UNIT-II

Greedy method: General method, Knapsack problem, Job sequencing with deadlines, Minimum cost spanning trees, Optimal storage on tapes, Single source shortest paths.

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.**

GENERAL METHOD:

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 isassigned to X.Feasible is a Boolean value function that determines whether X can beincluded 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.

2.1. KNAPSACK PROBLEM

we are given **n** objects and knapsack or bag with capacity m object **i** has a weightWi where **i** varies from **1 to n**.

The problem is we have to fill the bag with the help of **n** objects and the resultingprofit has to be maximum.

Formally the problem can be stated as

Maximize $\sum_{1<=i<=n} x_i p_i$

subject to $\sum_{1<=i<=n} x_i w_i <= m$

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.Algorithm Greedy knapsack (m,n)

2//P[1:n] and the w[1:n]contain the profit3.//

& 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.

2.2. 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;

{ // 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;


}

}
```

return k;

}

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

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.

2.3 .MINIMUM COST 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.

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 thenodes 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-1edges and also it does not form a cycle.

Definition:

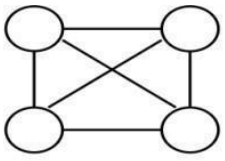
A spanning tree of a graph is an **undirected** tree consisting of only those edge thatare necessary to connect all the vertices in the original graph.

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.

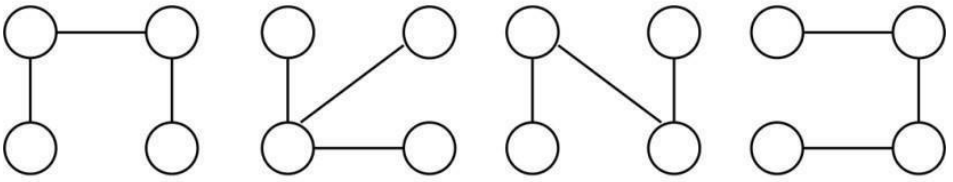
SPANNING TREE: - A Sub graph 'n' of o graph 'G' is called as a spanning tree if

(i) It includes all the vertices of 'G'

(ii) It is a tree



A connected,
undirected graph



Four of the spanning trees of the graph

Application of the spanning tree:

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

2.3.1 METHODS OF 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.**

2.3.2 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 increasingorder 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 (heapnot 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;

}

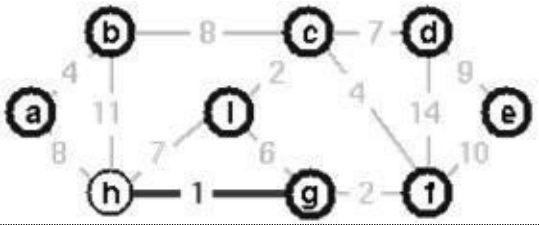
Analysis

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

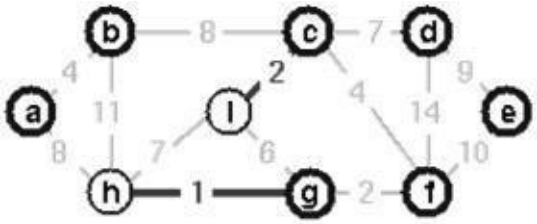
where E is the edge set of G.

Example: Step by Step operation of Kurskal algorithm.

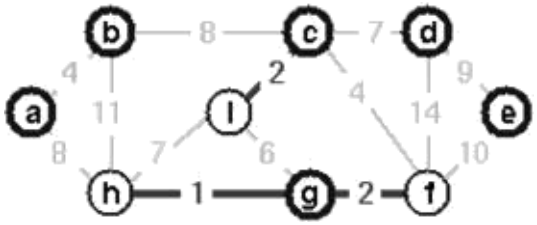
Step 1. In the graph, the Edge(g, h) is shortest. Either vertex g or vertex h could be representative. Lets choose vertex g arbitrarily.



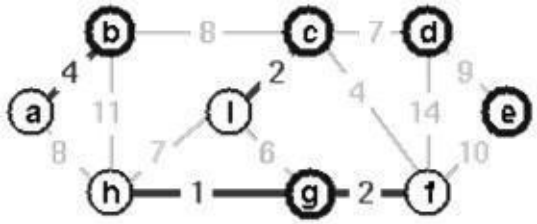
Step 2. The edge (c, i) creates the second tree. Choose vertex c as representative for second tree.



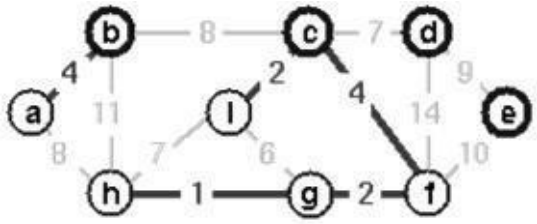
Step 3. Edge (g, g) is the next shortest edge. Add this edge and choose vertex g as representative.



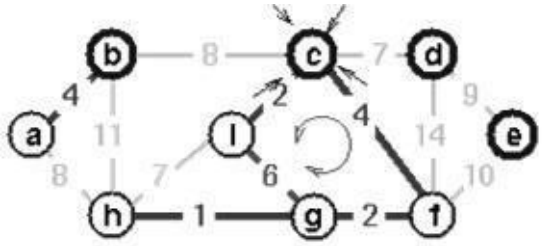
Step 4. Edge (a, b) creates a third tree.



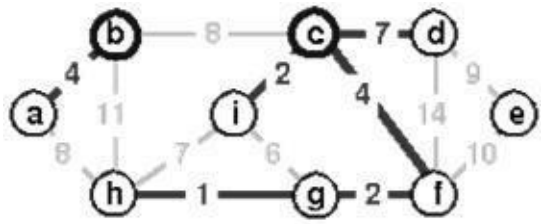
Step 5. Add edge (c, f) and merge two trees. Vertex c is chosen as the representative.



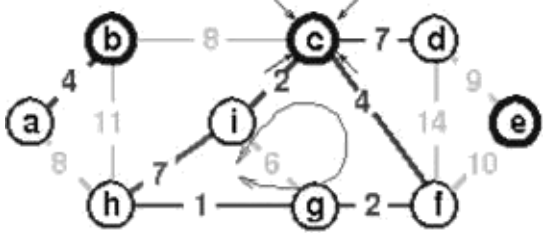
Step 6. Edge (g, i) is the next next cheapest, but if we add this edge a cycle would be created. Vertex c is the representative of both.



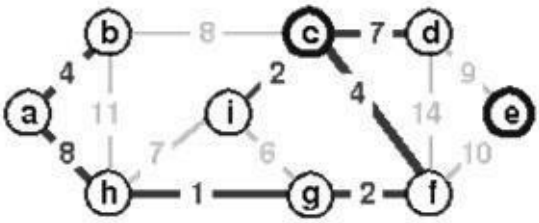
Step 7. Instead, add edge (c, d).



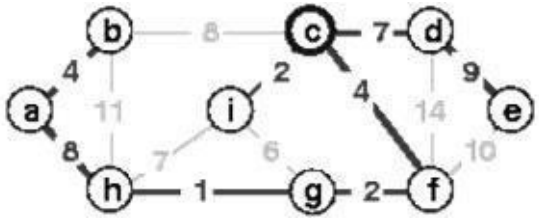
Step 8. If we add edge (h, i), edge(h, i) would make a cycle.



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

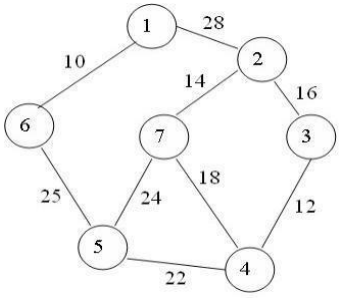


Step 10. Again, if we add edge (b, c), it would create a cycle. Add edge (d, e) instead to complete the spanning tree. In this spanning tree all trees joined and vertex c is a sole representative.



2.3.3PRIM'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.

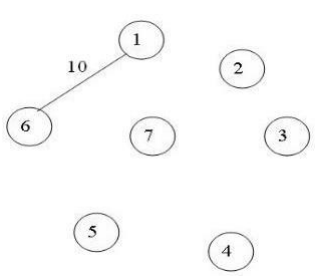
```
Algorithm prims(e,cost,n,t)
{
  Let (k,l) be an edge of minimum cost in E;
  Mincost :=cost[k,l];
  T[1,1]:=k; t[1,2]:=l;
  For I:=1 to n do
     If (cost[i,l]<cost[i,k]) then near[i]:=l;
     Else near[i]:=k;
 Near[k]:=near[l]:=0;
 For i:=2 to n-1 do
  {
    Let j be an index such that near[j]≠0 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;
}
```

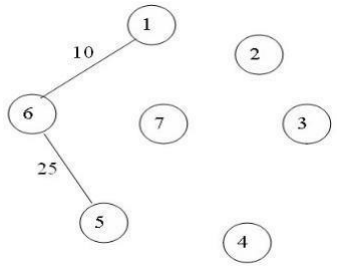
The prims algorithm will start with a tree that includes only a minimum cost edge of G.

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.

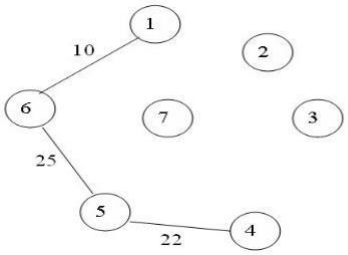
14. The working of prims will be explained by following diagram

Step 1: **Step 2:**

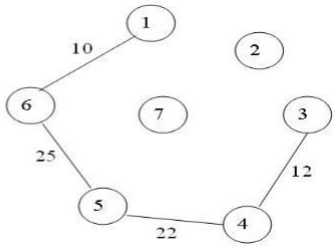




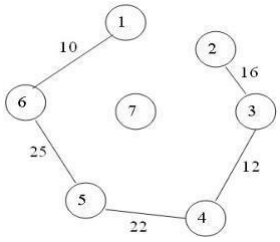
Step 3:



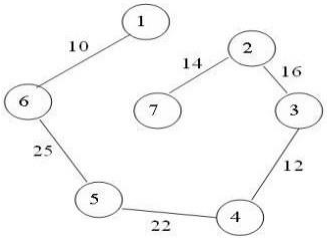
Step 4:



Step 5:



Step 6:



2.4 OPTIMAL STORAGE ON TAPES :

There are ‘n’ programs that are to be stored on a computer tape of length ‘L’. Each program ‘i’ is of length li, $1 \leq i \leq n$. All the programs can be stored on the tape if and only if the sum of the lengths of the programs is at most ‘L’. We shall assume that whenever a program is to be retrieved from this tape, the tape is initially positioned at the front. If the programs are stored in the order i = i1, i2, ... , i_n, the time tJ needed to retrieve program iJ is proportional to □ l i k 1□ k □ j 74 If all the programs are retrieved equally often then the expected or mean retrieval time (MRT) is: 1 . □ t n 1 □ J □ n j For the optimal storage on tape problem, we are required to find the permutation for the ‘n’ programs so that when they are stored on the tape in this order the MRT is minimized. d (I) □ n □ J □ 1 J □ K □ 1 l i k Example Let n = 3, (l1, l2, l3) = (5, 10, 3). Then find the optimal ordering? Solution: There are n! = 6 possible orderings.

They are:

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

3, 1, 2	3 + (3 + 5) + (3 + 5 + 10) = 29
3, 2, 1	3 + (3 + 10) + (3 + 10 + 5) = 34

From the above, it simply requires to store the programs in non-decreasing order (increasing order) of their lengths. This can be carried out by using a efficient sorting algorithm (Heap sort). This ordering can be carried out in O (n log n) time using heap sort algorithm. The tape storage problem can be extended to several tapes. If there are m □ 1 tapes, To,..... ,Tm – 1, then the programs are to be distributed over these tapes. m □1 The total retrieval time (RT) is □ J □0 d(IJ) The objective is to store the programs in such a way as to minimize RT. The programs are to be sorted in non decreasing order of their lengths li's, l1 < l2 < .. . . ln. The first 'm' programs will be assigned to tapes To, ...,Tm-1 respectively. The next 'm' programs will be assigned to T0, ... ,Tm-1 respectively. The general rule is that program i is stored on tape Ti mod m. 75 Algorithm: The algorithm for assigning programs to tapes is as follows: Algorithm Store (n, m) // n is the number of programs and m the number of tapes { j := 0; // next tape to store on for i :=1 to n do { Print ('append program', i, 'to permutation for tape', j); j := (j + 1) mod m; } } On any given tape, the programs are stored in non-decreasing order of their lengths

2.5 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:

- Is there a path from A to B?
- If there is more than one path from A to B? Which is the shortestpath?

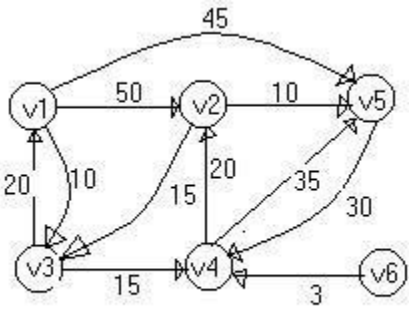


Fig 7.1

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

- v1 v2 = 50 units
- v1 v3 v4 v2 = 10+15+20=45 units
- v1 v5 v4 v2 = 45+30+20= 95 units
- 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. Thena 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	∞	45	∞
V2	∞	-	15	∞	10	∞
V3	20	∞	-	15	∞	∞
V4	∞	20	∞	-	35	∞
V5	∞	∞	∞	30	-	∞
V6	∞	∞	∞	3	∞	-

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	∞	45	∞
V1 V3 Vw	10+∞	10+15	10+∞	10+∞
Minimum	50	25	45	∞

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.