Module-3 Greedy Method

→ Greedy Method is a straightforward design technique can be applied to solve variety of problems.

→ Most of these problems have n inputs & require us to obtain a subset that satisfies some constraints.

→ Any subset that satisfies these constraints is called feasible solution.

→ We need to find a feasible solution that either maximizes or minimizes a given objective function.

→ A feasible solution that does this is called an optimal solution.

→ A greedy method suggest that one can devise an algorithm that works in stages, considering one input at a time.

→ At each stage, a decision is made regarding whether a particular input is in an optimal solution.

→ A general Greedy Algorithm -

```
Algorithm Greedy(a,n)
// a[1:n] contains the n inputs.
{
    solution = 0; // Initialize the solution
    for i=1 to n do
    {
        x = Select(a); // function select selects an I/p from a[]
        If feasible(solution x) Then
            solution = Union(solution, x);
    }
    return solution;
}
```

⊛ Coin change Problem Using Greedy Method -

Problem statement - Give change for a specific amount n with minimum/least number of coins of the denominations $d_1 > d_2 > d_3 \dots > d_m$ used in local.

→ Change-making problem faced by million bay of cashiers.

Example:

① Make change for given amount n = 49 with denominations availabel as - 1, 2, 5, 10, 20.
(must arrange denominations in ~~Ascending~~ Descending order to solve/get optimal solution).

Solution 1 - 10+10+10+10+5+2+2 ⇒ 49 & need 7 coins
Solution 2 - 20+10+10+5+2+2 ⇒ 49 & need 6 coins
Solution 3 - 20+20+5+2+2 ⇒ 49 & need 5 coins

Similarly, many solutions are possible for above given example -

But, optimal solution is solution 3 - Because, it just takes 5 coins to make a change of Rs. 49.

② Solve for given below exaple -
Make change for given amount n = 70 with availabel denominations as - 5, 10, 20, 50, 100.
Find minimum number of coins required to make change of Rs. 70.

* KNAPSACK Problem using Greedy Method -

Problem Statement : We are given n dinisible objects and a knapsacks or bag. object i has a weight w_i and the knapsacks has a maximum capacity m. If a fraction x_i, $0 \leq x_i \leq 1$, of object i is placed into the knapsacks, then a profit of $p_i x_i$ is earned. The objective is to obtain a completely filling of the knapsacks that maximizes the total profit earned. Since the knapsacks capacity is m, we require the total weight of all choosen objects to be at most m. Formally, the problem can be stated as -

$$\text{maximum weight} = \sum_{1 \leq i \leq n} w_i x_i \leq m$$

$$\text{maximum profit} = \sum_{1 \leq i \leq n} p_i x_i$$

And $0 \leq x_i \leq 1 \quad \forall 1 \leq i \leq n$

Example 1:

Consider the following instance of the knapsack problem :
Given number of objects, as n = 3, knapsacks/bag capacity, as m = 20, profit & weight of each object, $(P_1, P_2, P_3) = (25, 24, 15)$ and $(w_1, w_2, w_3) = (18, 15, 10)$ Respectively.
find the maximum profit for the above given problem instance.

Solution:

First calculate the ration Profit/weight; (P_i/w_i) for each given object & arrange the P_i/w_i ratio such that

$P_i/w_i \geq P_i/w_i \geq P_i/w_i$ ----

Object	1	2	3
Profit	25	24	15
Weight	18	15	10
P_i/w_i	1.3	1.6	1.5

$\Rightarrow$ Rearrage All objects in descending order of P_i/w_i

18

Bag capacity = m = 20

Objects	Profit P_i	Weight W_i	P_i/W_i	Fraction of Object chosen (x_i)	Remaining Bag Capacity	Maximum Profit $\sum P_i x_i$
2	24	15	1.6	1	20-15 = 5	24×1 = 24
3	15	10	1.5	5/10	5-5 = 0	15×5/10 = 7.5
1	25	18	1.3	0	0	25×0 = 0

Maximum Profit = 24 + 7.5 + 0 = 31.5 //

Example-2: find the optimal solution to the knapsack instant n=7, m=15 using greedy method —

Object	1	2	3	4	5	6	7
Profit	10	05	15	07	06	18	03
Weight	02	03	05	07	01	04	01
P_i/W_i	5	1.6	3	1	6	4.5	3

Now, Reorder objects in descending order of P_i/W_i with their corresponding profit and weight to get maximum profit — & Given Maximum knapsack capacity = 15.

Objects	Profit (P_i)	Weight (W_i)	P_i/W_i in DS order	Fraction of object chosen (x_i)	Remaining knapsack Capacity (m)	Maximum Profit $\sum P_i x_i$
5	06	01	6	1	15-01 = 14	6×1 = 06
1	10	02	5	1	14-02 = 12	10×1 = 10
6	18	04	4.5	1	12-04 = 08	18×1 = 18
3	15	05	3	1	08-05 = 03	15×1 = 15
7	03	01	3	1	03-01 = 02	03×1 = 03
2	05	03	1.6	2/3	02-02 = 0	5×2/3 = 03.33
4	07	07	1	0	0	00

Maximum Profit = 06 + 10 + 18 + 15 + 03 + 3.33 = 55.33 //

JOB SEQUENCING WITH DEADLINES - By Greedy Method -

Problem Statement: We are given a set of n jobs where each job i associated with an integer deadline $d_i \geq 0$ and a profit $P_i > 0$. For any job i the profit P_i is earned iff the job is completed by it's deadline. To complete a job, one has to process the job on a machine for one unit of time. Only one machine is availabel for processing jobs. The value of feasible solution J is the sum of the profits of the jobs in J, or $\sum_{i \in J} P_i$. An optimal solution is a feasible solution with maximum profit value.

Example 1: Given n = 4, jobs with their corresponding profits & deadlines as $(P_1, P_2, P_3, P_4) = (100, 10, 15, 27)$ & $(d_1, d_2, d_3, d_4) = (2, 1, 2, 1)$. find the maximum profit value.

Solution:

Jobs	1	2	3	4
Profits	100	10	15	27
Deadlines	2	1	2	1

Reorder Above Jobs Based on descending order of profit earned - Maximum deadline given in above example is 2. Hence, Maximum two jobs are Completed -

Jobs	Profits	Deadlines	JOB Scheduling	Profit Earned
1	100	2	[| J1]	100
4	27	1	[J4 | J1]	27
3	15	2	[J4 | J1]	JOB-3 Rejected
2	10	1	[J4 | J1]	JOB-2 Rejected-

Jobs Completed - Job1, Job4 Jobs Rejected - J_2, J_3

Total Profit = 100 + 27 = 127 ||

Exaple-2 - JOB Sequencing With deadlines - Solve for given exaple -

Jobs	1	2	3	4	5	6	7
Profits	3	5	20	18	1	6	30
Deadlines	1	3	4	3	2	1	2

Solution : From Above Exaple, Maximum deadline = 4, Hence, maximum four jobs will be completed.
Reorder All the jobs in descending order of profits.

Jobs	Profits	Deadlines	JOB SCHEDULING.	Profit Earned
J_7	30	2	[ ] [J_7] [ ] [ ]	30
J_3	20	4	[ ] [J_7] [ ] [J_3]	20
J_4	18	3	[ ] [J_7] [J_4] [J_3]	18
J_6	6	1	[J_6] [J_7] [J_4] [J_3]	06
J_2	5	3	[J_6] [J_7] [J_4] [J_3]	J_2 Rejected
J_1	3	1	[J_6] [J_7] [J_4] [J_3]	J_1 Rejected
J_5	1	2	[J_6] [J_7] [J_4] [J_3]	J_5 Rejected

Completed Jobs - $J_3 \rightarrow J_4 \rightarrow J_6 \rightarrow J_7$

Rejected Jobs - $J_1 \rightarrow J_2 \rightarrow J_5$

Maximum Profit = 30 + 20 + 18 + 06

= 74 //

Example-3 - Solve below exaple of Job Sequencing With Deadlines - Using Greedy Method -

Jobs	1	2	3	4	5
Profits	20	15	10	5	1
Deadlines	2	2	1	3	3

Solution: Maximum deadline given is 3, Hence, maximum 3 jobs will be completed out of 5 jobs.

Now, Reorder All the jobs in descending order of profits in a two table shown below -

Jobs	Profits	Deadlines	JOB SCHEDULING			Profit Earned
J1	20	2		J_1		20
J2	15	2	J_2	J_1		15
J3	10	1	J_2	J_1		J_3 Rejected
J4	05	3	J_2	J_1	J_4	05
J5	01	3	J_2	J_1	J_4	J_5 Rejected

Job Completed - J_1, J_2, J_4

Job Rejected - J_3, J_5

Maximum Profit = 20 + 15 + 05

= 40 //.

Spanning Tree :

Defn: A spanning tree of a connected graph is its connected acyclic graph/subgraph that contains all the vertices of the graph.

Minimum Cost Spanning Tree :

Defn: A minimum cost spanning tree of a weighted connected graph is its spanning tree of the smallest weight, where the weight of the tree is defined as the sum of the weights on all its edges.

→ The minimum spanning tree problem of finding a minimum spanning tree for a given weighted connected graph.

→ Below figure represents simple example of weighted connected graph & it's possible spanning trees —

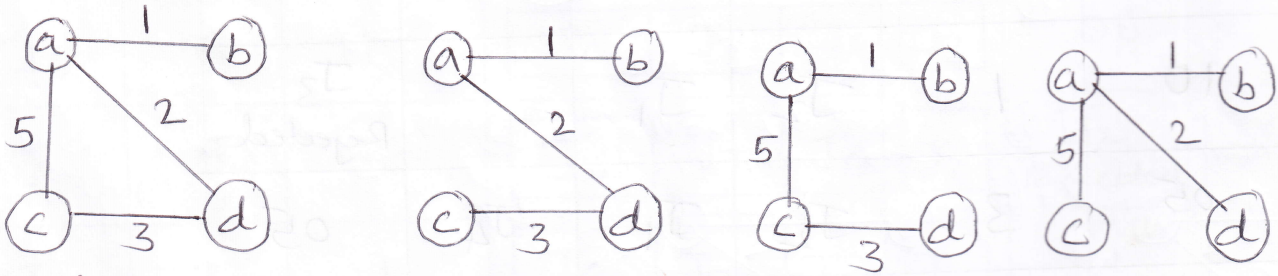


Figure - Graph & It's spanning Trees. T_1 is the minimum cost Spanning Tree.

→ Two Algorithms are used to Generate Minimum cost spanning tree -

1. Prim's Algorithm
2. Kruskal's Algorithm

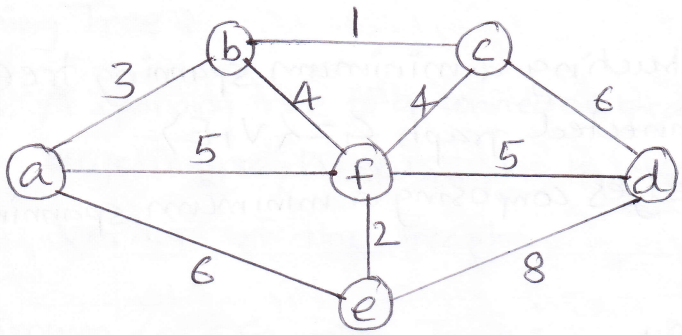
Prim's Algorithm

ALGORITHM Prim(G)
// Prim's Algorithm for constructing a minimum spanning tree
// Input: A weighted connected graph $G = \langle V, E \rangle$
// Output: E_T, the set of edges composing a minimum spanning tree of G.

$V_T \leftarrow \{v_0\}$ // the set of tree vertices can be initilized with any vertex
$E_T \leftarrow \emptyset$
for $i \leftarrow 1$ to $|V| - 1$ do
    Find a minimum-weight edge $e^* = (v^*, u^*)$ among all edges (v, u) such that v is in V_T and u is in $V - V_T$
    $V_T \leftarrow V_T \cup \{u^*\}$ // U - union
    $E_T \leftarrow E_T \cup \{e^*\}$ // U - union
return E_T.

→ Prim's Algorithm constructs a minimum cost spanning tree through a sequence of expanding subtrees.
→ The initial subtree in such a sequence consists of a single vertex selected arbitrarily from the set V of the graph's vertices.
→ On each iteration, we expand the current tree in the greedy manner by simply attaching to it the nearest vertex not in the subtree.
→ The algorithm stops after all the graph's vertices have been included in the tree being constructed.
→ Since the algorithm expands a tree by exactly one vertex on each of it's iterations, the total number of such iterations is n-1, where n is the number of vertices in the graph.
→ The tree generated by the algorithm is obtained as the set of edges used for the tree expansions.
→ Above is the pseudocode for Prim's Algorithm

Example 1: Apply Prim's Algorithm for below Given Graph & find cost of minimum spanning tree.



Solution: Step1: choose any one vertex randomaly from set of Vertices.

Step2: choose next vertex from remaining vertices that is connected by minimum edge to the already selected vertex.

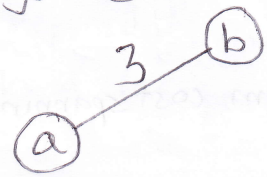
Step3: Carefully choose the edges to avoid cycle in resultant Spanning tree.

Step4: finally, check spanning tree includes all the vertices or no

Initialize with vertex a -

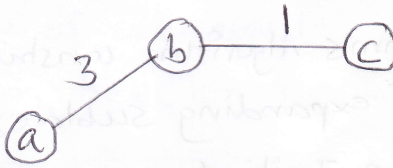
Step-1

(b) is the next vertex connected by minimum edge to (a)



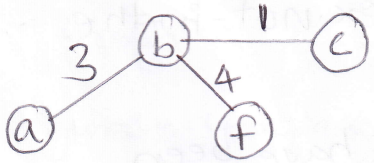
Step-2

(c) is the next vertex connected by minimum edge to (b)



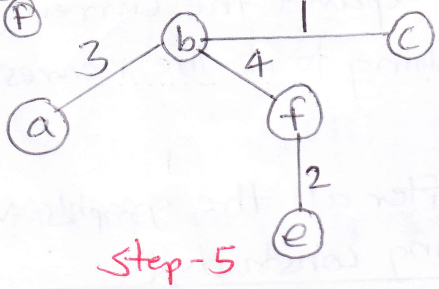
Step-3

(f) is the next vertex connected by minimum edge to (b)



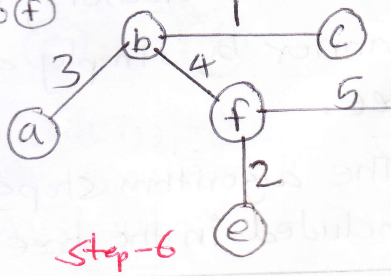
Step-4

(e) is the next vertex connected by minimum edge to (f)



Step-5

(d) is the next vertex connected by minimum edge to (f)



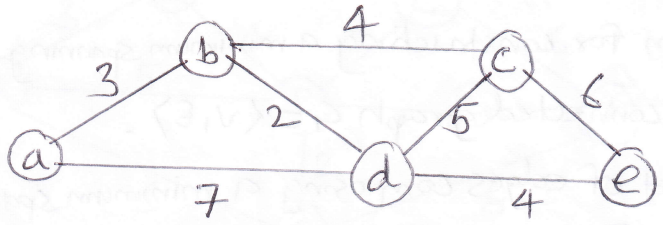
Step-6

Cost of spanning tree = sum of weights on all the edges of spanning

$= 3+1+4+5+2$

$= 15$.

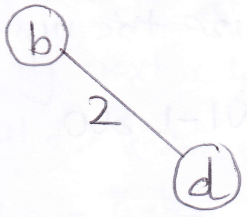
Example 2: Apply Prim's Algorithm for below given graph & find cost of minimum spanning tree -



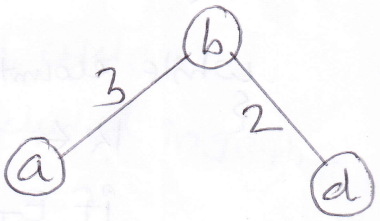
Solution

Initialized with vertex (b)

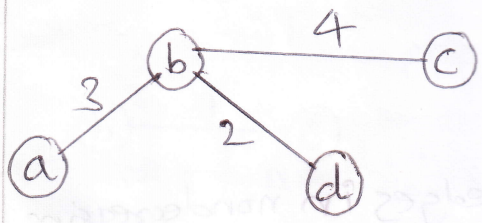
Step-1



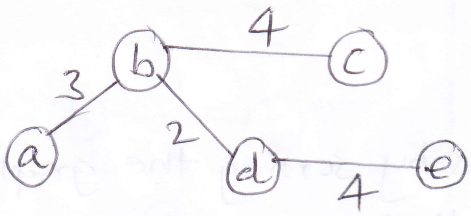
Step-2



Step-3



Step-4



Step-5

Minimum cost = W(T) = 3+4+2+4

= 13.

Kruskal's Algorithm

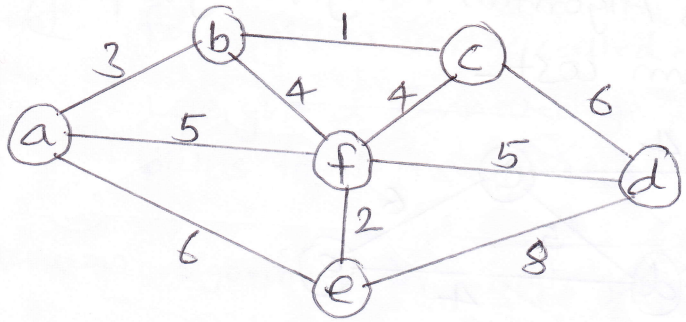
```
ALGORITHM Kruskal(G)
// Kruskal's algorithm for constructing a minimum spanning tree
// Input: A weighted connected graph G = <V, E>.
// Output: E_T, the set of edges composing a minimum spanning tree
           of G, sort E in ascending order of the edges
           weights w(e_i1) ≤ w(e_i2) --- ≤ w(e_i|E|)
E_T ← ∅; // Initilize the set of tree edges & its size
ecounter ← 0
k ← 0    // Initilize the number of processed edges
while ecounter < |V|-1 do
{
    k ← k+1
    if E_T ∪ {e_ik} is acyclic
    {
        E_T ← E_T ∪ {e_ik};
        ecounter ← ecounter+1
    }
}
return E_T.
```

→ The algorithm begins by sorting the graph's edges in nondecreasi ascending order of their weights.

→ Then starting with the empty subgraph, it scans this sorte list adding the next edge on the list to the current subgra if such an inclusion does not create a cycle and simpl skipping the edge otherwise.

→ Above is the pseudocode for Kruskal's algorithm.

Example 1: Apply Kruskal's Algorithm for given graph & find the minimum cost of spanning tree.



Solution: Step 1: Sort all the edges in ascending order of their weights.

Step 2: Start with minimum edge, and choose the next edge & repeat for all edges.

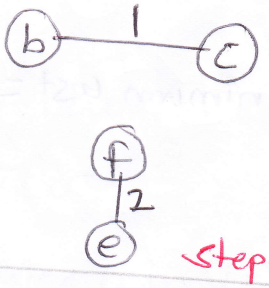
Step 3: While choosing next edge avoid cycle.

Step 4: Stop when all the vertices are included in graph.

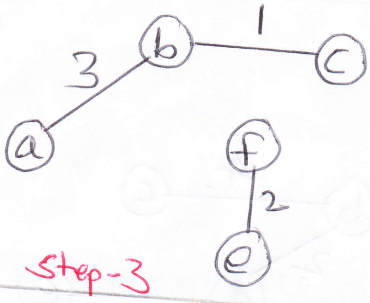
→ Sorted list of edges for above given Graph –

bc	ef	ab	bf	cf	af	df	ae	cd	de
1	2	3	4	4	5	5	6	6	8

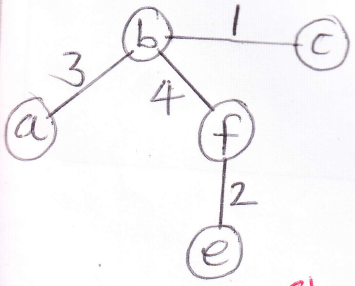
Step-1



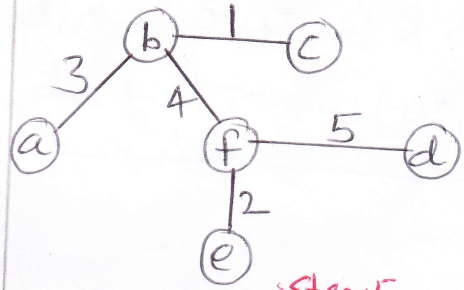
Step-2



Step-3



Step-4



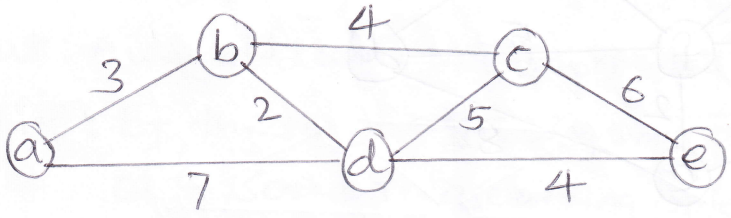
Step-5

Note: Don't include the edges cf, af. Because those edges create cycle in a graph.

Minimum Cost = W(T) = 3+1+4+5+2

= 15.

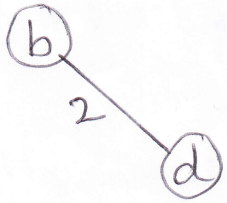
Example2: Apply Kruskal's Algorithm for given graph & find the minimum cost.



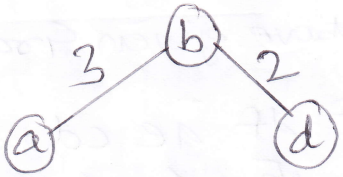
Solution:

Generate sorted list of edges for given Graph.

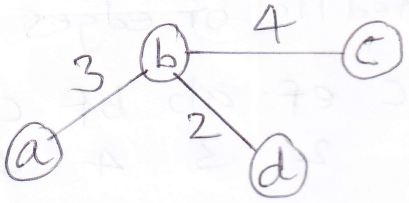
bd	ab	bc	de	dd	ce	ad
2	3	4	4	5	6	7



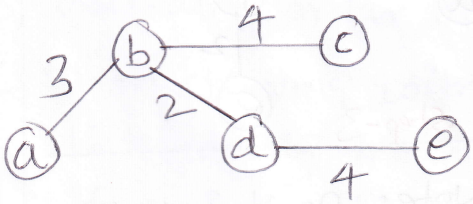
Step-1



Step-2



Step-3



Step-4

Minimum cost = $\sum(T) = 3+4+2+4$

$= 13$.

Single - Source Shortest Path Problem -

Statement: For a given vertex called the source in a weighted connected graph, find shortest paths to all its other vertices.

→ Best known algorithm for single-source shortest path is, DIJKSTRA'S ALGORITHM.

```
ALGORITHM Dijkstra(G,S)
//Input: A weighted connected Graph G=(V,E)
        with non-negative weights & source vertex S.
//Output: The length dv of shortest path from source S to
        all remaining vertices in |V|-S.
for every vertex v in V do
{
    dv ← ∞;
    Pv ← NULL;
}
ds ← 0
VT ← ∅
For i ← 0 to |V|-1 do
    VT ← VT ∪ {u*}
    for every vertex u in V-VT that is adjacent to u* do
        if du* + w(u*,u) < du
            du ← du* + w(u*,u)
            Pu ← u*
```

Single - Source Shortest Path Problem -

Statement: For a given vertex called the source in a weighted connected graph, find shortest paths to all its other vertices.

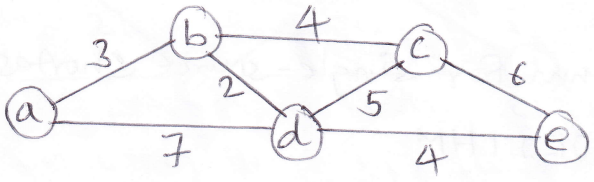
→ Best known algorithm for single-source shortest path is, DIJKSTRA'S ALGORITHM.

```
ALGORITHM Dijkstra(G,S)
//Input: A weighted connected Graph G=(V,E)
        with non-negative weights & source vertex s.
//Output: The length dv of shortest path from source s to
        all remaining vertices in |V|-s.

for every vertex v in V do
{
    dv ← ∞;
    Pv ← NULL;
}
ds ← 0
VT ← ∅
For i ← 0 to |V|-1 do
    VT ← VT ∪ {u*}
    for every vertex u in V-VT that is adjacent to u* do
        if du* + w(u*,u) < du
            du ← du* + w(u*,u)
            Pu ← u*
```

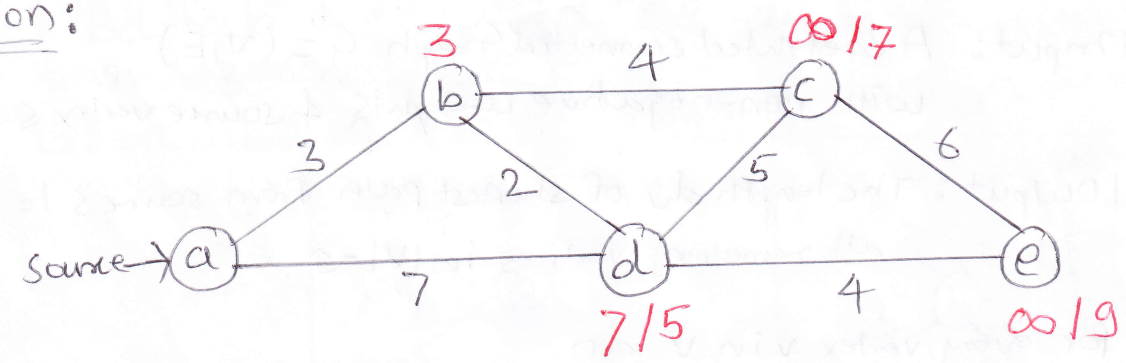
DIJKSTRA'S ALGORITHM EXAMPLE

Exaple- Solve single source shortest path problem for below given graph using Dijkstra's Algorithm-



Consider vertex (a) as the source vertex -

Solution:



Shortest Path from Source to All vertices are -

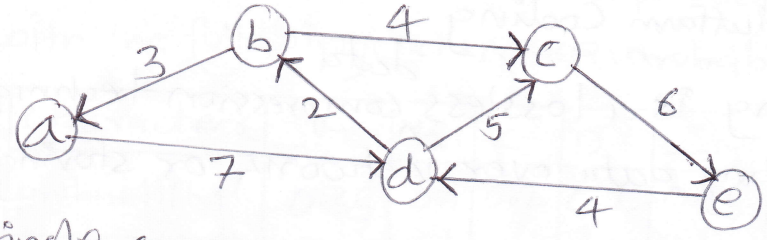
From a to b : $a \xrightarrow{3} b$ of length 3

From a to c : $a \xrightarrow{3} b \xrightarrow{4} c$ of length 7

From a to d : $a \xrightarrow{3} b \xrightarrow{2} d$ of length 5

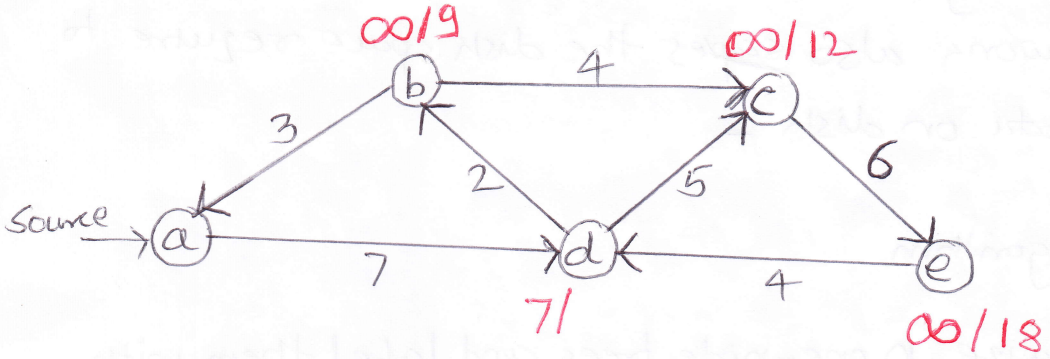
From a to e : $a \xrightarrow{3} b \xrightarrow{2} d \xrightarrow{4} e$ of length 9

DIJKSTRAM'S ALGORITHM EXAMPLE



Find single source shortest path for above given graph from source vertex @.-

Solution:



Shortest Path from source @ to all vertices -

from a to b - a 7 d 2 b of length 9

from a to c - a 7 d 5 c of length 12

from a to d - a 7 d of length 7

from a to e - a 7 d 5 c 6 e of length 18

Huffman Trees & Huffman Coding

→ Huffman Coding is a lossless data compression technique used while transmitting data over network or storing data on the disk.

→ Huffman coding assigns codewords of different lengths to different characters that you want to transfer over n/w.

→ Huffman Coding saves the transmission time of data over network also saves the disk space require to save data on disk.

Huffman's Algorithm

Step-1 - Initialize n one-node trees and label them with the character of the alphabet. Record the frequency of each character in its tree's root to indicate the tree's weight.

Step 2 - Repeat the following operation until a single tree is obtained - find two trees with the smallest weight. Make them left & right subtree of a new tree & record the sum of their weights in the root of the new tree as its weight.