UNIT – 3 : GREEDY METHOD

UNIT STRUCTURE

3.1 LEARNING OBJECTIVE

After going through this unit, you will be able to:

- know about the greedy algorithm
- describe the greedy method applied in knapsack problem
- elaborate the job sequencing with deadline
- define minimum spanning tree problem
- describe application of minimum spanning tree problem in Prim's and Kruskal algorithm
- elaborate the shortest path problem using Dijkstra algorithm

3.2 INTRODUCTION

Greedy algorithm is typically used in optimization problem. Algorithm for optimization problems go through a sequence of steps. All of these problems have n inputs and require us to obtain a subset that satisfies some constraints. Any subset that satisfies some constraints is called feasible solution. The solution finds a given objective function which value is either maximizes or minimizes. A feasible solution that does this is called an optimal solution. In this unit, we will discuss about the concept of greedy methods and its application in various problems like Knapsack problems and minimum spanning tree etc.

3.3 GENERAL STRATEGY OF GREEDY ALGORITHM

A greedy algorithm always makes the choice that looks best at the moment. That is it makes a locally optimal choice that may be lead to a globally optimal solution. This algorithm is simple and more efficient compared to other optimization algorithm. This heuristic strategy does not always produce an optimal solution, but sometimes it does.

There are two key ingredients in greedy algorithm that will solve a particular optimization problem.

1. Greedy choice property
2. Optimal substructure

1. *Greedy choice property*:

A globally optimal solution can be arrived at by making a locally optimal (greedy) choice. In other words, when a choice is to be made, then it looks for best choice in the current problem, without considering results from the sub-problems. In

this algorithm choice is made that seems best at the moment and solve the sub-problems after the choice is made. The choices made by a greedy algorithm may depend on choices so far, but it can not depend on any future choice or solution to the sub-problems. The algorithm progress in a top down manner, making one greedy choice one after another, reducing each given problem instances into smaller one.

2. *Optimal substructure:*

A problem is said to have optimal substructure if an optimal solution can be constructed efficiently from optimal solution to its sub-problem. The optimal substructure varies across problem domain in two ways-

- **i)** How many sub-problems are used in an optimal solution to the original problem.
- **ii)** How many choices we have in determining which sub-problem to use in an optimal solution.

In Greedy algorithm a sub-problem is created by having made the greedy choice in the original problem. Here, an optimal solution to the sub-problem, combined with the greedy choice already made, yield an optimal solution to the original problem.

3.4 KNAPSACK PROBLEM

There are n items, i^{th} item is worth v_i dollars and weight w_i pounds, where v_i and w_i are integers. Select item to put in knapsack with total weight is less than W, So that the total value is maximized. This problem is called knapsack problem.
This problem finds, which items should choice from n item to obtain maximum profit and total weight is less than W.

The problem can be explained as follows-

A thief robbing a store finds n items, the i^{th} item is worth v_i dollar and weight w pounds, where v_i and w_i are integers. He wants to take as valuable load as possible, but he can carry atmost W pounds in his knapsack, where W is an integer. Which item should he take?

There are two types of knapsack problem.

1. ***0-1 knapsack problem****:*
   In **0**-1 knapsack problem each item either be taken or left behind.

2. ***Fractional knapsack problem***:
   In fractional knapsack problem fractions of items are allowed to choose.

3.5 GREEDY STRATEGY APPLIED IN 0-1 KNAPSACK PROBLEM

The greedy algorithm in 0-1 knapsack problem can be applied as follows-

1. ***Greedy choice***:
   Take an item with maximum value per pound.
2. ***Optimal substructure***:
   Consider the most valuable load that weights atmost W pounds. These W pounds can be choose from n item. If j^{th} item is choose first then remaining weight $W-w_i$ can be choose from n-1 remaining item excluding j.

3.6 GREEDY STRATEGY APPLIED IN FRACTIONAL KNAPSACK PROBLEM

1. ***Greedy choice:***
   Take an item or fraction of item with maximum value per pound.
2. ***Optimal substructure***:
   If we choose a fraction of weight w of the item j, then the remaining weight atmost W-w can be choose from the n-1 item plus w_i-w pounds of item j.

Although, both the problems are similar, the fractional knapsack problem is solvable by greedy strategy, but 0-1 knapsack problem are not solvable by greedy algorithm.

Consider the following problem-

There are 3 items. The knapsack can hold 50 pounds. Item1 weight 10 pounds and its worth is 60 dollar, item2 weight 20 pounds and its worth 100 dollars, item3 weight 30 pounds and its weight 120 dollars. Find out the items with maximum profit which the knapsack can carry.

Solution:

Here,

W = 50 pounds

Item	***Weight (w pound)***	***Worth (v dollar)***
Item1	10	60
Item2	20	100
Item3	30	120

Let, an item I has weight w_i pounds and worth v_i dollar.

Value per pound of I = v_i / w_i .

Thus, value per pound for-

Item1 = w_1 / v_1

= 60 dollars / 10 pounds

= 6 dollars/pounds

Item2 = w_2 / v_2

= 100 dollars / 20 pounds

= 5 dollars/pounds

Item3 = w_3 / v_3

= 120 dollars / 30 pounds

= 4 dollars/pounds

We can select maximum of 50 pounds.

So, using greedy strategy in 0-1 knapsack problem

1st choice is Item1.

2nd choice is Item2.

Total weight = 10 + 20 pounds

= 30 pounds

Total worth = 60 +100 dollars

=160 dollars

But this is not the optimal choice.

The optimal choice will choose item 2 and 3. Then,

Total weight = 20 + 30 pounds

= 50 pounds

Total worth = 100 + 120 dollars

= 220 dollars.

Hence , 0-1 knapsack problem is not solved by greedy strategy.

Now, using greedy strategy in fractional knapsack problem –

1st choice is item1.

2nd choice is item2

Total weight = 30 pounds

But the size of the knapsack is 50 pounds.

So, it will take remaining 20 pounds from item3 (fraction of item3) and its worth is 4 x 20=80 dollars.

Hence,

Total weights = 50 pounds.

Total worth = 60+100+ 80 dollars

=240 dollars.

Hence , an optimal solution can be obtain from fractional knapsack problem using greedy strategy.

CHECK YOUR PROGRESS

1. Write True or False

   a) Greedy choice always looks for the best choice in the current problem.

   b) 0-1 knapsack problem is solvable by greedy algorithm.

2. What is optimal substructure?

3. What is greedy strategy for knapsack problem?

3.9 MINIMUM SPANNING TREE

Before going to the definition of the minimum spanning tree let us define what a spanning tree is :

Spanning tree:

A spanning tree is a connected graph, say G = (V, E) with V as set of vertices and E as set of edges, is its connected acyclic sub-graph that contain all the vertices of the graph.

Now the minimum spanning tree can be defined as:

Minimum spanning tree:

A minimum spanning tree T of a positive weighted graph G is a minimum weighted spanning tree in which total weight of all edges are minimum

$\mathbf{w(T) = \sum_{(u,v) \in T} w(u, v)}$ is minimized.

Where w(u, v) is the cost of the edge (u, v).

For example;

Let us consider connected graph G given in fig

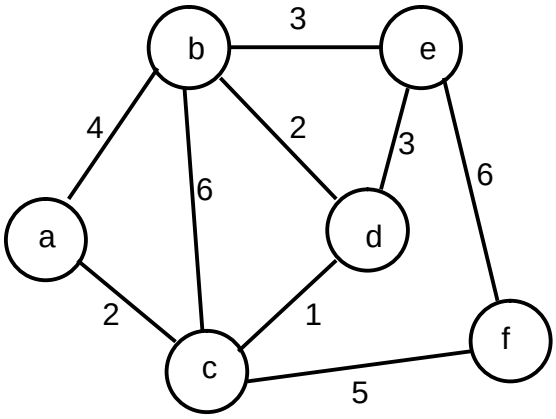


Fig. 3.2 A Connected graph G

Now, the minimum spanning trees are for the graph G is-

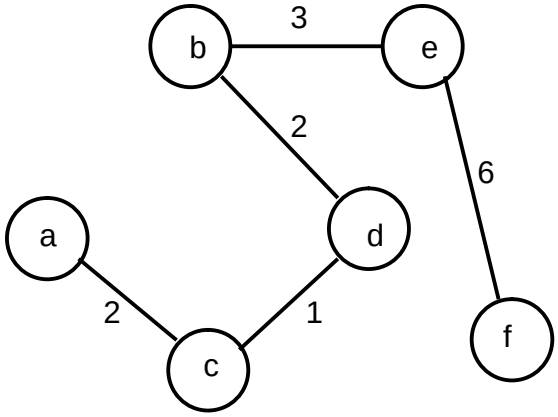


Fig. 3.3 A spanning tree for the graph G

Application of Minimum spanning tree:

i. In design of electric circuit network .
ii. It is used in traveling salesman problem.

The ***minimum spanning tree problem*** is the problem of finding a minimum spanning tree for a given weighted connected graph

There are two algorithms to solve minimum spanning tree problem

1. Kruskal algorithm
2. Prim algorithm

The general approaches of these algorithms are-

- The tree is built edge by edge.
- Let T be the set of edges selected so far.
- Each time a decision is made. Include an edge e to T s.t. Cost (T) + w (e) is minimized, and T U {e} does not create a cycle.

Both these algorithms are greedy algorithm. Because at each step of an algorithm, one of the best possible choices must be made. The greedy strategy advocates making the choice that is best at the moment. Such a strategy is not generally guaranteed to globally optimal solution to a problem.

3.10 PRIM'S ALGORITHM

The prim's algorithm uses greedy method to build the sub-tree edge by edge to obtain a minimum cost spanning tree. The edge to include is chosen according to some optimization criterion. Initially the tree is just a single vertex which is selected arbitrarily from the set V of vertices of a given graph G. Next edge is added to the tree by selecting the minimum weighted edge from the remaining edges and which does not form a cycle with the earlier selected edges. The tree is represented by a pair (V', E') where V' and E' represent set of vertices and set of edges of the sub-tree of minimum spanning tree.

The algorithm is as follows-
The algorithm continuously increases the size of a tree, one edge at a time, starting with a tree consisting of a single vertex, until it finds all vertices.

- *Input:* A non-empty connected weighted graph with vertices *V* and edges *E* (the weights are positive).

- *Initialize:* V' = {*x*}, where *x* is an arbitrary node (starting point) from V,

  E' = { }

- Repeat until V' = V;
- Choose an edge (*u*, *v*) with minimal weight such that *u* is in V' and *v* is not in V' (if there are multiple edges with the same weight, any of them may be picked)
- Add *v* to V' and (*u*, *v*) to E' if edge (u, v) will not make a cycle with the edges already in E'.
- Output: V' and E' describe a minimal spanning tree

Example:

Let us consider the following graph G.

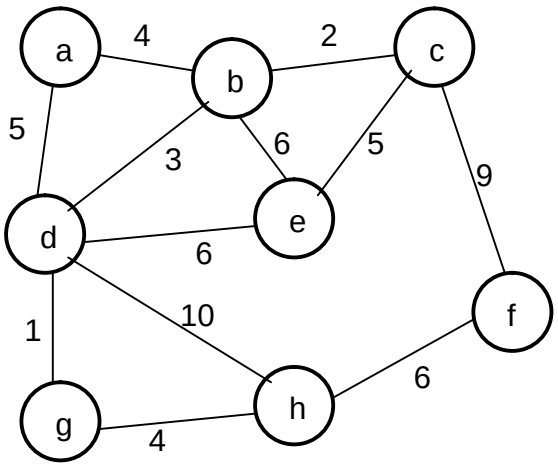


Fig. 3.4 Prim's algorithm applied on the Graph G

Initially vertex *a* is selected. So, V' will contain *a*.

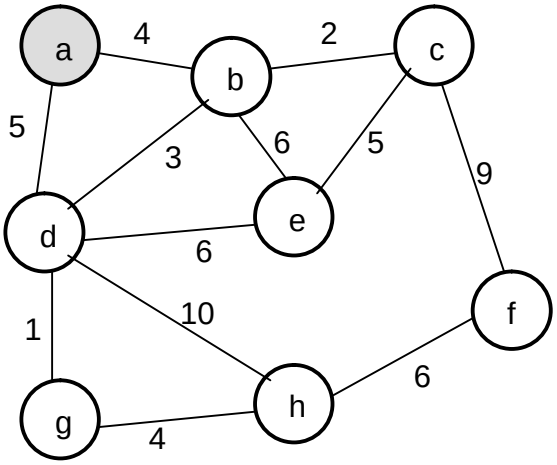


Fig. 3.5 Vertex a is selected

V' = {a}

E' = Ø

After first iteration, the minimum weight edge connected *a* and other vertices of V is selected. In this case from vertex *a* there are two edges *ab* and *ad* to vertex *b* and *d*.

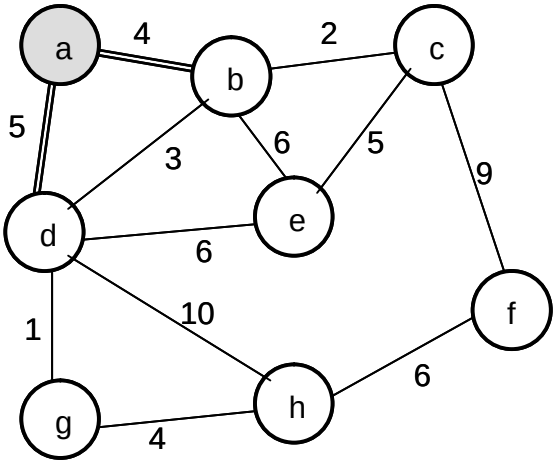


Fig. 3.6 Finds the minimum weighted edge

Between *ab* and *ad* weight of *ab* is minimum. Hence, after first iteration vertex *b* is include to V' and edge *ab* is included to E'.

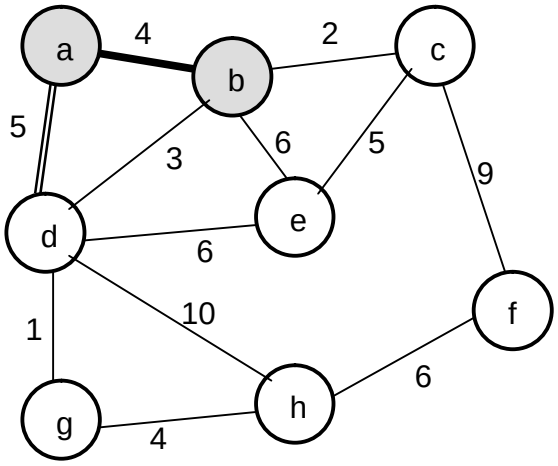


Fig. 3.7 Minimum weighted edge selected

V' = {a, b}

E' ={ ab }

In the next iteration we select the minimum weight edge, which does not make a cycle with previously selected edges in E', from the edges not included in E' and edges connected one vertex from V' and another vertex not in V'. Here edges from a and b to any other vertex. Here, edges are ad, bd, be, bc from which we can select the minimum weight edge.

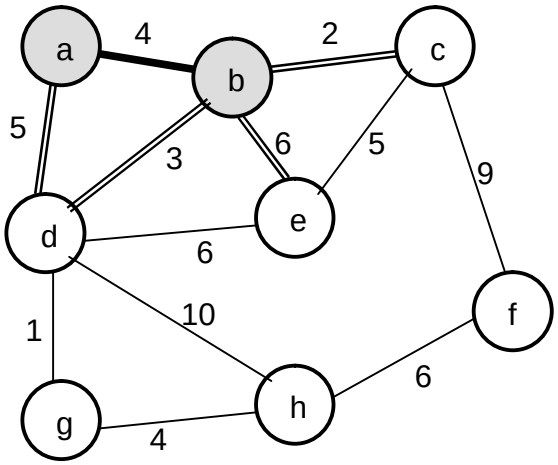


Fig. 3.8 Finds the minimum weighted edge

Here, weight of bc is minimum and it does not make a cycle with ab. Thus bc is selected in this iteration.

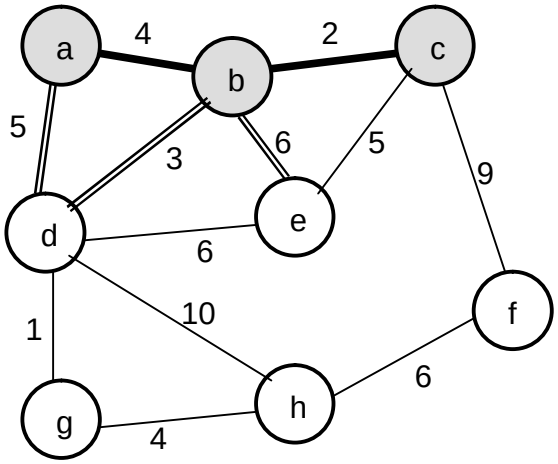


Fig. 3.9 Minimum weighted edge selected

V' = { a, b, c }

E' = { ab, bc }

In the next iteration we can consider the edges that have a,b or c as one of the vertex . Here the edges are ad, bd, be, ce, cf. we can not consider ab and bd because they are already selected.

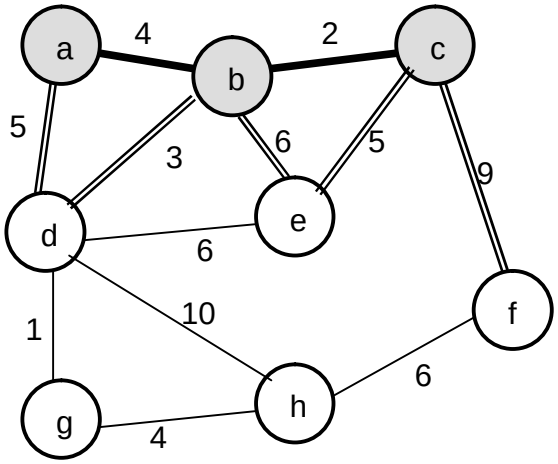


Fig. 3.10 Finds minimum weighted edge

From these edges weight of bd is minimum and it does not make a cycle with the edge in E'. Thus bd is selected.

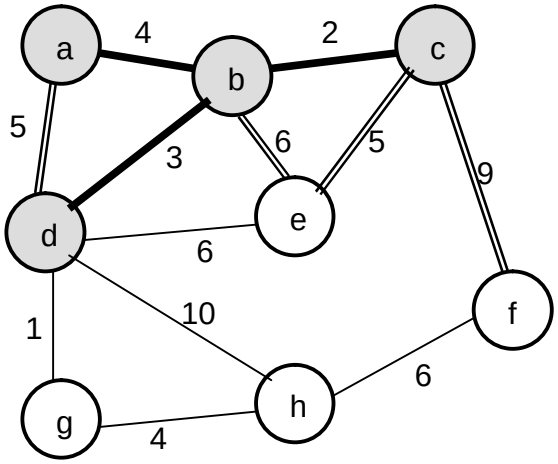


Fig. 3.11 Minimum weighted edge selected

V' = { a, b, c, d }

E' = { ab, bc, bd }

In the next iteration we consider the edges (excluding already selected edges) that have a, b, c, d as one vertex. Here edges are ad, be, ce, cf, de, dh, dg.

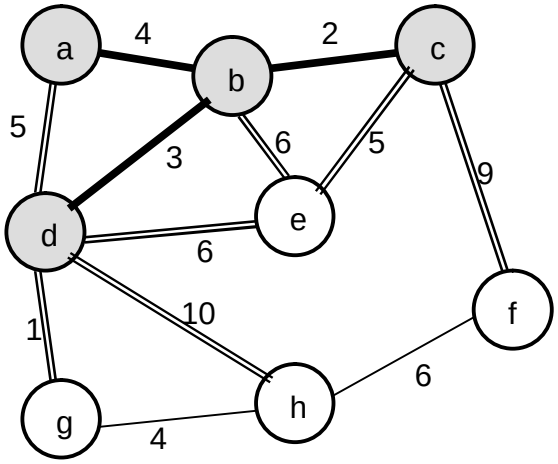


Fig. 3.12 Finds Minimum weighted edge

The weight of dg is minimum and it does not make a cycle with the edges in E'. Thus dg is selected.

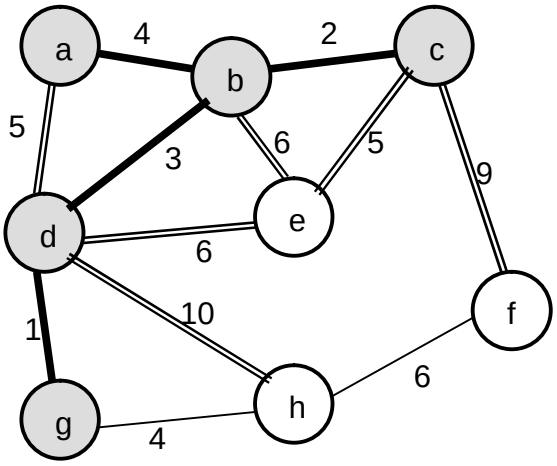


Fig. 3.13 Minimum weighted edge selected

V' = { a, b, c, d, g }

E' = { ab, bc, bd, dg }

In the next iteration considered edges are ad, be, de, gh, dh, ce, cf .

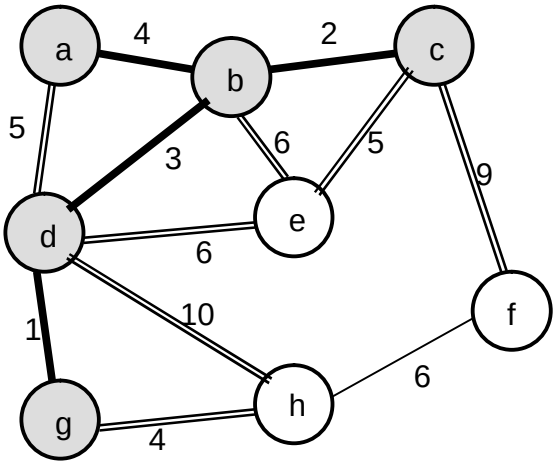


Fig. 3.14 Finds Minimum weighted edge

Among these edges weight of gh is minimum and it does not make any cycle with already selected edges in E'. Thus, gh is selected.

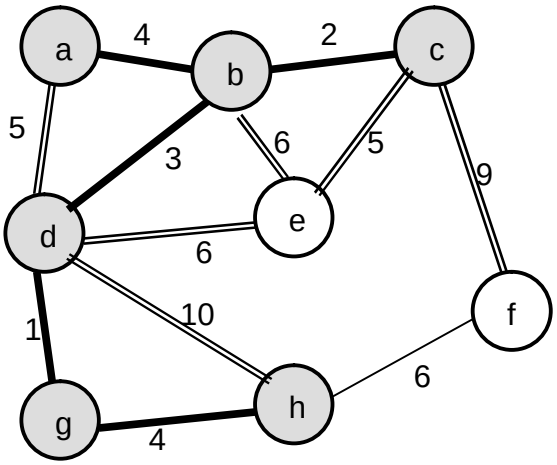


Fig. 3.15 Minimum weighted edge selected

V' = { a, b, c, d, g, h }

E' = { ab, bc, bd, dg, gh }

In the next iteration consider the edges that has one vertex from V' and connect another vertex excluding already selected edges. Here edges are ad, be, ce, cf, de, dh, hf.

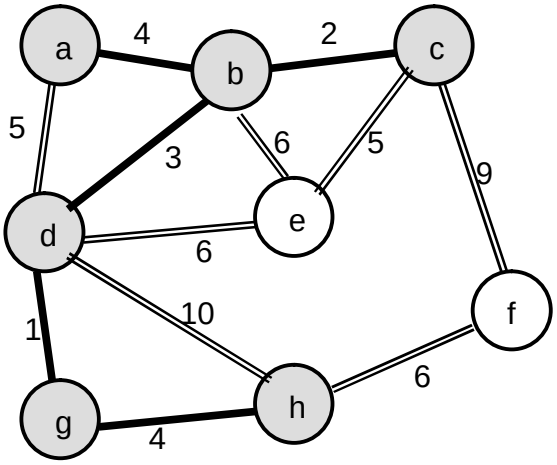


Fig. 3.16 Finds Minimum weighted edge

Among these weight ad and ce are minimum. If select ad then it make a cycle with the already selected edge ab and bd of E'. So,

ad can not be selected. If we select ce it will not make a cycle with the edges of E'.

Thus ce is selected.

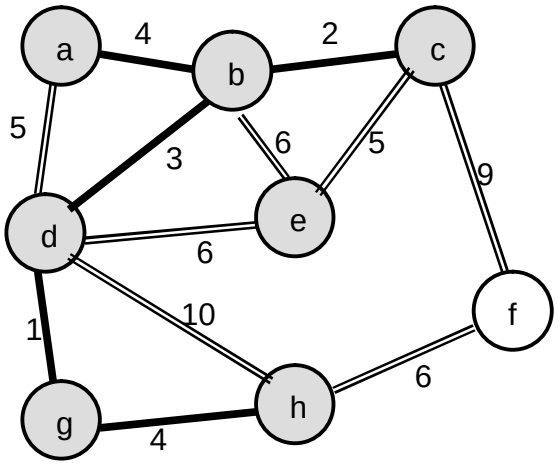


Fig. 3.17 Minimum weighted edge selected

V' = { a, b, c, d, e, g, h }

E' = { ab, bc, bd, dg, gh, ce }

In the next iteration considered edges are ad, be, de, dh, cf, hf.

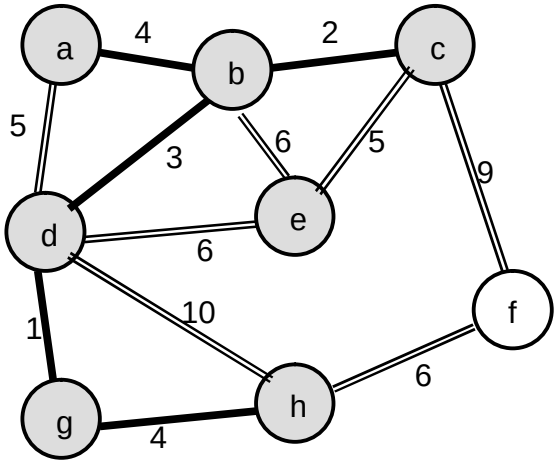


Fig. 3.18 Finds Minimum weighted edge

Among these weight of ad is minimum. But it makes a cycle with already selected edges ad and ab. So, ad is rejected. Next minimum weight is of edge de, be and hf. But this two edges will also make cycle. So de and be are also rejected. hf will not make a cycle. Thus hf is considered.

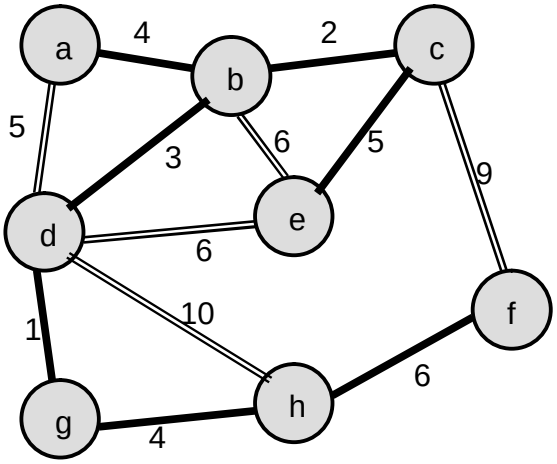


Fig. 3.19 Minimum weighted edge selected

V' = { a, b, c, d, e, f, g, h }

E' = { ab, bc, bd, dg, gh, ce, hf }

Next, the edges be, de, cf ,ad, dh can not included to form the tree because they make a cycle with already selected edges. Hence the final spanning tree is-

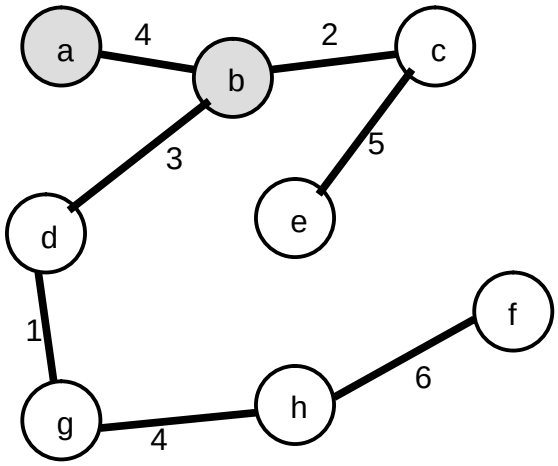


Fig. 3.20 Final spanning tree of graph G

3.11 KRUSKAL ALGORITHM

Another method of finding minimum spanning tree is Kruskal algorithm. In this algorithm the edges of the graph are considered in non decreasing order. The result is a forest of trees that grows until all the trees in a forest (all the components) merge in a single tree.

The algorithm is as follows-

- create a forest *F* (a set of trees), where each vertex in the graph is a separate tree
- create a set *S* containing all the edges in the graph
- while *S* is nonempty and F is not yet spanning
- remove an edge with minimum weight from *S*
- if that edge connects two different trees, then add it to the forest, combining two trees into a single tree
- otherwise discard that edge.

Example:

Let us consider the following graph-

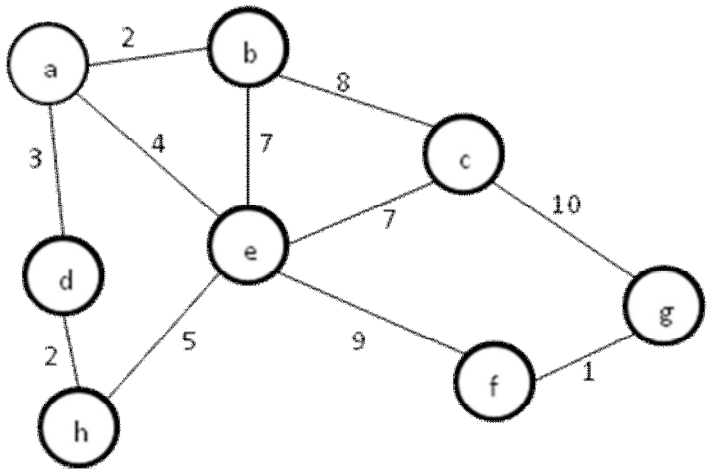


Fig. 3.21 Krushkal algorithm applied on graph G

Initially there are 8 vertices. So initially there are 8 trees in the forest F and S contains all the edges in the graph.

F = a b c d e f g h

S = ab, bc, ad, ae, be, ce, dh, cg, ef, fg, he

In the first iteration we consider the smallest weight edge from the set S. If the both vertex of the edge connect two different trees in the forest F then that edge is selected and combined the two trees into a single tree. Here, in first iteration weight of fg edge is 1, which is minimum. It connects two vertices f and g. In the forest F, f and g belongs to two separate tree. Thus, fg edge is selected and fg is removed from the set S and f and g trees are combined into a single tree fg in F.

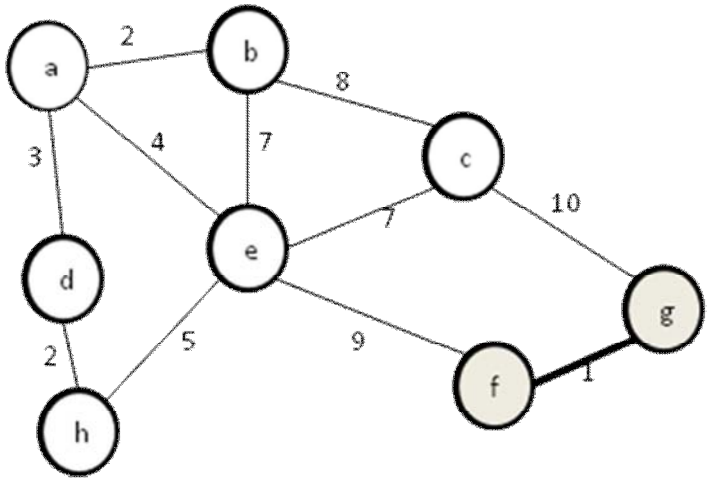


Fig. 3.22

F = a b c d gf e h

S = ab, bc, ad, ae, be, ce, dh, cg, ef, he

In the second iteration, after removing fg edge form S we consider the minimum weight edge from the new set S i.e from remaining edges we consider the minimum weight edge. Next the minimum weight edges are ab and dh. Each of which has weight 2. We can consider any one of the edge. Because for each edges their

connected vertices belongs to two different tree. Let us select the edge ab. Thus,

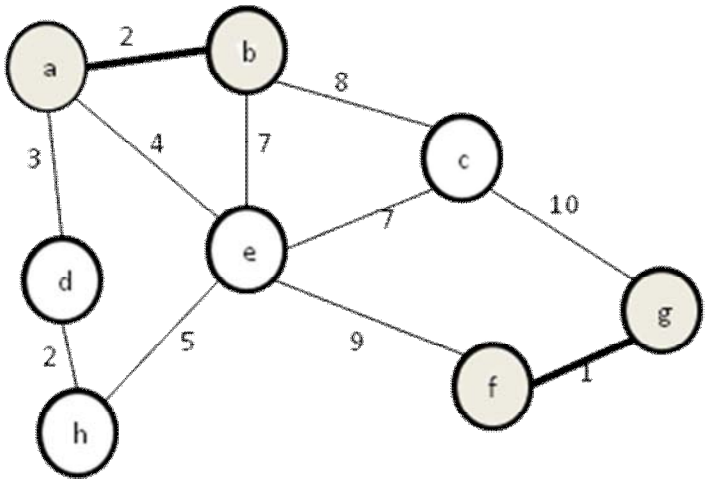


Fig. 3.23

F = ab c d gf e h

S = bc, ad, ae, be, ce, dh, cg, ef, he

In third iteration minimum weight edge in S is dh of weight 2. Now vertex d and h of edge dh belongs to two different tree. Hence, we can select edge dh. Thus, F and S becomes

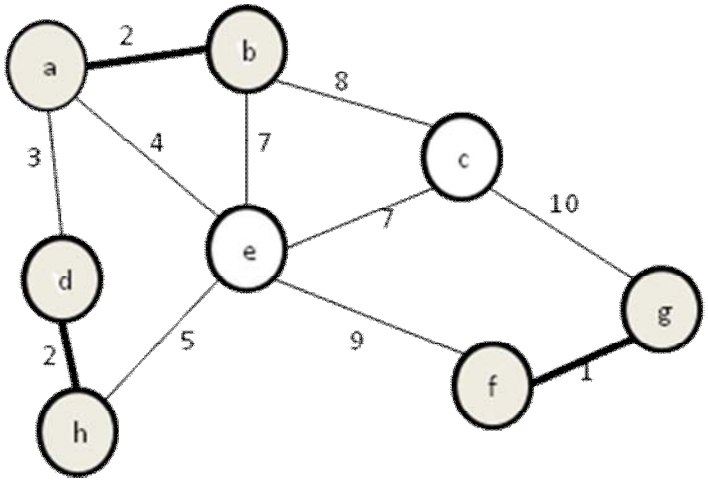


Fig. 3.24

F = ab c dh gf e

S = bc, ad, ae, be, ce, cg, ef, he

In fourth iteration from the remaining edges set S minimum edge is ad of weight 3.Now the vertices a and d belongs to two different tree ab and dh respectively in F. So. Edge ad is selected. Now

,

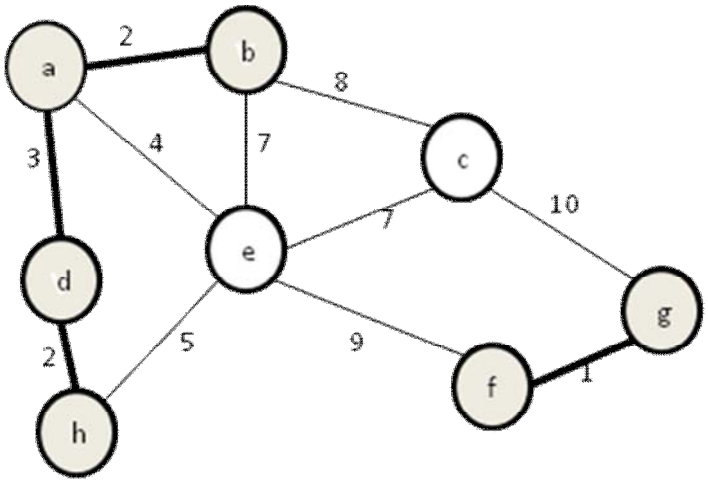


Fig. 3.25

F = abdh c gf e

S = bc, ae, be, ce, cg, ef, he

Now, in next iteration the minimum weight edge in S is ae. Here a and e belongs to two different tree abdh and e respectively in F. Hence, ae is selected and removed from S and two tree combined to a single tree abdhe. Thus,

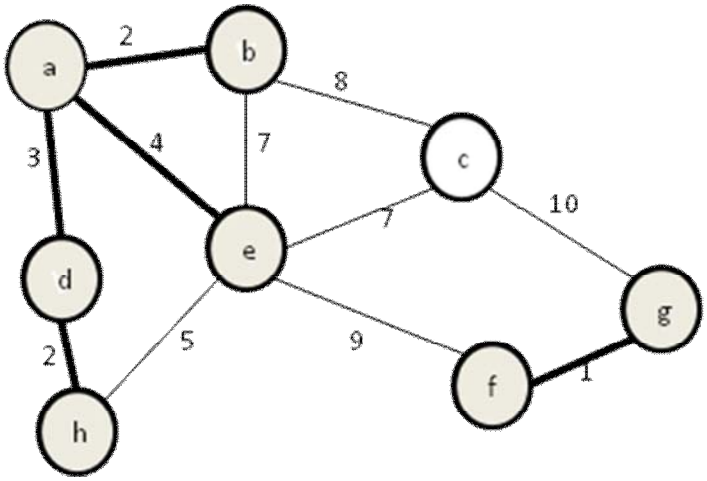


Fig. 3.26

F = abdhe c gf

S = bc, be, ce, cg, ef, he

In next iteration minimum weight edge in S is he of weight 5. Now, h and e belongs to same tree abdhe. So, edge he is not selected. Simply remove he from S. So, S becomes

S = bc, be, ce, cg, ef

Now, from S the minimum weight edges are be and ce of weight 7 in each. For edge be , b and e is from same tree abdhe in F. So, be can't selected. So, this be remove from S.

S = bc, ce, cg, ef

We can select ce because c and e belongs to two different tree in F. Hence

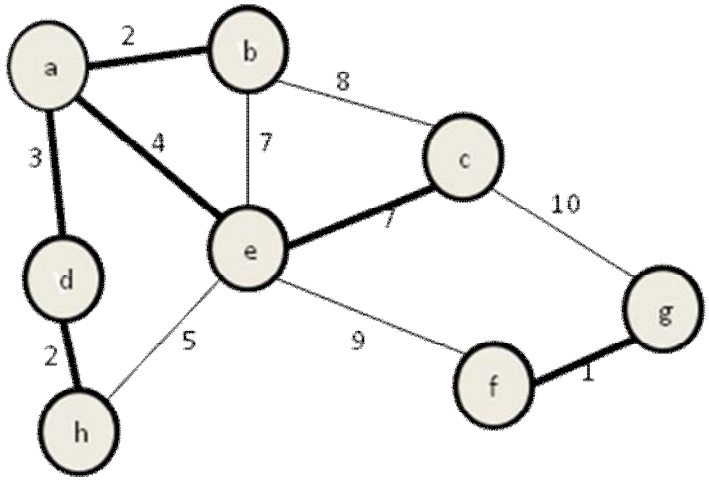


Fig. 3.27

F = abcdeh gf

S = bc, cg, ef

In next iteration minimum weight edge is bc of weight 8.We can't consider bc because b and c is in same tree in F. Thus

S = cg, ef

Next minimum weight edge in S is ef. Edge ef can selected because vertices e and f belongs to two different trees in F. Thus

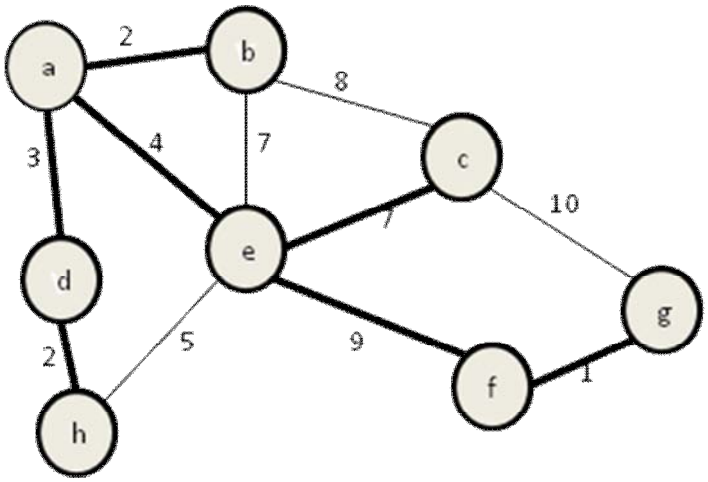


Fig. 3.28

F = abcdefgh

S = cg

In next iteration edge is cg. Edge cg is not selected because vertices c and g is in one tree in F. Hence the final tree is-

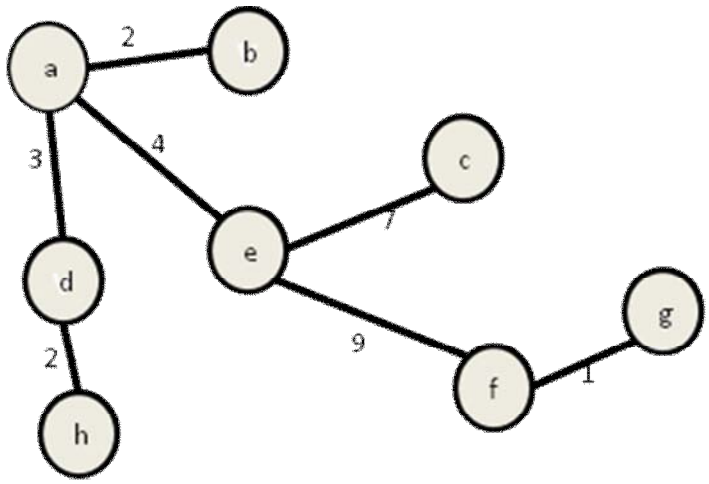


Fig. 3.29 Final tree of graph G

F = abcdefgh

S = nil

CHECK YOUR PROGRESS

7. What is minimum spanning tree?
8. What are the algorithms to solve minimum spanning tree problem ?

3.12 DIJKSTRA'S ALGORITHM

Shortest path problem:

For a given weighted and directed graph G= (V, E), the shortest path problem is the problem of finding a shortest path between any two vertex v ϵ V in graph G. The property of the shortest path is such that a shortest path between two vertices contains other

shortest path within it i.e any other sub-path of a shortest path is also a shortest path.

Single source shortest path problem:

In a single source shortest path problem , there is only one source vertex S in the vertex set V of graph G=(V, E). Now this single source shortest path problem finds out the shortest path from the source vertex S to any other vertex in v € V.

Optimal substructure of a shortest path:

Optimal substructure of a shortest path can be stated that any other sub-path of a shortest path is also a shortest path. Here is the lemma-

Lemma:

Given a weighted directed graph G=(V, E) with weight function w: E -> R, let p = (v_1,v_2 ----, v_k) be a shortest path from vertex v_1 to vertex v_k, and for any i and j such that $1 \le i \le j \le k$, *let P_{ij} = (v_1,v_{i+1}, ------,v_j) be the sub-path of P from vertex v_i to vertex v_j. Then P_{ij} is a shortest path from v_i to v_j.*

Dijsktra algorithm solves the single source shortest path problem. But the algorithm works only on a directed and positive weighted graph. Positive weighted graph means where weights of all edges are non negative i.e G=(V, E) is a positive weighted graph then w(u, v) ≥ 0. Dijsktra algorithm is a greedy algorithm.

Dijkstra algorithm is as follows-

For a given graph G=(V, E) and a source vertex s, it maintains a set F of vertices .Initially no vertex is in F. For a vertex u ϵ V - F, (i.e for a vertex which is in V, but is not in F) if it has minimum shortest distance from source s to u then u is added to F. This process is continue till V - F is not equal to null.

```
DIJKSTRA( G, w, s)
1.      INITIALIZE_SINGLE_SOURCE(G,s)
        1.1 for each vertex v € V[G]
        1.2 do d [v] = ∞
        1.3  Π [v] = NIL
        1.4  d [s] = 0
2.      F = Ø
3.      Q = V [G]
4.      while Q != Ø
5.          do u = EXTRACT_MIN(Q)
6.            F = F U {u}
7.            for each vertex v € Adj [u]
8.             do RELAX (u, v, w)
                  8.1 if d [v] > (d [u] + w(u, v))
                  8.2  then d [v] = d [u] + w(u, v)
                  8.3 Π [v] = u;
```

In line 1 (from line 1.1 to 1.4) initialize the value of d and π . Here d [v] means distance from source to vertex v and π [v] means parent of vertex v. Initially source to source distance is 0. So, d [s] = 0 . Also for all vertices v € V , d[v] is set as ∞ and π [v] as NIL.

In line 2 it initializes set F to empty set as initially no vertex is added to it.

In line 3 Q is a min-priority queue and initially it contains all vertices set V[G] of graph G.

In line 4 the while loop of line 4-8 will continue until the min-priority queue Q become empty.

In line 5 it extracts the minimum distance vertex u from source s i.e u € V - F for which d [u] is minimum.

In line 6 u is added to F.

In line 7-8 (from line 8.1 to 8.3) for all vertex v which is adjacent to u , calculate the distance to v through vertex u. If this

value is less than d[v] then update d [v] to this new value.

Make parent of v, π [v] = u.

3.12.1 Example:

Apply dijkstra algorithm for the following graph G.

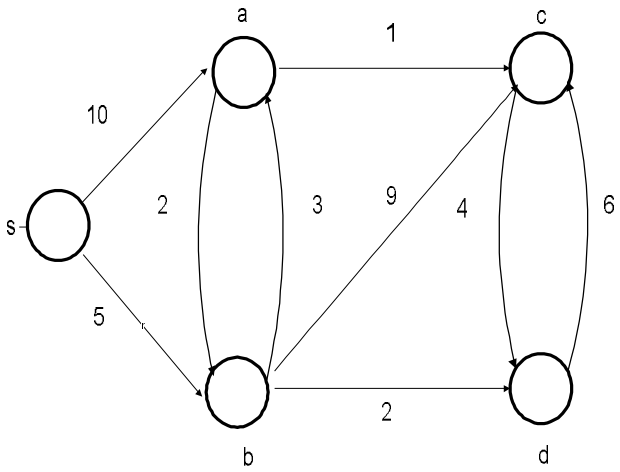


Fig. 3.30 Dijkstra algorithm applied on G

Initially

d[s]=0;

[s]=NIL;

And distance of all other vertices set as ∞.

d[a]= ∞, π [a]=NIL;

d[b]= ∞, π [b]=NIL;

d[c]= ∞, π [c]=NIL;

d[d]= ∞, π [d]=NIL;

s is added to F.

F={s}

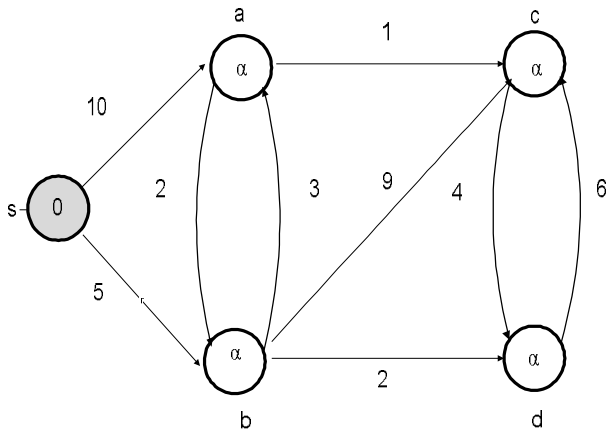


Fig. 3.31

In 2nd iteration

Adj[s]={a, b} and they are not in F.

Now d[a]=d[s]+w(s,a)

=0+10

=10

This new d[a] value is less than previous d[a] value i.e 10<∞

Hence d[a] is updated to 10

d[a]=10;

π [a]=s;

Similarly d[b]=d[s]+w(s,b)

=0+5

=5<previous d1[b]

=5<∞

d[b]=5;

π [b]=s;

and d[c]=∞, π [c]=NIL;

d[d]=∞, π [d]=NIL;

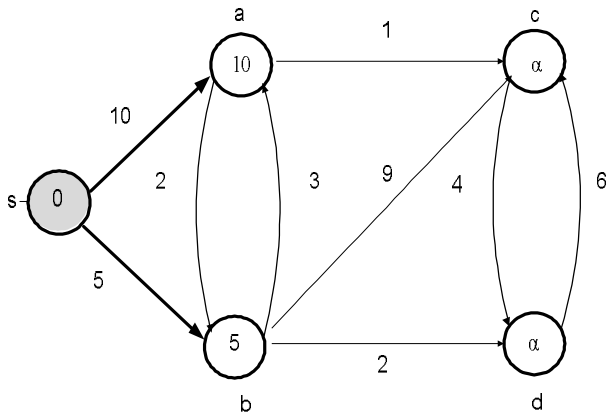


Fig. 3.32

Now, among these d[a],d[b],d[c],d[d] minimum value is d[b]i.e distance of vertex b is minimum from source.

Hence b is added to F

F={s,b}

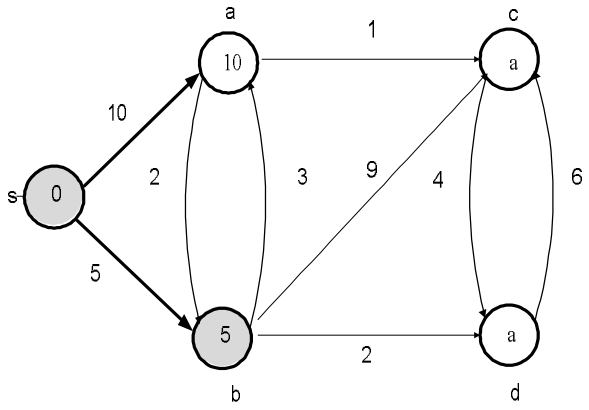


Fig. 3.33

In 3rd iteration,

Adj[b]={a,c,d} and they are not in F

For vertex a,

d[a]=d[b]+w(b,a)

=5+3

=8< previous d[a]

=8<10

Hence,

d[a]=8;

π [a]=b;

For vertex c

d[c]=d[b]+w(b,c)

=5+9

=14< previous d[c]

=14<∞

d[c]=14;

π [c]=b;

For vertex d

d[d]=d[b]+w(b,d)

=5+2

=7< previous d[d]

=7<∞

d[d]=7;

π [d]=b;

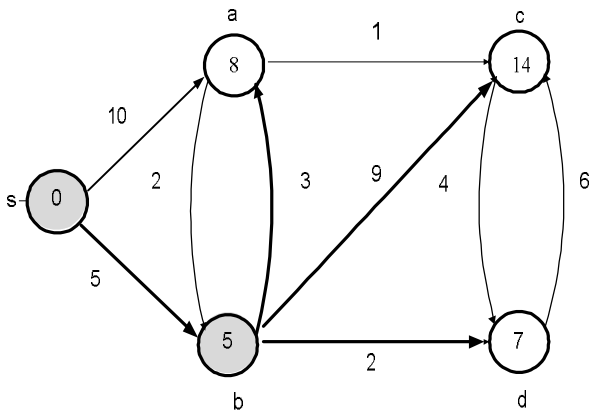


Fig. 3.34

Now among d[a],d[c],d[d] the minimum d value is d[d]

So, vertex d is added to F.

F={s,b,d}

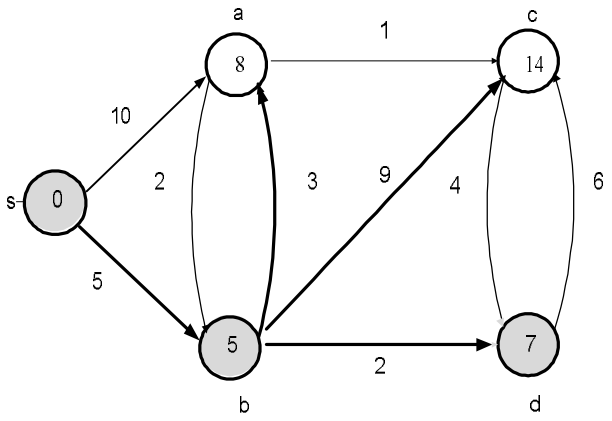


Fig. 3.35

In 4th iteration

Adj[d]={c} and c is not in F

Now d[c]=d[d]+w(d,c)

= 7+6

=13< previous d[c]

=13<14

d[c]=13;

π [c]=d;

And d[a]=8;

π [a]=s;

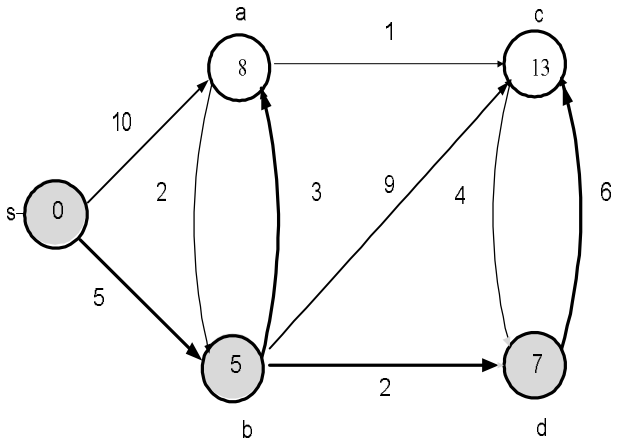


Fig. 3.36

Now the minimum of d[a] and d[c] is d[a].

So, vertex a is added to F

F={s,b,d,a}

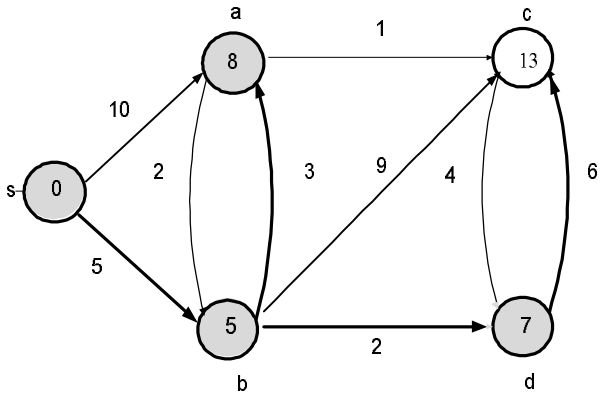


Fig. 3.37

In 5th iteration last added vertex is a.

Adj[a]={b,c},

Here c is not in F. But b is in F i.e b is already selected.

So, we will consider vertex c only.

d[c]=d[a]+w(a,c)

=8+1

=9< previous d[c]

So, d[c]=9

π [c]=a

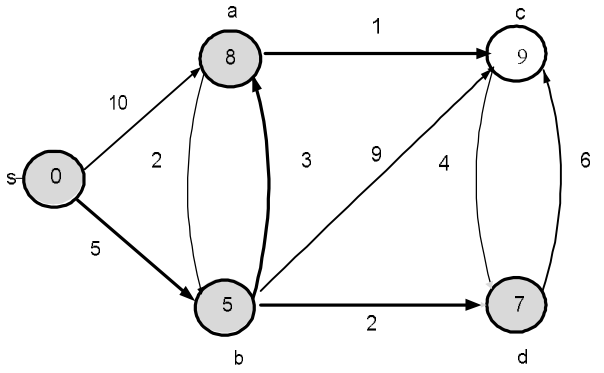


Fig. 3.38

Hence c is added to F

F={s,b,d,a,c}

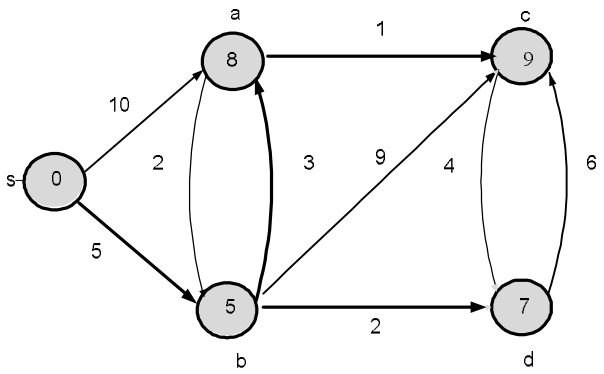
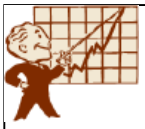


Fig. 3.39

There is no vertex to added in F.So, the algorithm terminate here.



CHECK YOUR PROGRESS

9. What is single source shortest path problem?
10. True/False
    i. In Dijkstra algorithm there are two source vertices.
    ii. Dijkstra algorithm can solve shortest path problem.

3.13 LET US SUM UP

- Greedy algorithm is typically used in optimization problem.
- Optimal solution finds a given objective function which value is either maximizes or minimizes.
- A greedy algorithm always makes the choice that looks best at the moment. That is it makes a locally optimal choice that may be lead to a globally optimal solution.

- In Greedy algorithm choice is made that seems best at the moment and solve the sub-problems after the choice is made.
- Greedy algorithm progress in a top down manner,
- A problem is said to have optimal substructure if an optimal solution can be constructed efficiently from optimal solution to its sub-problem.
- Knapsack problem: There are n items, i^{th} item is worth v_i dollars and weight w_i pounds, where v_i and w_i are integers. Select item to put in knapsack with total weight is less than W, So that the total value is maximized
- There are two types of knapsack problem.
    i. 0-1 knapsack problem
    ii. fractional knapsack problem:
- In 0-1 knapsack problem each item either be taken or left behind.
- In fractional knapsack problem fractions of items are allowed to choose.
- the fractional knapsack problem is solvable by greedy strategy, but 0-1 knapsack problem are not solvable by greedy algorithm.
- In the job sequencing with deadline problem, a feasible solution is a subset of job J such that each job is completed by its deadline and optimal solution is a feasible solution with a maximum profit.
- the optimal way to pair wise merge n sorted files
- A spanning tree is a connected graph , say G = (V, E) with V as set of vertices and E as set of edges, is its connected acyclic sub-graph that contain all the vertices of the graph.
- A minimum spanning tree T of a positive weighted graph G is a minimum weighted spanning tree in which total weight of all edges are minimum

- Two algorithm to solve minimum spanning tree problem are- Kruskal algorithm and Prim algorithm
- For a given weighted and directed graph G= (V, E), the shortest path problem is the problem of finding a shortest path between any two vertex v € V in graph G.
- In a single source shortest path problem , there is only one source vertex S in the vertex set V of graph G=(V, E).

3.14 ANSWERS TO CHECK YOUR PROGRESS

CHECK YOUR PROGRESS – 1

1. a) True, b) True

2. A problem is said to have optimal substructure if an optimal solution can be constructed efficiently from optimal solution to its sub problem.

3. i)Greedy choice ii) Optimal substructure

4. i. True ii. False

5. According to greedy choice property at each step it looks for its best solution in the current set. In optimal merge pattern sorts the list of file and at each step merge the two smallest size files(best choice at that moment) together from the current file sets.

6. i. True ii. False

7. A minimum spanning tree T of a positive weighted graph G is a minimum weighted spanning tree in which total weight of all edges are minimum

$$w(T) = \sum_{(u,v) \in T} w(u, v)$$ is minimized.

Where w(u, v) is the cost of the edge (u, v).

8. Prim's algorithm and Kruskal algorithm

9. Single source shortest path problem of graph G=(V, E) is to finds out the shortest path from the only source vertex S to any other vertex in v € V. Here,only one source vertex S in the vertex set V

10. i. False ii. True

3.15 FURTHER READINGS

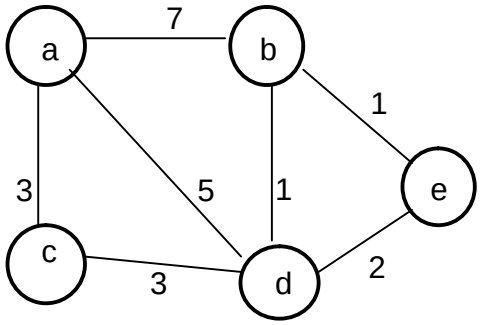
- T. H. Cormen, C. E. Leiserson, R.L.Rivest, and C. Stein, "Introduction to Algorithms", Second Edition, Prentice Hall of India Pvt. Ltd, 2006.
- Ellis Horowitz, Sartaj Sahni and Sanguthevar Rajasekaran, Computer Algorithms/ C++, Second Edition, Universities Press, 2007.

3.16 MODEL QUESTIONS

1. What is optimal substructure?
2. What is greedy strategy?
3. Write briefly about knapsack problem. Explain with an example that greedy algorithm does not work for 0-1 knapsack problem.
4. What is optimal substructure for 0-1 knapsack and fractional knapsack problem?
5. Consider the following job sequencing problem. Find the feasible solution set.

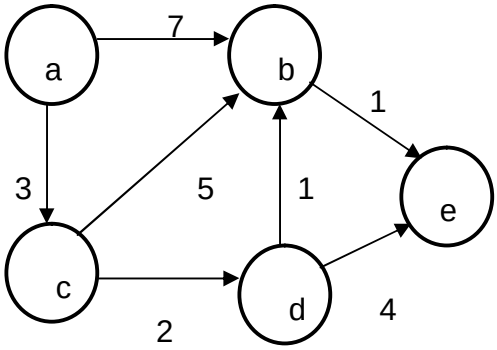
Job	1	2	3	4
Profit	10	20	15	5
Deadline	2	3	3	2

6. What is minimum spanning tree? Find the minimum spanning tree for the following graph using Prim's and Kruskal algorithm.



7. Find out the shortest path using Dijkstra algorithm for the following graph

source



8. "A globally optimal solution can be arrived at by making a locally optimal choice ". Explain briefly.