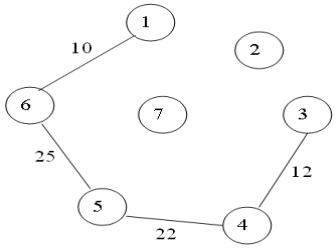
Step 2:



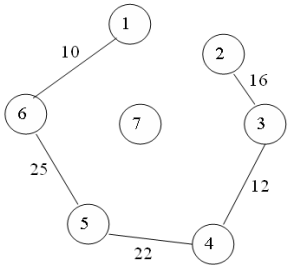
Step 3:



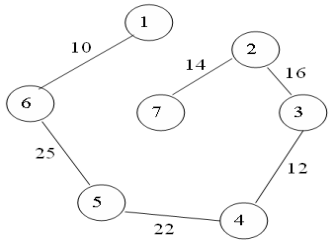
Step 4:



Step 5:



Step 6:



SINGLE SOURCE SHORTEST PATH

Single-source shortest path:

Graphs can be used to represent the highway structure of a state or country with vertices representing cities and edges representing sections of highway. The edges can then be assigned weights which may be either the distance between the two cities connected by the edge or the average time to drive along that section of highway. A motorist wishing to drive from city A to B would be interested in answers to the following questions:

1. Is there a path from A to B?
2. If there is more than one path from A to B? Which is the shortest path?

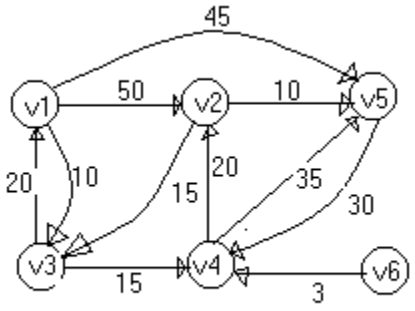


Fig 7.1

The problems defined by these questions are special case of the path problem we study in this section. The length of a path is now defined to be the sum of the weights of the edges on that path. The starting vertex of the path is referred to as the source and the last vertex the destination. The graphs are digraphs representing streets. Consider a digraph G=(V,E), with the distance to be traveled as weights on the edges. The problem is to determine the shortest path from v0 to all the remaining vertices of G. It is assumed that all the weights associated with the edges are positive. The shortest path between v0 and some other node v is an ordering among a subset of the edges. Hence this problem fits the ordering paradigm.

Example:
Consider the digraph of fig 7-1. Let the numbers on the edges be the costs of travelling along that route. If a person is interested travel from v1 to v2, then he encounters many paths. Some of them are

1. v1 v2 = 50 units
2. v1 v3 v4 v2 = 10+15+20=45 units
3. v1 v5 v4 v2 = 45+30+20= 95 units
4. v1 v3 v4 v5 v4 v2 = 10+15+35+30+20=110 units

The cheapest path among these is the path along v1 v3 v4 v2. The cost of the path is 10+15+20 = 45 units. Even though there are three edges on this path, it is cheaper than travelling along the path connecting v1 and v2 directly i.e., the path v1 v2 that costs 50 units. One can also notice that, it is not possible to travel to v6 from any other node.
To formulate a greedy based algorithm to generate the cheapest paths, we must conceive a multistage solution to the problem and also of an optimization measure. One possibility is to build the shortest paths one by one. As an optimization measure we can use the sum of the lengths of all paths so far generated. For this measure to be minimized, each individual path must be of minimum length. If we have already constructed i shortest paths, then using this optimization measure, the next path to be constructed should be the next shortest minimum length path. The greedy way to generate these paths in non-decreasing order of path length. First,

a shortest path to the nearest vertex is generated. Then a shortest path to the second nearest vertex is generated, and so on.
A much simpler method would be to solve it using matrix representation. The steps that should be followed is as follows,

Step 1: find the adjacency matrix for the given graph. The adjacency matrix for fig 7.1 is given below

	V1	V2	V3	V4	V5	V6
V1	-	50	10	Inf	45	Inf
V2	Inf	-	15	Inf	10	Inf
V3	20	Inf	-	15	inf	Inf
V4	Inf	20	Inf	-	35	Inf
V5	Inf	Inf	Inf	30	-	Inf
V6	Inf	Inf	Inf	3	Inf	-

Step 2: consider v1 to be the source and choose the minimum entry in the row v1. In the above table the minimum in row v1 is 10.

Step 3: find out the column in which the minimum is present, for the above example it is column v3. Hence, this is the node that has to be next visited.

Step 4: compute a matrix by eliminating v1 and v3 columns. Initially retain only row v1. The second row is computed by adding 10 to all values of row v3.
The resulting matrix is

	V2	V4	V5	V6
V1□ Vw	50	Inf	45	Inf
V1□ V3□ Vw	10+inf	10+15	10+inf	10+inf
Minimum	50	25	45	inf

Step 5: find the minimum in each column. Now select the minimum from the resulting row. In the above example the minimum is 25. Repeat step 3 followed by step 4 till all vertices are covered or single column is left.

The solution for the fig 7.1 can be continued as follows

	V2	V5	V6
V1□ Vw	50	45	Inf
V1□ V3□ V4□ Vw	25+20	25+35	25+inf
Minimum	45	45	inf

	V5	V6
V1□ Vw	45	Inf
V1□ V3□ V4□ V2□ Vw	45+10	45+inf
Minimum	45	Inf

	V6
V1□ Vw	Inf
V1□ V3□ V4□ V2□ V5□ Vw	45+inf
Minimum	inf

Finally the cheapest path from v1 to all other vertices is given by V1□ V3□ V4□ V2□ V5.

Optimal Storage on Tapes

Given programs stored on a computer tape and length of each program is where 1<=i<=n, find the order in which the programs should be stored in the tape for which the Mean Retrieval Time (MRT given as 1/n ∑i=1 to n ∑j=1 to I Li) is minimized.

Example:

Input : n = 3

L[] = { 5, 3, 10 }

Output : Order should be { 3, 5, 10 } with MRT = 29/3

Let us first break down the problem and understand what needs to be done.

A magnetic tape provides only sequential access of data. In an audio tape/cassette, unlike a CD, a fifth song from the tape can't be just directly played. The length of the first four songs must be traversed to play the fifth song. So in order to access certain data, head of the tape should be positioned accordingly.

Now suppose there are 4 songs in a tape of audio lengths 5, 7, 3 and 2 mins respectively. In order to play the fourth song, we need to traverse an audio length of 5 + 7 + 3 = 15 mins and then position the tape head.
Retrieval time of the data is the time taken to retrieve/access that data in its entirety. Hence retrieval time of the fourth song is 15 + 2 = 17 mins.

Now, considering that all programs in a magnetic tape are retrieved equally often and the tape head points to the front of the tape every time, a new term can be defined called the Mean Retrieval Time (MRT).

Let's suppose that the retrieval time of program is . Therefore,

MRT is the average of all such . Therefore ,

or

The sequential access of data in a tape has some limitations. Order must be defined in which the data/programs in a tape are stored so that least MRT can be obtained. Hence the order of storing becomes very important to reduce the data retrieval/access time.
Thus, the task gets reduced – to define the correct order and hence minimize the MRT, i.e. to

minimize the term

For e.g. Suppose there are 3 programs of lengths 2, 5 and 4 respectively. So there are total 3! = 6 possible orders of storage.

	ORDER	TOTAL RETRIEVAL TIME	MEAN RETRIEVAL TIME
1	1 2 3	2 + (2 + 5) + (2 + 5 + 4) = 20	20/3
2	1 3 2	2 + (2 + 4) + (2 + 4 + 5) = 19	19/3
3	2 1 3	5 + (5 + 2) + (5 + 2 + 4) = 23	23/3
4	2 3 1	5 + (5 + 4) + (5 + 4 + 2) = 25	25/3

5	3 1 2	4 + (4 + 2) + (4 + 2 + 5) = 21	21/3
6	3 2 1	4 + (4 + 5) + (4 + 5 + 2) = 24	24/3

It’s clear that by following the second order in storing the programs, the mean retrieval time is least.

In above example, the first program’s length is added ‘n’ times, the second ‘n-1’ times…and so on till the last program is added only once. So, careful analysis suggests that in order to minimize the MRT, programs having greater lengths should be put towards the end so that the summation is reduced. Or, the lengths of the programs should be sorted in increasing order. That’s the **Greedy Algorithm** in use – at each step we make the immediate choice of putting the program having the least time first, in order to build up the ultimate optimized solution to the problem piece by piece.