BYJU'S
EXAM PREP

# Vision 2023

A Course for GATE & PSUs

Computer Science Engineering

Algorithm

CHAPTER 4

Greedy Method

# CHAPTER 4 ALGORITHM

## GREEDY METHOD

- "Greedy Method finds out of many options, but you have to choose the best option."
  In this method, we have to find out the best method/option out of many present ways.
  In this approach/method we focus on the first stage and decide the output, don't think about the future.
- Greedy Algorithms solve problems by making the best choice that seems best at the particular moment. Many optimization problems can be determined using a greedy algorithm. Some issues have no efficient solution, but a greedy algorithm may provide a solution that is close to optimal.
  A greedy algorithm works if a problem exhibits the following two properties:
  1. **Greedy Choice Property:** A globally optimal solution can be reached at by creating a locally optimal solution. In other words, an optimal solution can be obtained by creating "greedy" choices.
  2. **Optimal substructure:** Optimal solutions contain optimal sub-solutions. In other words, answers to subproblems of an optimal solution are optimal.

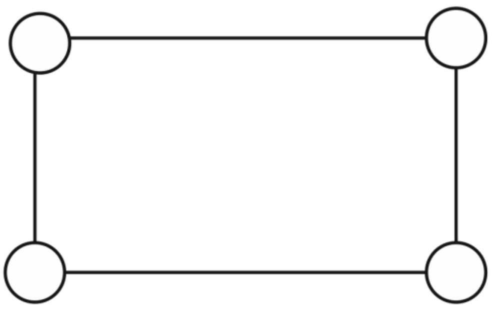
**Spanning Tree :**

Let G (V, E) be an undirected connected graph A subset T (V, E') of G (G, E) is said to be a spanning tree if T is a tree.

**Connected graph :**

In a connected graph between every pair of vertices there exists a path.

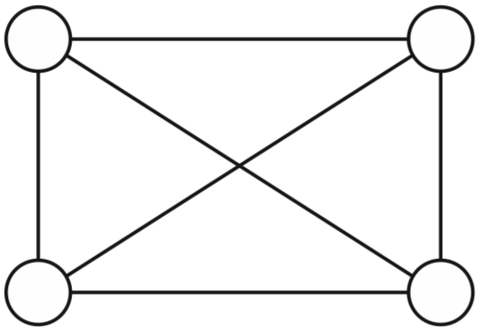
e.g.



**Complete graph :**

In a complete graph between every pair of vertices there exists are edge.

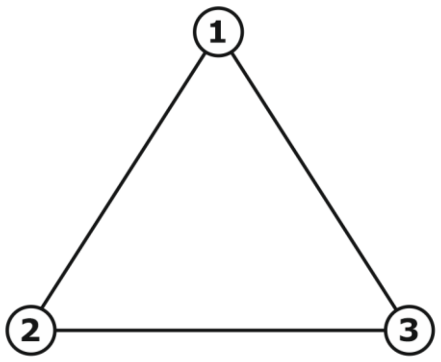
e.g.



→ Total no of edges in a complete undirected graph with n vertices = (n – 1) + (n – 2) + ...... + 0

$= \frac{n(n-1)}{2}$

e.g.

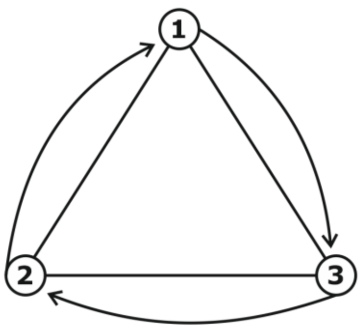


| | | | | |
|---|---|---|---|---|
| Vertices : | 1 | 2 | 3 | |
| Outbeg : | 2 | +1 | + 0 | = 3 |

→ Total no of edge in a complete directed graph with n vertices $= 2 \times \left(\frac{n(n-1)}{2}\right)$

$= n\,(n - 1)$

e.g.



⇒ 2 × 3 = 6 edges

**Theorem :**

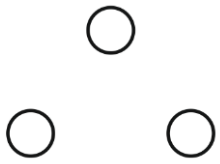
Prove that maximum no. of undirected graph with 'n' vertices = 2 no. of edges

e.g. Take n = 3 vertices.

⇒ Maximum no. of edges $\frac{n(n-1)}{2} = \frac{3(3-1)}{2} = 3\,\text{edges}$

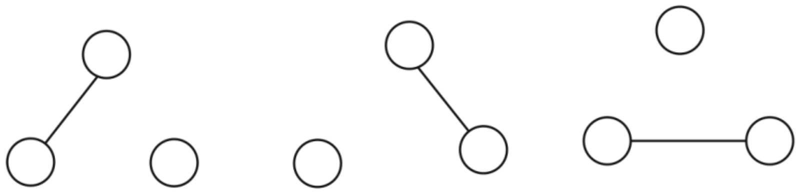
↓ undirected graph.

**Case (i) :** Graph with 'o' no of edges.



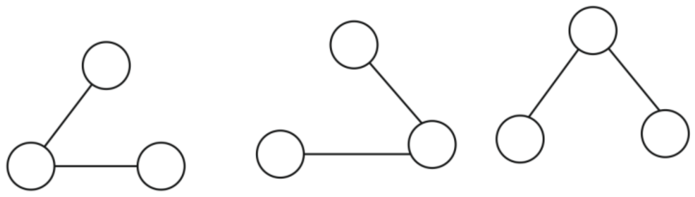
= $^3C_0$ = 1

**Case (ii) :** Graph with '1' no. of edges.



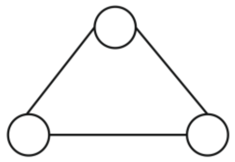
= $^3C_1$ = 3

**Case (iii) :** Graph with '2' no. of edges.



= $^3C_2$ = 3

**Case (iv) :** Graph with '3' no. of edges.



= $^3C_3$ = 1

∴ Total no. of graph = 1 + 3 + 3 + 1 = 8

**Proof :**

n = vertices

edges $= \frac{n(n-1)}{2}$

(i) Graph with '0' edge $= \frac{n(n-1)}{2_{C_0}}$

(ii) Graph with '1' edge $= \frac{n(n-1)}{2_{C_1}}$

Graph with $\frac{n(n-1)}{1}$ edges = $\frac{n(n-1)}{2}C\frac{n(n-1)}{2}$

Total no of graphs $= \frac{n(n-1)}{2_{C_0}} + \frac{n(n-1)}{2_{C_1}} + \ldots\ldots + \frac{n(n-1)}{2_{C_{\frac{n(n-1)}{2}}}}$

$$= 2\frac{n(n-1)}{2} \left[\because {}^nC_0 + {}^nC_1 + \ldots . + {}^nC_n = 2^n\right]$$

∴ Total no. of graph = 2 No. of edges.

**Properties of spanning tree :**

1) Every spanning tree of G (V, E)
2) To construct spanning tree of G (V, E) we have to remove (E – V + 1) no. of edges from G (V, E)
3) Every spanning tree is maximally a cyclic that is by addition of 1 edge to the S.T. it forms a cycle.
4) Every spanning tree (S.T.) is minimally connected i.e. by the removal of a 1 edge from ST it becomes disconnected.
5) Maximum no of possible ST of the complete graph G (V, E) is $V^{V-2}$.
6) To identify minimum cost ST out of $V^{V-2}$ ST we can take help of either prim's or Kruskal's algorithm.

**Ex:**

Consider a weighted complete graph G on the vertex set $\{v_1, v_2, \ldots.. v_n\}$ such that weight of are edge <vi, vj> = 2 | i – j |. Then weight of minimum cost spanning tree (MST) =

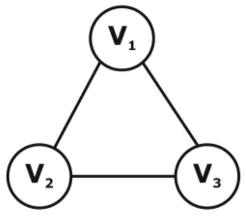
A. $n^2$

B. 2n – 1

C. 2n – 2

D. n/2

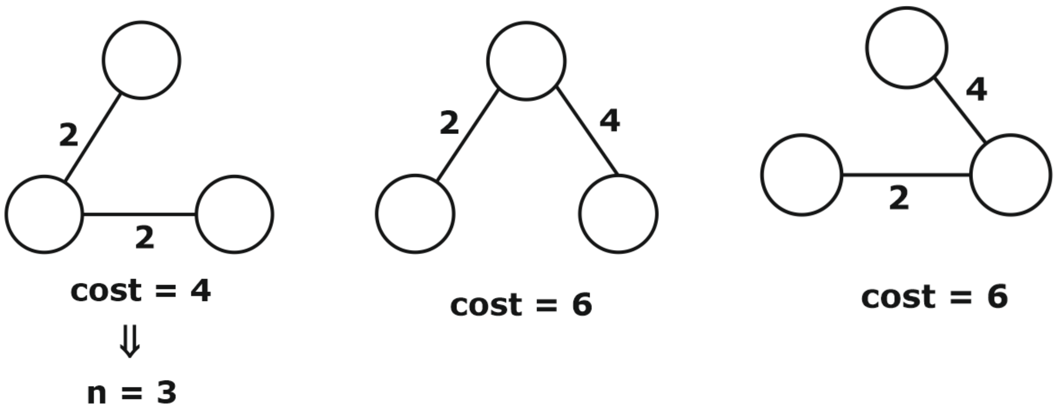
→ e.g.



⇒ <vi, vj> = 2 |i – j|

= 2 |1 – 2|

= 2



Put n = 3 is options & check out gives 4 or not.

$$\begin{tikzcd}
 & \text{Graph} \arrow[dl] \arrow[dr] & \\
\text{Adjacency Matrix} \arrow[d] \arrow[drr] & & \text{Adjacency List} \\
\text{Unweighted} & & \text{Weighted}
\end{tikzcd}$$

$$\begin{array}{c|ccc|} & 1 & 2 & 3 \\ \hline 1 & 0 & 1 & 1 \\ 2 & 1 & 0 & 1 \\ 3 & 1 & 1 & 0 \\ \hline \end{array}_{3 \times 3} \qquad \begin{array}{|cccc|} & 1 & 2 & 3 \\ \hline 1 & 0 & 2 & 3 \\ 2 & 2 & 0 & 4 \\ 3 & 3 & 4 & 0 \\ \hline \end{array}$$

**Ex:**

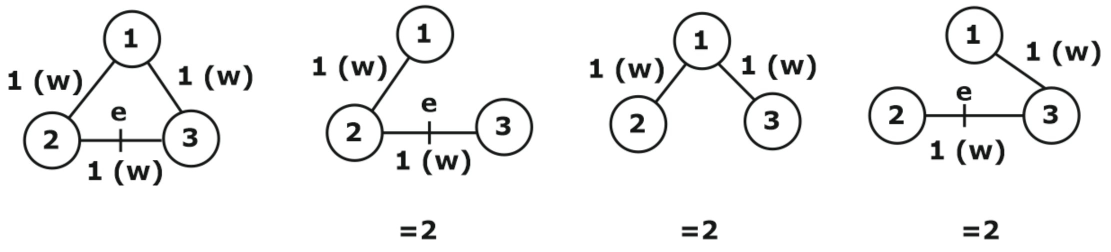
Let ‘w’ be the minimum weight among all edge weights that are undirected connected graphs. Let ‘e’ be a specific edge which contains weight ‘w’. Which of the following is false.

A. There is a MST which contains an edge ‘e’.

B. Every MST contains an edge of weight ‘w’.

C. If ‘e’ is not in MST, ‘T’ then by adding c to ‘T’ it forms a cycle.

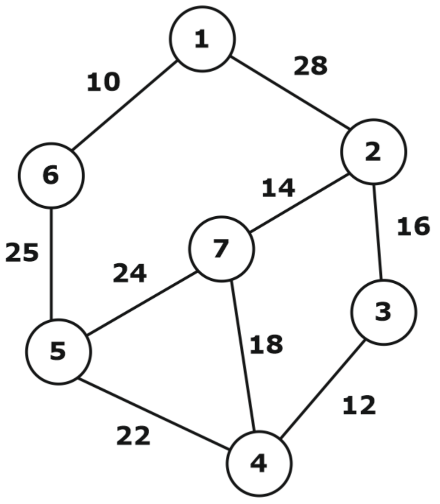
D. Every MST contains an edge ‘e’.

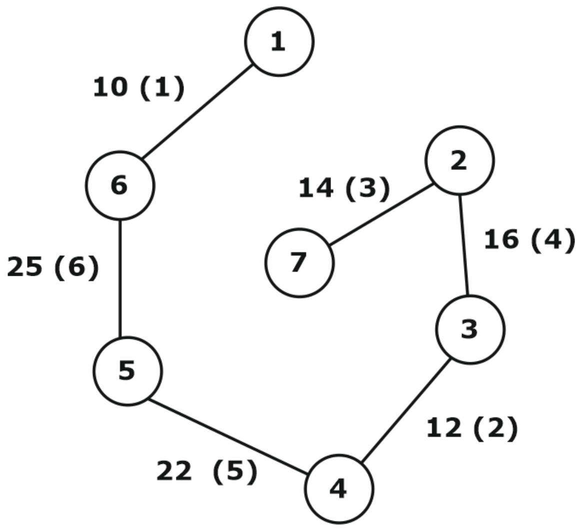


Here we have to take some weight edges because we count more than 1 MST.

**Kruskal's Algorithm :**

Fine MST for the following graph.





COST = 99

It satisfies all the properties of spanning trees.

**Analysis of Kruskal's Algorithm :**

1) Edges must be arrange in the increasing order of their weight with order of $\boxed{\text{ElogE}}$ time.

2) In each iteration delete root node from priority queue with log E time and include it into the partially constructed forest without forming a cycle.

   In the worst case we may perform an E delete operation. So time complexity for deletion is E log E.

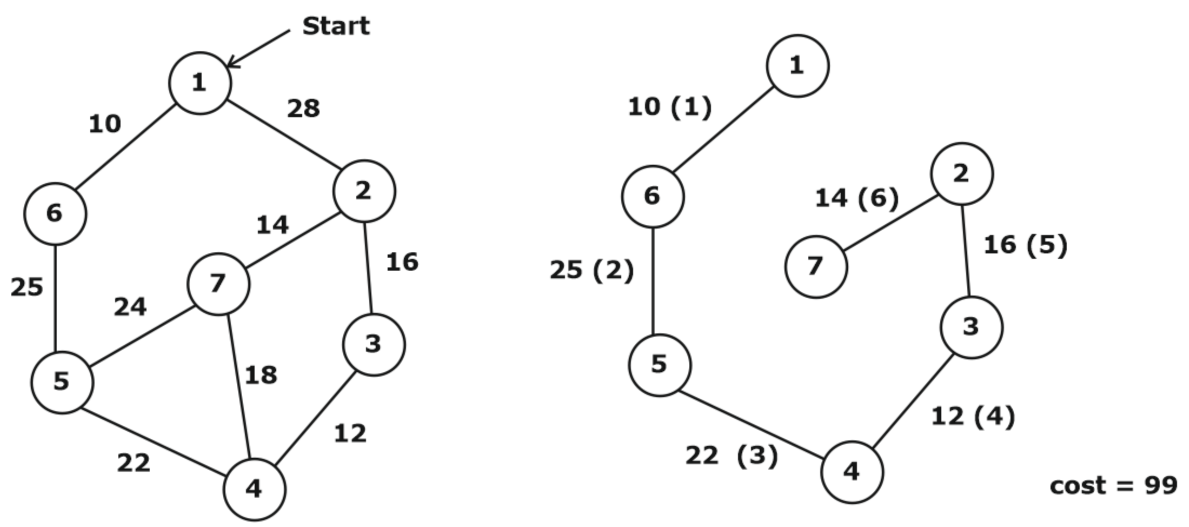
3) Time complexity of K.A. = Step 1 + Step 2

   = E log E + E log E

   = 0 (E log E)

**Prim’s Algorithm :**

Apply Prim’s algo. On the following graph on starting vertex 1.



→ Comparison between Prim’s & K. Algo.

1. Structure of MST w.r.t. Prim’s & K.A. is always some if graph contains distinct edge weights.
2. K.A. does not maintain continuity where prim’s algo. maintain continuity.

**Analysis of Prim’s Algo :**

1) By using adjacency matrix :

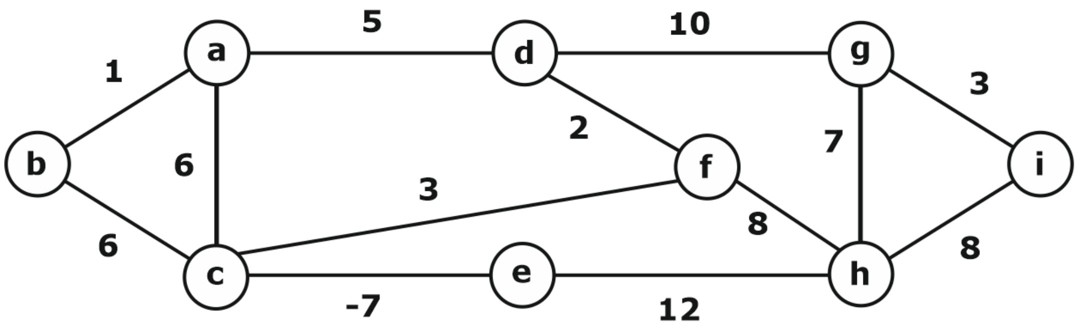
Since the adjacency matrix contains $n^2$ elements, time complexity is bounded by $O(n^2)$.

2) By using binary heap (Priority queue) :

By using binary heap time complexity $\boxed{= 0((v + E)\log V)}$

**Ex :**

For the undirected weighted graph given below which of the following sequence of edges represent correct execution of Prim’s algo to construct MST.



A. (a, b) (d, f) (f, c) (g, i), (d, a) (c, e) (f, h)

B. (c, e) (c, f) (f, d) (d, a) (a, b) (g, h) (h, f) (g, i)

C. (d, f) (f, c) (d, a) (a, b) (c, e) (f, h) (g, h) (g, i)

D. (h, g) (g, i) (h, f) (f, c) (f, d) (d, a) (a, b) (c, e)

→ No. of edges = V – 1 = 8

**Difference between Prim's and Kruskal's algorithm-**

| **PRIM'S ALGORITHM** | **KRUSKAL'S ALGORITHM** |
|---|---|
| It starts to build the Minimum Spanning Tree from any vertex in the graph. | It starts to build the Minimum Spanning Tree from the vertex carrying minimum weight in the graph. |
| It traverses one node more than one time to get the minimum distance. | It traverses one node only once. |
| Prim's algorithm has a time complexity of $O(V^2)$, V being the number of vertices and can be improved up to O(E + log V) using Fibonacci heaps. | Kruskal's algorithm's time complexity is O(E log V), V being the number of vertices. |
| Prim's algorithm gives connected components as well as it works only on connected graphs. | Kruskal's algorithm can generate forest(disconnected components) at any instant as well as it can work on disconnected components |
| Prim's algorithm runs faster in dense graphs. | Kruskal's algorithm runs faster in sparse graphs. |

**SINGLE SOURCE SHORTEST PATHS**

In a shortest- paths problem, we are given a weighted, directed graph G = (V, E), with weight function w: E → R mapping edges to real-valued weights. The weight of path p = (v0,v1,..... vk) is the total of the weights of its constituent edges:

$$w(P) = \sum_{i=1}^{k} w(v_{i-1}v_i)$$

**Single Source Shortest Paths**

We define the shortest - path weight from u to v by δ(u,v) = min (w (p): u→v), if there is a path from u to v, and δ(u,v)= ∞, otherwise.

The shortest path from vertex s to vertex t is then defined as any path p with weight w (p) = δ(s,t).

**1. Dijkstra Algorithm-**

**Single Source Shortest Path (Dijkstra's Algorithm)**

→ Time complexity : O(V + E)log V) using binary min heap.

→Drawback of Dijikstra'sAlgorithm : It will not give the shortest path for some vertices if the graph contain-negative weight cycle.

→ Time complexity of Dijkstra's and Prim's algorithm using various data structures.

(i)Using Binary Heap : O(V + E)log V)

(ii)Fibonacci heap : O((V log V + E))

(iii)Binomial Heap : O((V + E)log V)

(iv)Array : $O(V^2 + E)$

**Conditions-**

It is important to note the following points regarding Dijkstra Algorithm-

- The Dijkstra algorithm works only for connected graphs.
- The Dijkstra algorithm works only for those graphs that do not contain any negative weight edge.
- The actual Dijkstra algorithm does not output the shortest paths.
- It only provides the value or cost of the shortest paths.
- By making minor modifications in the actual algorithm, the shortest paths can be easily obtained.
- Dijkstra algorithm works for directed as well as undirected graphs.

**Time Complexity Analysis-**

**Case-01:**

This case is valid when-

- The given graph G is represented as an adjacency matrix.
- Priority queue Q is represented as an unordered list.

Here,

- A[i,j] stores the information about edge (i,j).
- Time taken for selecting i with the smallest dist is O(V).
- For each neighbor of i, time taken for updating dist[j] is O(1) and there will be maximum V neighbors.
- Time taken for each iteration of the loop is O(V) and one vertex is deleted from Q.
- Thus, total time complexity becomes $O(V^2)$.

**Case-02:**

This case is valid when-

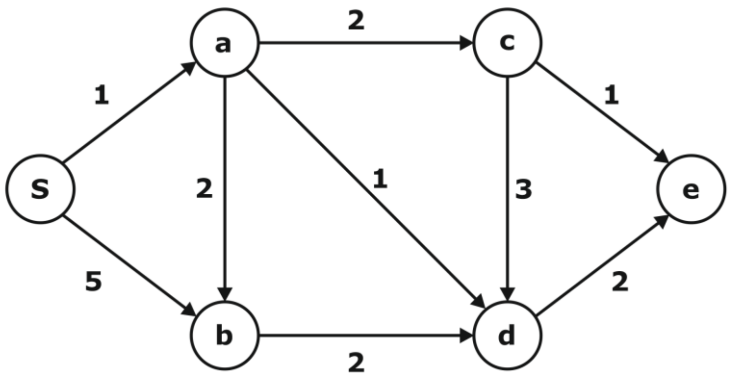
- The given graph G is represented as an adjacency list.
- Priority queue Q is represented as a binary heap.

Here,

- With adjacency list representation, all vertices of the graph can be traversed using BFS in O(V+E) time.
- In min heap, operations like extract-min and decrease-key value takes O(logV) time.
- So, overall time complexity becomes O(E+V) x O(logV) which is O((E + V) x logV) = O(ElogV)
- This time complexity can be reduced to O(E+VlogV) using the Fibonacci heap.

**Example-**

Using Dijkstra’s Algorithm, find the shortest distance from source vertex ‘S’ to remaining vertices in the following graph-



Also, write the order in which the vertices are visited.

**Solution-**

**Step-01:**

The following two sets are created-

- Unvisited set : {S , a , b , c , d , e}
- Visited set : { }

**Step-02:**

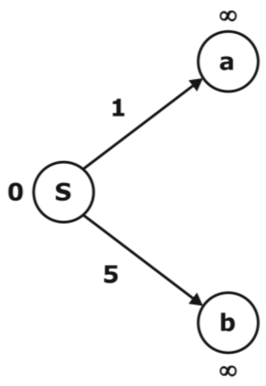
The two variables Π and d are created for each vertex and initialized as-

- Π[S] = Π[a] = Π[b] = Π[c] = Π[d] = Π[e] = NIL
- d[S] = 0
- d[a] = d[b] = d[c] = d[d] = d[e] = ∞

**Step-03:**

- Vertex 'S' is chosen.
- This is because shortest path estimate for vertex 'S' is least.
- The outgoing edges of vertex 'S' are relaxed.

**Before Edge Relaxation-**



Now,

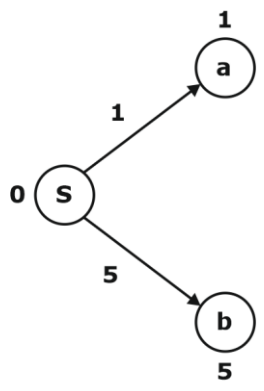
- d[S] + 1 = 0 + 1 = 1 < ∞

∴ d[a] = 1 and Π[a] = S

- d[S] + 5 = 0 + 5 = 5 < ∞

∴ d[b] = 5 and Π[b] = S

After edge relaxation, our shortest path tree is-



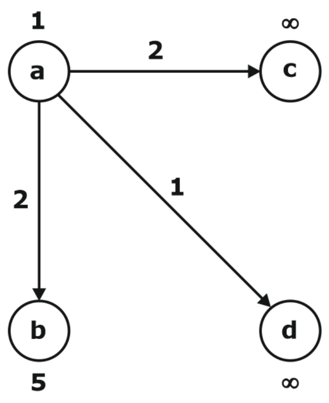
Now, the sets are updated as-

- Unvisited set : {a , b , c , d , e}
- Visited set : {S}

**Step-04:**

- Vertex 'a' is chosen.
- This is because shortest path estimate for vertex 'a' is least.
- The outgoing edges of vertex 'a' are relaxed.

**Before Edge Relaxation-**



Now,

- d[a] + 2 = 1 + 2 = 3 < ∞

∴ d[c] = 3 and Π[c] = a

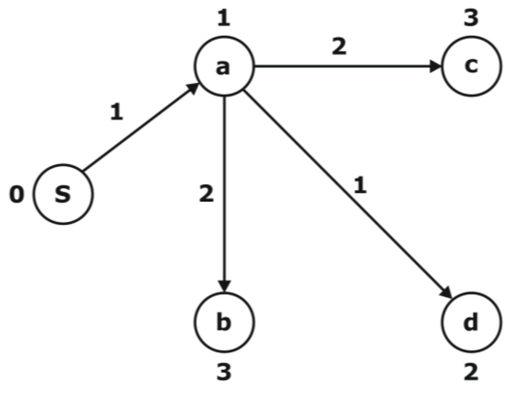
- d[a] + 1 = 1 + 1 = 2 < ∞

∴ d[d] = 2 and Π[d] = a

- d[b] + 2 = 1 + 2 = 3 < 5

∴ d[b] = 3 and Π[b] = a

After edge relaxation, our shortest path tree is-



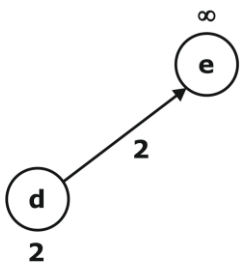
Now, the sets are updated as-

- Unvisited set : {b , c , d , e}
- Visited set : {S , a}

**Step-05:**

- Vertex 'd' is chosen.
- This is because shortest path estimate for vertex 'd' is least.
- The outgoing edges of vertex 'd' are relaxed.

**Before Edge Relaxation-**

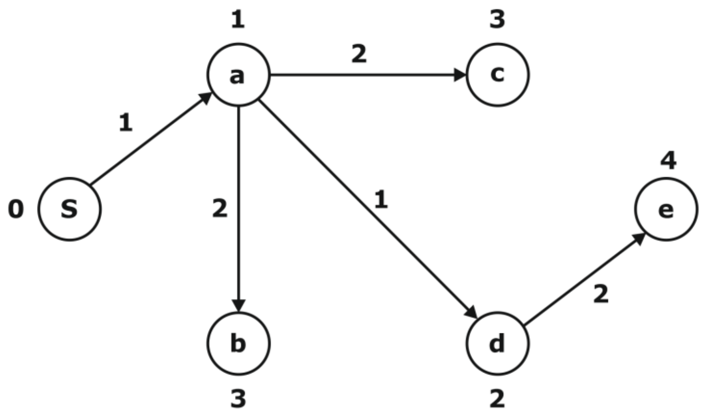


Now,

- d[d] + 2 = 2 + 2 = 4 < ∞

∴ d[e] = 4 and Π[e] = d

After edge relaxation, our shortest path tree is-



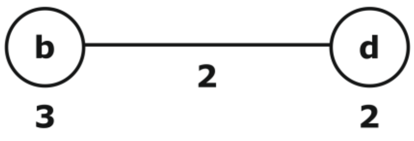
Now, the sets are updated as-

- Unvisited set : {b , c , e}
- Visited set : {S , a , d}

**Step-06:**

- Vertex 'b' is chosen.
- This is because shortest path estimate for vertex 'b' is least.
- Vertex 'c' may also be chosen since for both the vertices, shortest path estimate is least.
- The outgoing edges of vertex 'b' are relaxed.

**Before Edge Relaxation-**



Now,

- d[b] + 2 = 3 + 2 = 5 > 2

∴ No change

After edge relaxation, our shortest path tree remains the same as in Step-05.

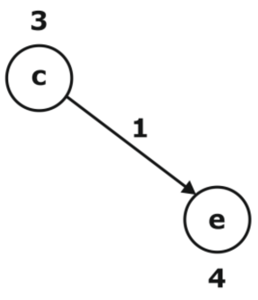
Now, the sets are updated as-

- Unvisited set : {c , e}
- Visited set : {S , a , d , b}

**Step-07:**

- Vertex 'c' is chosen.
- This is because the shortest path estimate for vertex 'c' is least.
- The outgoing edges of vertex 'c' are relaxed.

**Before Edge Relaxation-**



Now,

- d[c] + 1 = 3 + 1 = 4 = 4

∴ No change

After edge relaxation, our shortest path tree remains the same as in Step-05.

Now, the sets are updated as-

- Unvisited set : {e}
- Visited set : {S , a , d , b , c}

**Step-08:**

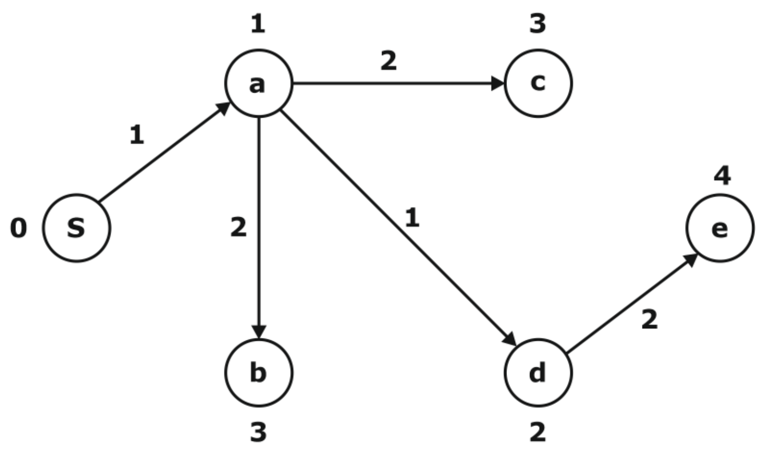
- Vertex 'e' is chosen.
- This is because the shortest path estimate for vertex 'e' is least.
- The outgoing edges of vertex 'e' are relaxed.
- There are no outgoing edges for vertex 'e'.
- So, our shortest path tree remains the same as in Step-05.

Now, the sets are updated as-

- Unvisited set : { }
- Visited set : {S , a , d , b , c , e}

Now,

- All vertices of the graph are processed.
- Our final shortest path tree is as shown below.
- It represents the shortest path from source vertex 'S' to all other remaining vertices.



**Shortest Path Tree**

The order in which all the vertices are processed is :

**S , a , d , b , c , e**

2. **Bellman Ford Algorithm-**

- It finds the shortest path from source to every vertex. If the graph doesn't contain a negative weight cycle.
- If a graph contains a negative weight cycle, it doesn't compute the shortest path from source to all other vertices but it will report saying "negative weight cycle exists".

Time complexity = O(VE) when dense graph E = V2 and for sparse graph E = V.

**Difference Between Bellman fors's and Dijkstra's Algorithm.**

| BELLMAN FORD'S ALGORITHM | DIJKSTRA'S ALGORITHM |
|---|---|
| Bellman Ford's Algorithm works when there is a negative weight edge, it also detects the negative weight cycle. | Dijkstra's Algorithm doesn't work when there is a negative weight edge. |
| The result contains the vertices which contain the information about the other vertices they are connected to. | The result contains the vertices containing whole information about the network, not only the vertices they are connected to. |
| It can easily be implemented in a distributed way. | It can not be implemented easily in a distributed way. |
| It is more time consuming than Dijkstra's algorithm. Its time complexity is O(VE). | It is less time consuming. The time complexity is O(E logV). |
| Dynamic Programming approach is taken to implement the algorithm. | Greedy approach is taken to implement the algorithm. |

**Job sequencing problem :**

Find maximum profit by processing below jobs.

| Job | $J_1$ | $J_2$ | $J_3$ | $J_4$ |
|---|---|---|---|---|
| Dead lines | 2 | 1 | 2 | 1 |
| Profit | 100 | 10 | 15 | 27 |

→ Let us consider n jobs ($J_1$, $J_2$, ......... $J_n$)

→ Each job having deadline di & it cue process the job within its deadline we

→ only one job can be process at a time

→ Only one CPU is available for processing all jobs.

→ CPU can take only one unit at time for processing any job.

→ All jobs arrived at the same time.

→ Objective of job sequencing (JS) is process as many job as possible within its dead line & generate maximum profit.

**Shortcut :**

If there are n jobs, possible subsets are $2^n$. Objective of J.S. is to find the subset which generates maximum profit.

Arrange all jobs in decreasing order of process their profit & then process in that order within its deadline.

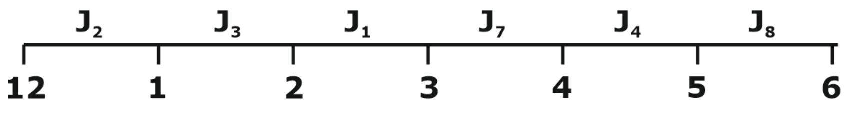
$J_1 \quad \geq \quad J_4 \quad \geq \quad J_3 \quad \geq \quad J_2$

$<J_1 . J_4> \Rightarrow 100 + 27 = 127$

**Ex:**

| | $J_1$ | $J_2$ | $J_3$ | $J_4$ | $J_5$ | $J_6$ | $J_7$ | $J_8$ |
|---|---|---|---|---|---|---|---|---|
| Deadline | 6 | 5 | 6 | 6 | 3 | 4 | 4 | 5 |
| Profit | 10 | 8 | 9 | 12 | 3 | 6 | 11 | 13 |

$J_8 \geq J_4 \geq J_7 \geq J_1 \geq J_3 \geq J_2 \geq J_6 \geq J_5$



Max. Profit = 63

**Ex :** Linked que.

| Task | $T_1$ | $T_2$ | $T_3$ | $T_4$ | $T_5$ | $T_6$ | $T_7$ | $T_8$ | $T_9$ |
|---|---|---|---|---|---|---|---|---|---|
| Deadline | 7 | 2 | 5 | 3 | 4 | 5 | 2 | 7 | 3 |
| Profit | 15 | 20 | 30 | 18 | 18 | 10 | 23 | 16 | 25 |

1. Are all tasks completed ?

A. All are completed

B. $T_1$ & $T_6$ are left out

C. $T_4$ & $T_6$ are left out

D. $T_1$ & $T_8$ are left out

Ans. C

$T_3 \geq T_9 \geq T_7 \geq T_2 \geq T_4 \geq T_5 \geq T_8 \geq T_1 \geq T_6$

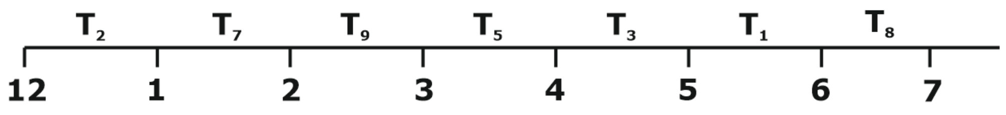
2. What is the maximum profit?

A. 144

B. 147

C. 150

D. 152



**Analysis of Job sequencing :**

To implement J.S. we use a priority queue where priorities are assigned to the profits of a job. So, the time complexity is O (n log n).

**Knapsack Problem :**

Find maximum profit by placing below objects into the knapsack.

| Objects | $0_1$ | $0_2$ | $0_3$ |
|---|---|---|---|
| $(W_1\ W_2\ W_3)$ | 18 | 15 | 10 |
| $(P_1\ P_2\ P_3)$ | 25 | 24 | 15 |

Greedy about weight ⇒ Profit $= 0(25) + \frac{10}{15}(24) + 1(15)$

= 31

Greedy about profit ⇒ Profit $= 1(25) + \frac{2}{15}(24) + 0(15)$

= 28.5

Greedy about unit weight profit ⇒

$$\left.\begin{array}{l} \frac{P_1}{W_2} = \frac{25}{18} = 1.3 \\ \frac{P_2}{W_2} = \frac{24}{15} = 1.6 \\ \frac{P_3}{W_3} = \frac{15}{10} = 1.5 \end{array}\right\} \frac{P_2}{W_2} \geq \frac{P_3}{W_3} \geq \frac{P_1}{W_1}$$

$\therefore$ Profit $= 0(25) + 1(24) + \frac{5}{10}(15)$

= 31.5

**Shortcut :**

Arrange all unit weight profit into decreasing order & then process objects in that order without exceeding knapsack size.

→ Since we are using priority queue, where priority is assigned to unit weight profit. So, time complexity = 0 (n log n)

**Note :**

Let 'x' denote decision on ith object then $0 \leq x_i \leq 1$ (Fractional knapsack problem).

**Optimal Merge Problem :**

Let us consider 3 files $F_1$, $F_2$ & $F_3$ with their corresponding record length 30, 10 & 20 respectively.

Since n = 3, we can arrange in n ! ⇒ 3 ! ⇒ 6 ways.

But merging can be done in $\frac{n!}{2}$ ways $\Rightarrow \frac{3!}{2} = 3$ ways.

Objective of OMP is out of $\frac{n!}{2}$ way, we have to find the weight with least no. of record movement.

**Shortcut :**

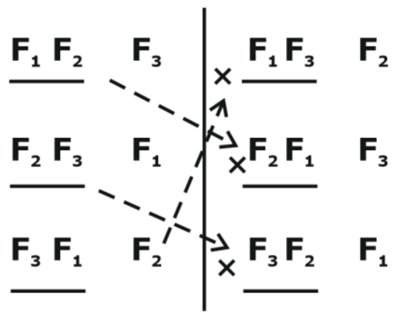
Arrange all files in the increasing order of their record length and in **each iteration** select two files which are having least no. of records and merge them into a single file.

Continue the process until all files are merged.

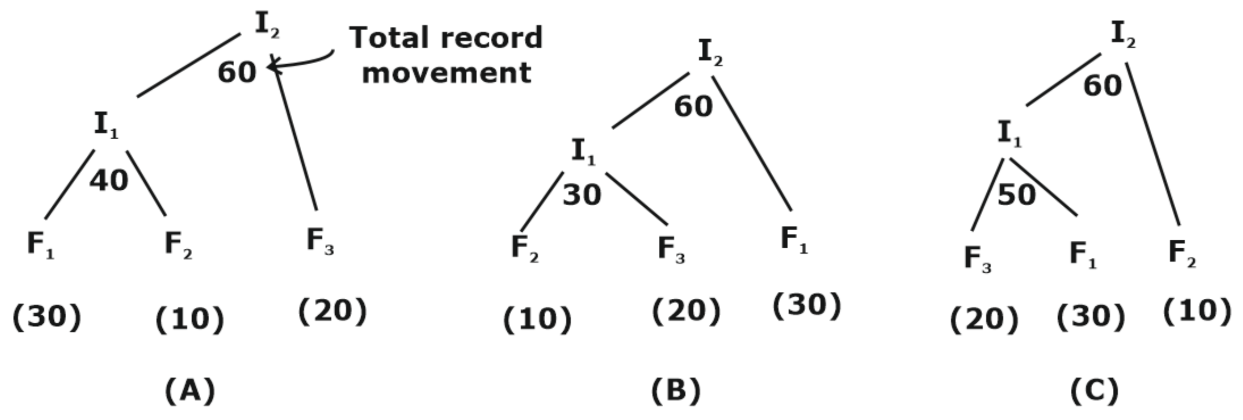
→ Here, n = 3

| | | |
|---|---|---|
| $F_1$ | $F_2$ | $F_3$ |
| (30) | (10) | (20) |

3! = 6 ways



$$\Rightarrow \frac{n!}{2} = \frac{3!}{2} = 3 \text{ ways}$$



$$(I_1 + I_2)$$

| | | |
|---|---|---|
| In (a) Total no. of record movements | = | 100 |
| In (B) Total no. of record movements | = | 90 |
| In (C) Total no. of record movements | = | 110 |

Find least no. of record movement.

| | | | | | | |
|---|---|---|---|---|---|---|
| $F_1$ | $F_2$ | $F_3$ | $F_4$ | $F_5$ | $F_6$ | $F_7$ |
| 4 | 8 | 15 | 3 | 7 | 9 | 10 |

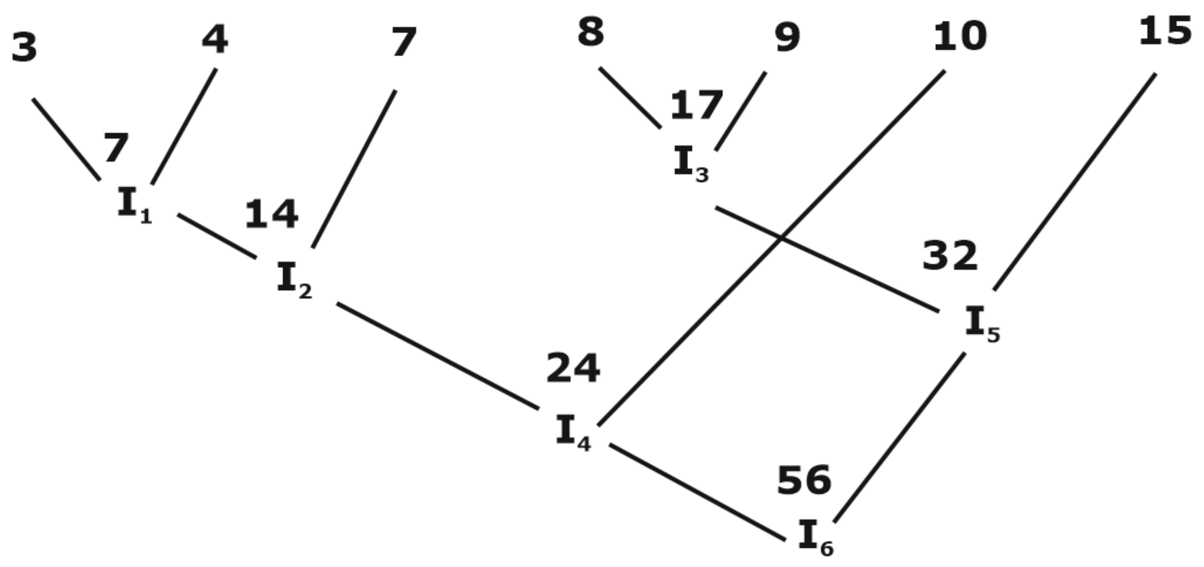
A. 150

B. 171

C. 169

D. 170

$F_4 \leq F_1 \leq F_5 \leq F_2 \leq F_6 \leq F_7 \leq F_3$



**Note :**

Total no. of record movements $= \sum_{i=1}^{n} d_i q_i$

Where, $d_i$ = distance from root to ith file

$q_i$ = Length of ith file.

For above

Total no. of record movements

= 4(3) + 4(4) + 3(7) + 3(8) + 3(9) + 2(10) + 2(15)

= 150

⇒ Analysis :

- Time complexity = 0 (n log n)

- We are using a priority queue where priorities are assigned to record lengths.

**Huffman Encoding :**

Objective of H.E. is to encode letters with least no. of bits. Let us consider a Gmail application which contains letters a, b, c, d, e. With corresponding frequency 10, 20, 4, 15, 6 respectively.

Therefor there are 5 letters to encode letter we need at least 3-bits,

So, total no. of bits required = (10 + 20 + 4 + 5 + 6) × 3 = 165

No. of bits. No. of letters to be encoded

$1 \quad \begin{cases} 0 - a \\ 1 - b \end{cases}$

$$2\begin{cases}00-a\\01-d\\10-c\\11-b\end{cases}$$

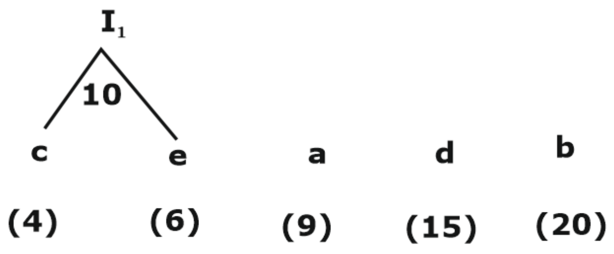
$$3\begin{cases}000-\\001-\\010-\\011-a\\100-e\\101-h\\110-c\\111-d\end{cases}$$

**Step 1**

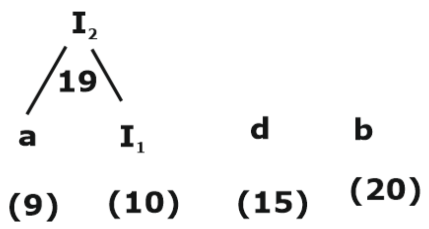
Arrange all letters in the increasing order of their frequencies & then in each iteration construct binary tree by assigning 1st least frequency letter as left child & 2nd least frequencies letter as a sight child.

**Step 2**

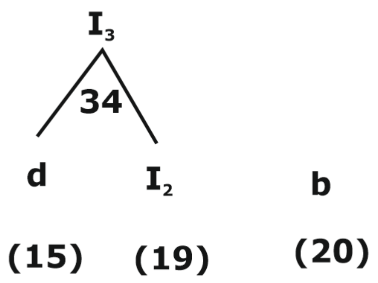
After constructing a binary tree from root to leaf path every left branch is assigned with O & right branch is assigned with 1.



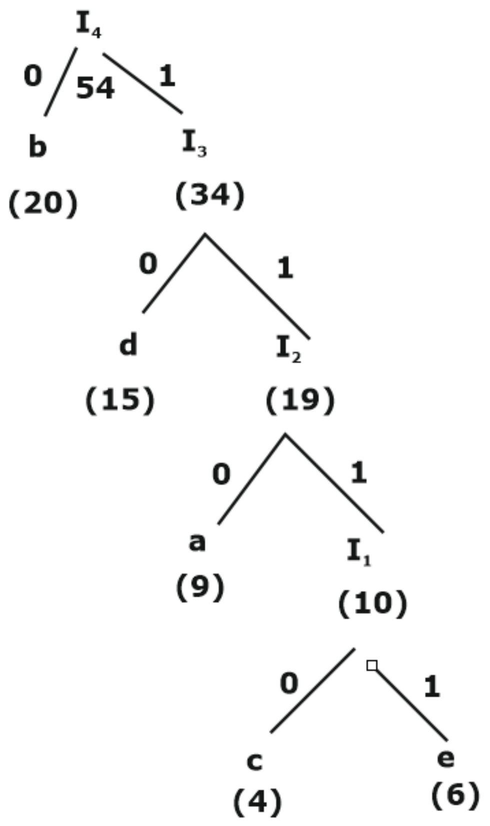
(I)



(II)



(iii)



(iv)

|   |   |   |   |   | $d_i$ $q_i$ |
|---|---|---|---|---|---|
| a | 1 | 1 | 0 |   | $= 3 \times 9 = 27$ |
| b | 0 |   |   |   | $= 1 \times 20 = 20$ |
| c | 1 | 1 | 1 | 0 | $= 4 \times 4 = 16$ |
| d | 1 | 0 |   |   | $= 2 \times 15 = 30$ |
| e | 1 | 1 | 1 | 1 | $= 4 \times 6 = \underline{24}$ |
|   |   |   |   |   | 117 bits |

**Note :**

Total no of bits required $= \sum_{i=1}^{n} d_i q_1$

Where, $d_i$ = distance from root to ith letter

$q_i$ = frequency of ith letter

to with →

In H.E. a more frequently occurring letter is encoded with least no. of bits.

****