3 Greedy Techniques

Greedy Algorithms

Algorithms for optimisation often follow a number of steps, each with a set of options.

- Dynamic programming is used to solve numerous optimisation challenges.
- Greedy Algorithms select a path that is optimal at that moment, expecting it leads to an optimal solution for the entire problem in future.
- A greedy algorithm won't work all the time.

Greedy algorithms have two main components: Greedy choice property and optimal substructure.

Greedy choice property:

The greedy-choice property is the first essential component.

Selecting locally optimal solutions which may lead us to global optimal solutions.

In other words, we choose the choice that appears to be the best in the current solving problem by not taking into account the results of subproblems.

Optimal substructure:

- If the global optimal solution includes the local optimal solutions, the problem has an optimal substructure. It is very important in both dynamic programming and greedy paradigms.

Huffman Codes

- Huffman coding compresses data very effectively.
- Consider the information as a string of characters.
- Huffman's algorithm finds an optimal way of expressing a character in binary. It gives less number of bits to characters with higher frequency and more number of bits to characters with lower frequency.

Example:

Let's say there exists a file with 130 characters. The characters and their respective frequencies in the file are given below:

Character	A	B	C	D	E	F	G
Frequency	7	11	12	19	21	25	35

Solution:

For 7 characters, we need a minimum of 3 bits to represent.

Character	A	B	C	D	E	F	G
Frequency	7	11	12	19	21	25	35
Fixed length Codeword	000	001	010	011	100	101	111

Variable length codeword:

Character	A	B	C	D	E	F	G
Frequency	7	11	12	19	21	25	35
Fixed length Codeword	0110	0111	010	110	111	00	10

For fixed length number of total bits = (7 + 11 + 12 + 19 + 21 + 25 + 35) × 3 = 390 bits

For variable length number of total bits = 4×7 + 4×11 + 3×12 + 3×19 + 3×21 + 2×25 + 2×35 = 348 bits.

Prefix codes:

- Prefix codes are codes which do not have the same prefix. These are used to encode and decode the characters in an optimal way.

Constructing a Huffman code:

- Huffman designed the Huffman code, a greedy approach for constructing an optimised prefix code.

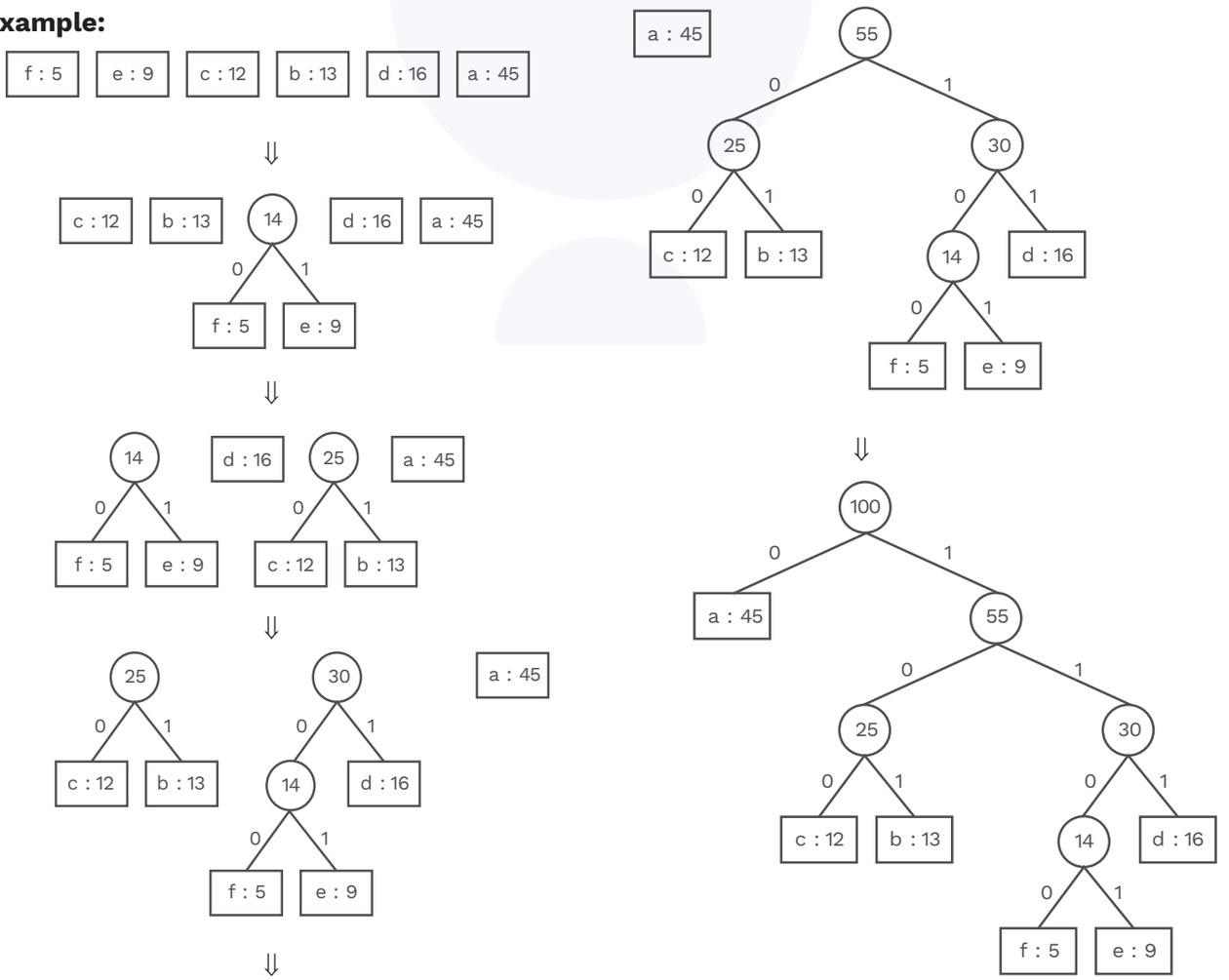
Pseudocode:

Huffman(C) // C represents the set of n character

1. n = |C|
2. Q = Create a min priority queue and make the string C inserted into it
3. for i = 1 to n–1
4. allocate a new node z
5. z.left = x = EXTRACT_MIN(Q)
6. z.right = y = EXTRACT_MIN(Q)
7. z.freq = x.freq + y.freq
8. INSERT (Q,z)
9. return EXTRACT_MIN(Q) // returns the root of the tree.

- In the pseudocode above, we assume that C is a character set of size n.
- Where character c ∈ C denotes an item with the c.freq attribute indicating its frequency.
- In a bottom-up approach, the algorithm constructs the tree T that corresponds to the optimal code.
- To produce the final tree, it starts with a set of |C| leaves and executes a series of |C|-1 "merging" operations.
- All the frequencies are in min priority Queue Q. The algorithm identifies two characters with the least frequency, removes their freq in min priority Queue, sums up the two frequencies then adds them to the Priority queue, which works based on the freq attribute.

Example:



- The ideal prefix code is represented by the final tree.
- The straightforward path from the root to the letter is the letter's codeword.

Analysis:

- We assume that the Queue is implemented as a binary min heap in the algorithm.
- In line 2, the Queue Q is intialised in O(n) for a character set of n characters (using BUILD_MIN_HEAP).
- From lines 3-8, the for loop iterates for exactly n-1 times, and each heap operation takes O(logn), which results in O(nlogn) time.
- Hence, the total time taken is O(nlogn), where n is the number of characters present in the character set C.

Previous Years' Question

A message is made up entirely of characters from the set X = (P, Q, R, S, T). The table of probabilities of each character is shown below:

Character	Probability
P	0.22
Q	0.34
R	0.17
S	0.19
T	0.08
Total	1.00

A message of 100 characters over X is encoded using Huffman coding. Then the excepted length of the encoded message in bits is __ **[2017 (Set-2)]**

(A) 225 (B) 226
(C) 227 (D) 228

Solution: (A)

Previous Years' Question

Consider the string abbccddeee. Each letter in the string must be assigned a binary code satisfying the following properties:

1. For any two letters, the code assigned to one letter must not be a prefix of the code assigned to the other letter.

2. For any two letters of the same frequency, the letter which occurs earlier in the dictionary order is assigned a code whose length is at most the length of the code assigned to the other letter.

Among the set of all binary code assignments which satisfy the above two properties, what is the minimum length of the encoded string? **[2021 (Set-2)]**

(A) 21 (B) 23
(C) 25 (D) 30

Solution: (B)

Fractional Knapsack Problem

- Let's assume a thief went to rob a store containing n items. Let w_i be the weight of the ith item and v_i is the value of the ith item.He has a knapsack that can handle a weight of w (integer).
- The items can be picked in fractions, and the problem is to find the set of items whose weight sums up to a weight w and whose value is maximum.

Example:

- When a thief walks into a store, he notices the following items.

Items	A	B	C
Cost	100	10	120
Weight	2 pounds	2 pounds	3 pounds

Knapsack holds, i.e (w) = 4 pounds

- Since, the thief can take a fraction of an item

$$\text{Solution} = \begin{cases} \text{2 pounds of item A} \\ + \\ \text{2 pounds of item C} \end{cases}$$

Value = 100 + 80
= 180

Greedy solution for fractional knapsack:

- Given a set of items I.

Weight	w_1	w_2	...	w_n
Cost	C_1	C_2	...	C_n

Let P denote the problem of selecting items from I, with weight limit K, such that the resulting cost (value) is maximum.

(1) Calculate $v_i = \frac{C_i}{w_i}$, for i = 1, 2..., n

(2) Sort the items by decreasing v_i. Let the sorted item sequence be 1,2,...,i,...n and the corresponding v and weight be v_i and w_i, respectively.

(3) Let k be the current weight limit (initially, k = K)

- In each iteration, we choose item i from the head of the unselected list; if k > = w_i, we take item i and update k = k–w_i, then consider the next unselected item.
- If k < w_i, we take a fraction f of item i, i.e., we only take f = $\frac{k}{w_i} (< 1)$ of item I, which weights exactly k. Then the algorithm is finished.
- The algorithm can take an item in fraction, i.e., an item need not be considered completely.

Previous Years' Question

Consider the weights and values of items listed below. Note that there is only one unit of each item.

Item number	Weight (in kgs)	Value (in Rupees)
1	10	60
2	7	28
3	4	20
4	2	24

The task is to pick a subset of these items such that their total weight is no more than 11 kgs and their total value is maximised. Moreover, no item may be split. The total value of items picked by an optimal algorithm is denoted by V_{opt} A greedy algorithm sorts the items by their value to weight ratios in descending order and packs them greedily, starting from the first item in the ordered list. The total value of items picked by the greedy algorithm is denoted by V_{greedy}. The value of V_{opt}-V_{greedy} is

Range [16 to 16] **[NAT]**

Time complexity:

- The sorting of all items in decreasing order of their cost/weight ratio is the most time-consuming step. It takes O(n logn) time.
- Iterating through every item in step (3) takes O(n) time if the items are already arranged in the required order.

Total time complexity = O(nlogn) + O(n) = O(nlogn)

Job Sequencing Problem

- Given a list of jobs, each with its own deadline and associated profit if the jobs are completed before the deadline.
- Given that each job takes a single unit of time, the shortest deadline possible for any job is 1.
- When only one job can be scheduled at a time, how can the total profit be maximized?

Example:

Input:

Jobs	A	B	C	D
Deadline	1	4	1	1
Profit	10	20	30	40

Output:

The following is a list of jobs in order of maximum profit.

D, B

Algorithm:

1. First sort (in decreasing order) the profit of given the input.
2. Iterate on jobs in order of decreasing profit.

For each job follow this process:

- Find a time slot i, such that slot is empty and i < deadline and i is a job with the current greatest profit. Keep the job in this slot and mark this slot filled.

Time complexity:

- For sorting the jobs in decreasing order of profit O(nlogn)
- For each job, checking the open time slot between deadline and 0 is O(n)
  for 'n' jobs $\rightarrow$ n * O(n)
  $= O(n^2)$

$\therefore$ Total time complexity = $O(nlogn) + O(n^2)$
$= O(n^2)$

Representation of graphs:

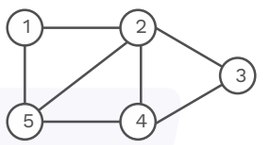
There are two common ways to express a graph G=(V,E): adjacency lists and adjacency matrices.

Adjacency list representation:

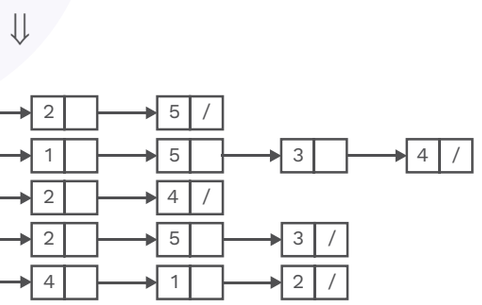
The Graph G(V,E) is expressed as an array Adj of size |V|, one for each element of Adj is a list for each vertex in V.

- For each vertex u in the set of vertices V, there exists an entry in the array Adj i.e. Adj[u] with list of all vertices adjacent to u.

i.e. Adj[u] has links of all the vertices one by one adjacent to u in G.



// An undirected graph G with 5 vertices and 7 edges



- We can readily adapt adjacency lists to represent weighted graphs.
- Simply store weight w(u,v) of the edge (u,v) $\in$ E with vertex v in u's adjacency list.

Weighted graphs:

Definitions

Graphs in which a weight w is given to each edge (u, v).

- If the length of an entry in an adjacency list is given by the number of vertices linked in that entry, then the sum of all

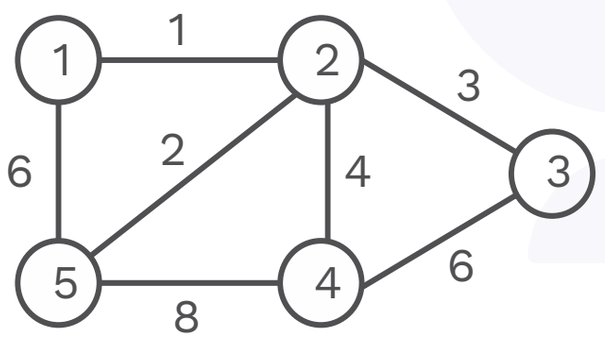
lengths of entries in the adjacency list is |E| for the directed graph and 2|E| for the undirected graph.

- Since an edge (u,v) is represented both in u and v entry for the undirected graph and only in u entry for the directed graph.
- Both the directed and undirect graphs need Θ(V+E) memory for their adjacency list representation.
- One of the disadvantages of the adjacency list is its O(V) time complexity to search whether an edge (u,v) exists in the graph or not.

Adjacency-Matrix representation:

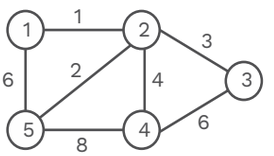
- In an adjacency matrix, a graph G(V,E) is represented as a matrix represented in |V| x |V| size.

$$a_{ij} = \begin{cases} 1 \text{ if } (i,j) \in E, \\ 0 \text{ otherwise} \end{cases}$$



The adjacency-matrix representation of G.

- An adjacency matrix of a graph requires $\Theta(V^2)$ space, independent of the number of edges in the graph.
- An adjacency matrix can represent a weighted graph also.
- For example, if G(V,E) is a weighted graph with edge-weight function w, we can simply store the weight w(u,v) of the edge (u,v) ∈ E as the entry in a row u and column v of the adjacency matrix.



⇓

	1	2	3	4	5
1	0	1	0	0	6
2	1	0	3	4	2
3	0	3	0	6	0
4	0	4	6	0	8
5	6	2	0	8	0

Minimum Spanning Trees

Minimum spanning tree T is a subset of the set of Edges E in a connected and undirected graph G(V, E) such that it forms an acyclic graph with

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

- Since T is acyclic and covers(spans) all the vertices in the graph, it is known as the minimum spanning tree of Graph G.
- The problem of finding subset Tree T in a set of Edges E such that it covers all the vertices and gives minimum total weight is called the minimum spanning tree problem.

Note:
A complete graph can have maximum nn-2 spanning trees, where n is the number of nodes.

Definitions

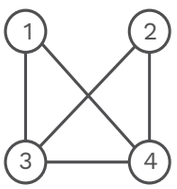
A complete graph is a graph in which each pair of graph vertices is connected by an edge.
A complete graph with n vertices is represented by symbol K_n.

- For non–complete graphs, we can use Kirchhoff's theorem to find number of spanning trees.

Kirchhoff's theorem:

1. For the given graph, create an adjacency matrix.
2. Substitute the degree of nodes for all the diagonal elements.
3. Substitute -1 for all non-diagonal 1's
4. Calculate the co-factor for any element.
   a) The total number of spanning trees for that graph is the cofactor we get.

Example:



// given graph G

1.

	1	2	3	4
1	0	0	1	1
2	0	0	1	1
3	1	1	0	1
4	1	1	1	0

// Adjacency matrix representation of graph G

2.

	1	2	3	4
1	2	0	1	1
2	0	2	1	1
3	1	1	3	1
4	1	1	1	3

// Substituting degree of the node for all the diagonal elements

3.

	1	2	3	4
1	2	0	-1	-1
2	0	2	-1	-1
3	-1	-1	3	-1
4	-1	-1	-1	3

// Substituting -1 for all non diagonal 1's

4. Co-factor of (1,1)
   = 2(9-1) – (-1) (-3-1) -1 (1+3)
   = 16 – 4 – 4
   = 8

∴ Total 8 spanning trees possible for given graph.

The two commonly used algorithms to solve minimum spanning tree problems are:

1. Kruskal's algorithm
2. Prim's algorithm

Kruskal's algorithm:

MST_KRUSKAL (G,w)

1. A = ϕ
2. for each vertex v $\in$ G.V
3. MAKE_SET(v)
4. For every edge (u, v) $\in$ G.E (Mentioned in increasing order by weight(u, v)):
5. if FIND-SET(u) $\neq$ FIND-SET(v):
6. A = A $\cup$ {(u, v)}
7. UNION(u, v) return A

- Set A is a set containing all the edges in the MST in the given graph, which is initially empty.
- The edge added first is always the least weighted edge in the entire graph.
- Kruskal's algorithm comes under the greedy algorithm since it chooses the least possible weight edge to add to the MST, at each step.
- **FIND_SET(u):** Operation that returns an element representing the set that is containing the vertex u.
- This operation helps in comparing whether two vertices u, v belongs to the same tree. (By equalising FIND_SET(u) and FIND_SET(v)).
- **UNION:** Operation used to combine 2 trees.
- Line 1 and Line 3 initialise the set A to empty and creates |V| individual trees for each vertex v $\in$ G.V.
- The for loop in the 5th line sorts edges in non-decreasing order.
- An edge (u,v) is not added to forest A if u and v belong to the same tree since it forms a cycle.
- Otherwise, the edge is added to the forest (done by the 7th line), and the two trees of vertices u and v are merged.

Time complexity analysis:

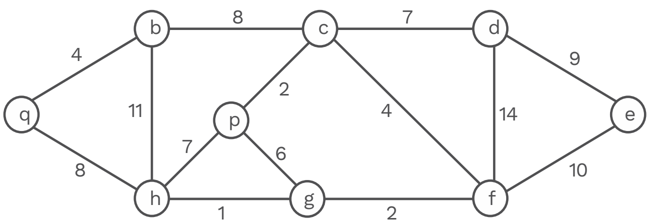
- Initializing set A to empty in Line 1 takes O(1) time.

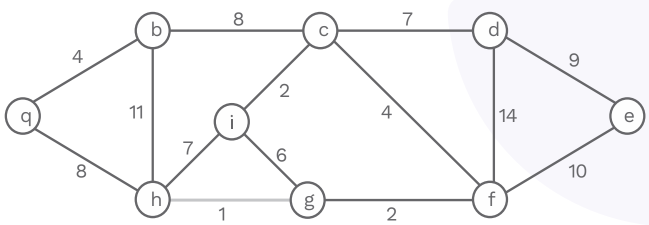
- Sorting edges takes O(ElogE) in line 4.
- For loop takes O(V) in 2nd and 3rd lines.
- For loop takes O(ElogV) to perform O(E) UNION and FIND_SET operations on disjoint_set forest in 5th and 8th lines.

∴ T(n) = O(1) + O(V) + O(ElogE) + O(ElogV) = O(ElogE) + O(VlogV) = O(ElogE) = O(Elog(V(V-1)/2)) = O(ElogV)

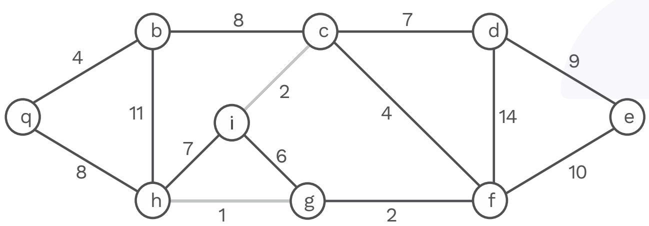
Example:

Applying Kruskal's algorithm on this graph gives,

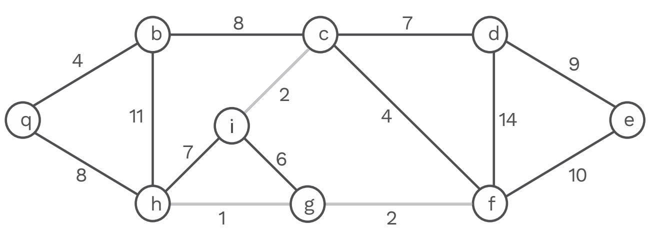




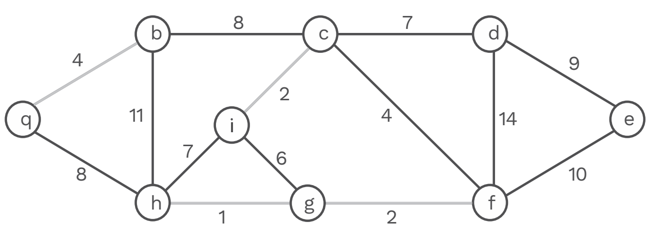
⇓



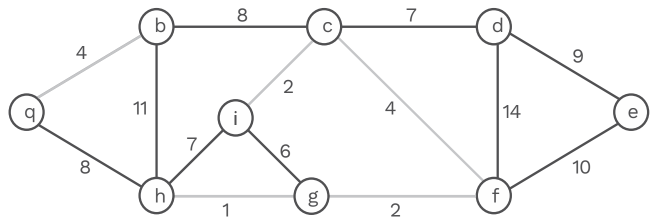
⇓



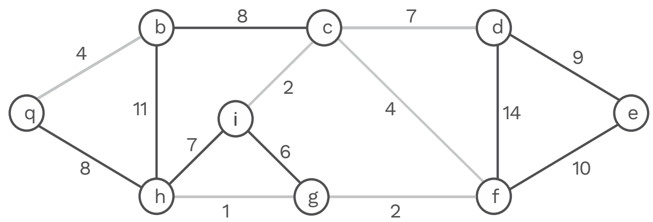
⇓



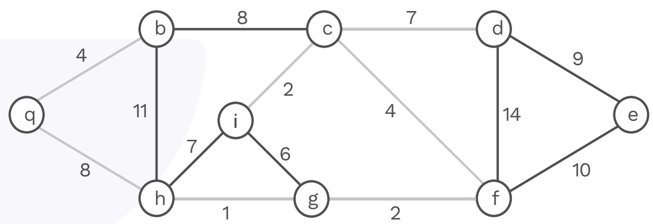
⇓



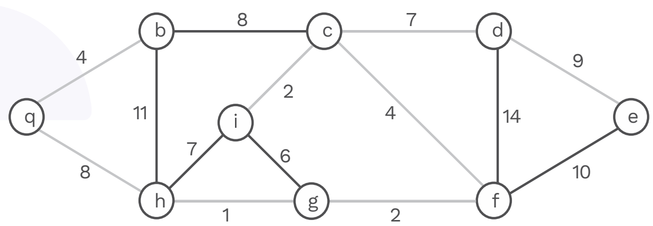
⇓



⇓

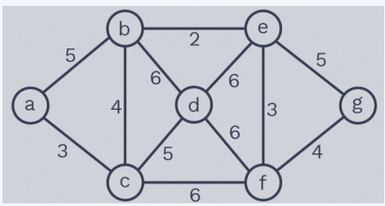


⇓



Previous Years' Question

Consider the following graph:



Which one of the following is not the sequence of edges added to the minimum spanning tree using Kruskal's algorithm?

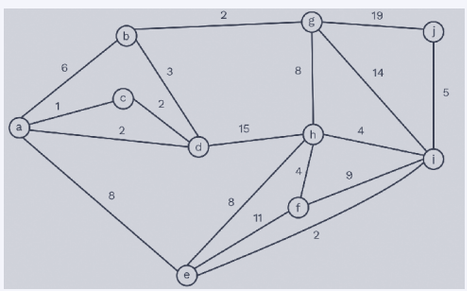
[2009]

(A) (b,e) (e,f) (a,c) (b,c) (f,g) (c,d)
(B) (b,e) (e,f) (a,c) (f,g) (b,c) (c,d)
(C) (b,e) (a,c) (e,f) (b,c) (f,g) (c,d)
(D) (b,e) (e,f) (b,c) (a,c) (f,g) (c,d)

Solution: (D)

Previous Years' Question

What is the weight ofa minimum spanning tree of the following graph?



(A) 29 (B) 31
(C) 38 (D) 41

Solution: (B)

Prim's algorithm:

PRIM_MST(G, w, r)

1. for each u ∈ G.V
2. u. key = ∞
3. u.π = NIL
4. r.key = 0
5. Q = G.V
6. while Q is not empty
7. u = Edge with minimum weight in Q
8. for each v ∈ G.Adj[u]
9. if v ∈ Q and w(u,v) < v.key
10. v.π = u
11. v.key = w(u,v)

- Prim's algorithm always assumes that the edges in set A form a single tree.
- The tree starts with a vertex v and grows till it spans all the vertices in the given Graph.
- At each step, a new edge is added to Tree A, which is the lightest among all the edges that are connected to the vertices in Tree A. This edge results in connecting an isolated vertex that is not included in Tree A.
- Prim's algorithm comes under the greedy algorithm since it selects a vertex that adds the minimum weight possible to the resulting minimum spanning tree.
- A connected graph G and a root r to be grown as a minimum spanning tree are the inputs given to the pseudocode above.
- The minimum priority queue Q contains all the vertices that are not in Tree A, based on key attributes.
- For a vertex v, v.key is the value of the minimum weighted edge among the edges connecting v to any other vertex in the tree. v.key=-∞, if no such edge exists.
- The attribute v.π names the parent of v in the tree
- Algorithm maintains the set A

$$A = \{(v, v \cdot \pi) : v \in V - \{r\} - Q\}$$

- When the algorithm terminates, the min-priority queue Q is empty
- The minimum spanning tree A for G is thus

$$A = \{(v, v \cdot \pi) : v \in V - \{r\}\}$$

- Lines 1-5 in pseudocode, make the key attribute of each vertex initialised to ∞ except the root. Since the root is the first vertex to be processed, its key is initialised to 0.

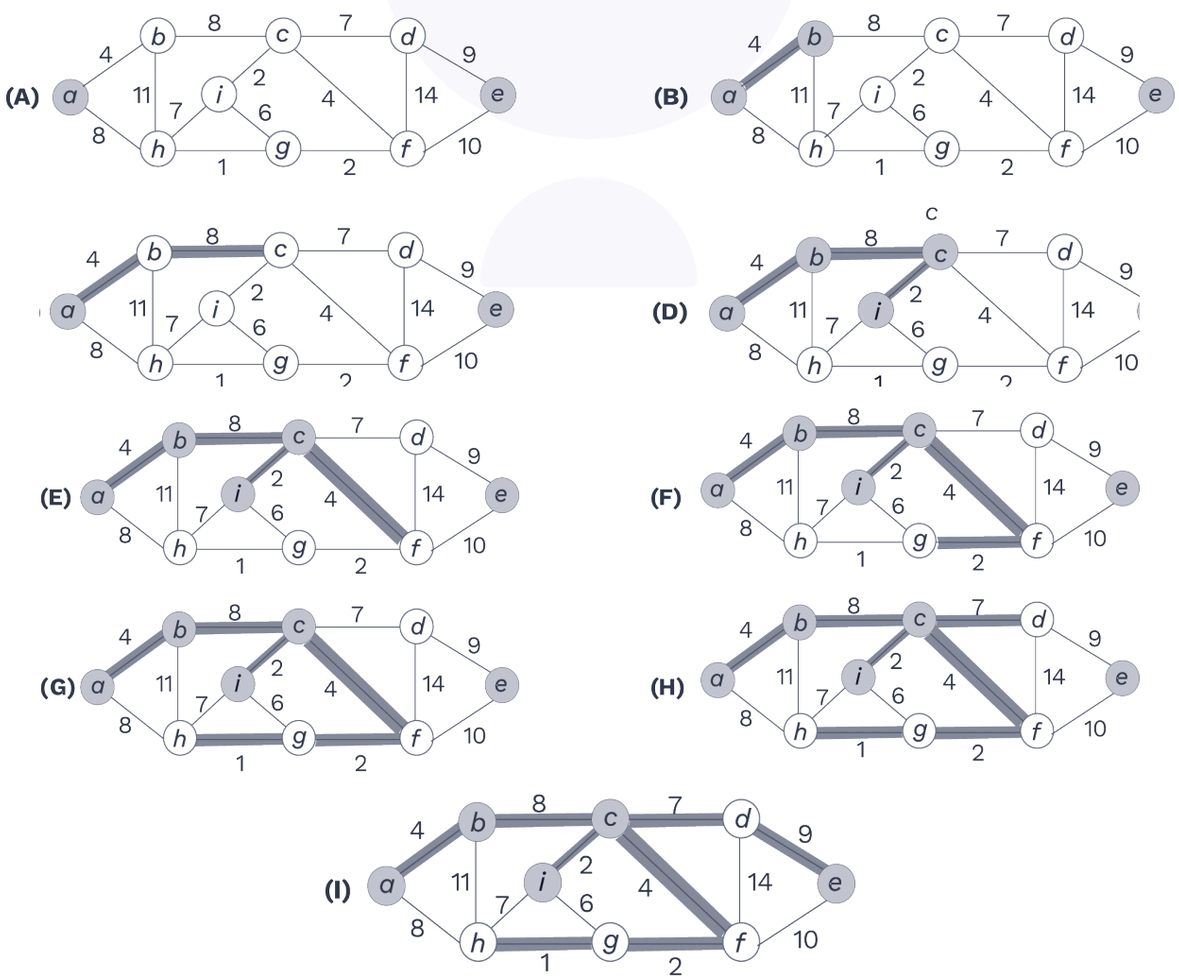
- Initialise each vertex's parent to NIL.
- Initialise the minimum priority queue Q with the set of all vertices in the graph.
- 7th Line finds the vertex u ∈ Q such that it is one of the vertices at the end of the light-weight edge that crosses the cut (V-Q, Q), except in the first iteration, in which u = r because of the 4th line.
- Deleting u from the set Q adds it to the set V-Q vertices in the tree, resulting in the addition of (u, u.π) to A.
- The for loop of the 8th to 11th lines updates the key and π attributes of each vertex v adjacent to vertex u but not the key and π attributes in the tree.

Running time of prim's algorithm:

- If we implement Q as a binary min-heap, BUILD-MIN-HEAP procedure to perform lines 1-5 in O(v) time.
- EXTRACT_MIN operation takes O(logV) time and is repeated for |V| times in the while loop.Therefore the total time is O(VlogV).
- The for loop in lines 8-11 executes O(E) times altogether since the sum of the lengths of all adjacency lists in 2|E|
- Within the for loop, line 9 takes constant time.
- The assignment in line 11 involves an implicit DECREASE-KEY operation on the min-heap which is O(logv) time
- Thus, the total time for Prim's algorithms is with vertices 'V' and edges 'E' O(vlogv + Elogv) = O(Elogv)

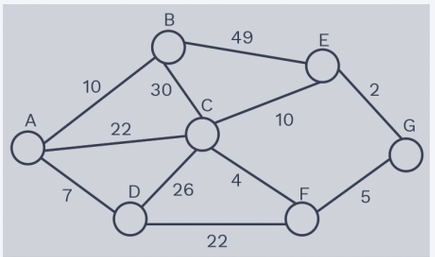
Note:
The running time of prim's algorithm by using the fibonacci heap is O(ElogV).

Example:



Previous Years' Question

Consider the undirected graph below:



Using Prim's algorithm to construct a minimum spanning tree starting with node A, which one of the following sequences of edges represents a possible order in which the edges would be added to construct the minimum spanning tree? **[2004]**

(A) (E, G), (C, F), (F, G), (A, D), (A, B), (A, C)
(B) (A, D), (A, B), (A, C), (C, F), (G, E), (F, G)
(C) (A, B), (A, D), (D, F), (F, G), (G, E), (F, C)
(D) (A, D), (A, B), (D, F), (F, C), (F, G), (G, E)

Solution: (D)

Single Source Shortest Path

- Let G = (V, E) be a weighted digraph, with real-valued weight W assigned to every edge E.
- The weight w(p) of path P = $<v_0, v_1, v.....,v_k>$ is the sum of the weights of its constituent edges

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

- The shortest-path weight $\delta(u,v)$ from u to v

$$\delta(u,v) = \begin{cases} \min\{w(p): u \overset{p}{\rightsquigarrow} v\}, & \text{if there is a path from u to v,} \\ \infty, & \text{otherwise} \end{cases}$$

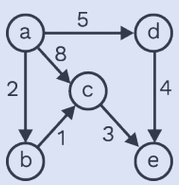
- A shortest path from vertex u to vertex v is then defined as path P with weight w(P) = $\delta(u,v)$.

The problem:

Let G (V,E) be the given digraph with positive edge weights. There exists a source vertex (distinguished from other vertices) S∈V from which the shortest path to every other vertex in the Graph needs to be found out.

Note:

Any subpath of a shortest path must also be a shortest path
eg:



δ(a,e) = 6 i.e(a,b,c,e)
δ(a,e) is a shortest path

Relaxation:

- For each vertex v ∈ V, maintain an attribute v.d which is an upper bound on the weight of the shortest path from source s to v, called as the shortest-path estimate.
- Initialise the estimates of the shortest path and predecessors by the following takes θ(v) time.

INITIALISE-SINGLE-SOURCE (G,S)

1. For each vertex v ∈ G.V
2. v.d = ∞
3. v.π = NIL
4. s.d = 0
   a) After initialisation, we have v.π = NIL for all v ∈ V, s.d = 0, and v.d = ∞ for
   b) v ∈ V – {s}
   c) An edge (u,v) is said to be relaxed after testing given that particular edge can improve the shortest path found so far. If yes, then v.d and v.π are updated.
   d) The following code performs a relaxation step on edge (u,v) in O(1) time

RELAX (u, v, w)

1. if v.d > u.d + w (u,v)
2. v.d = u.d + w(u,v)
3. v.π = u

Dijkstra's algorithm:

- Dijkstra's algorithm is a single source shortest path algorithm.
- It maintains a set of vertices S apart from the set of all vertices V, that keeps track of vertices to which the shortest path is found.
- The algorithm selects a vertex u ∈ V-S with the minimum path estimate, adds the vertex u to S and relaxes all the edges leaving from u.

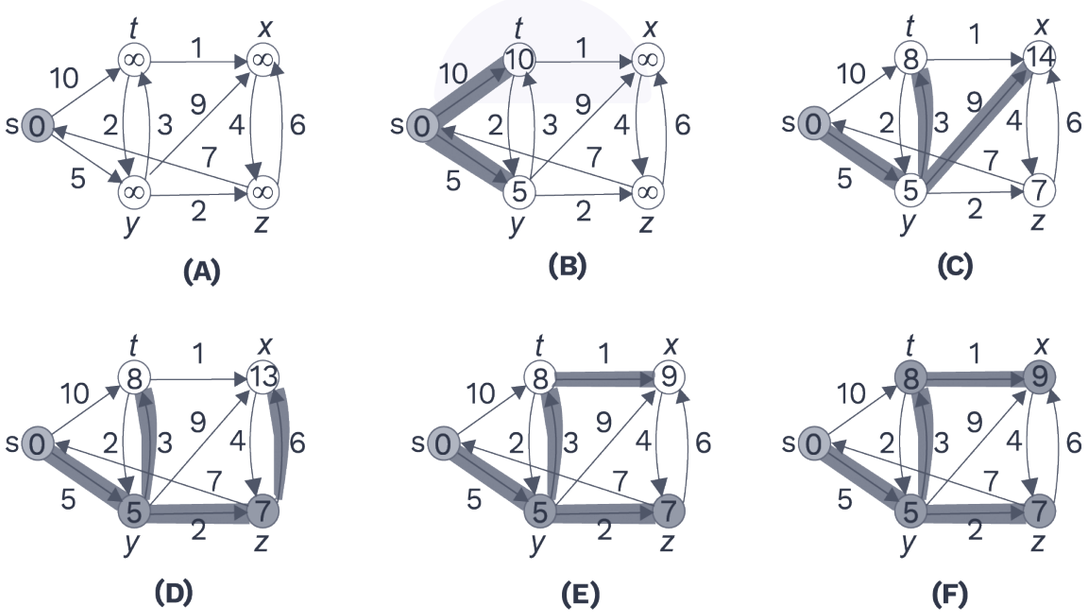
DIJKSTRA (G, w, s)

1. Initialise the source s
2. S = ϕ
3. Q = G.V
4. While Q ≠ ϕ
5. u = EXTRACT-MIN(Q)
6. S = S U {u}
7. For each vertex v ∈ G.adj[u]
8. RELAX (u, v, w)
   a) Because Dijkstra's algorithms always choose the "lightest" or "closest" vertex in V-S to add to set S, it falls into a greedy strategy.
   b) Min-priority queue Q of vertices, keyed by their d values, is used in the above algorithm.

Analysis of Dijkstra's algorithm:

- The initialisation uses only O(n) time,
- Each vertex is processed exactly once so condition in line 4 and EXTRACT-MIN are called exactly once for every vertex, i.e. V times in total.
- The inner loop for each V∈Adj[u] is called once for each edge in the graph.
- Each call of the inner loop does O(1) work plus, possibly, one decrease-key operation.
- Recalling that all the priority queue operations require O(log|Q|) = O(logV) time
- Thus, we have
  = O(V+V+VlogV +ElogV)
  = O((V+E) logV)

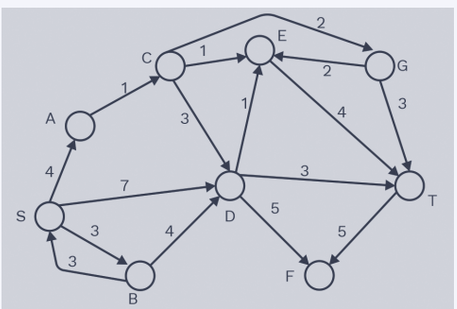
Example:



Previous Years' Question

Consider the directed graph shown in figure below. There are multiple shortest paths between vertices S and T. Which one will be reported by Dijkstra's shortest path algorithm? Assume that, in any iteration, the shortest path to a vertex v is updated only when a strictly shorter path to vis discovered.

[2012]



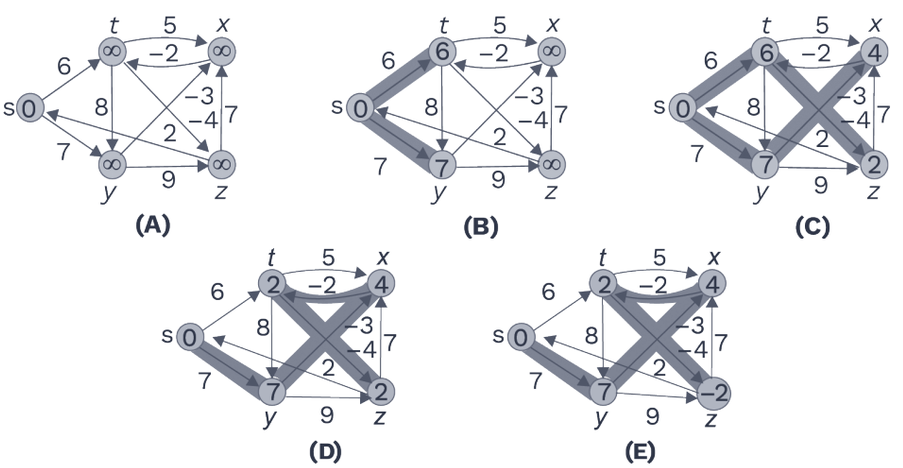
(A) SDT (B) SBDT
(C) SACDT (D) SAC ET

Solution: (D)

The Bellman-Ford algorithm:

- The Bellman-Ford algorithm is a single-source shortest path algorithm. It works on digraphs with negatively weighted edges, also.
- The Bellman-Ford algorithm results in a boolean value indicating whether a negative weight cycle is present or not in the graph.

Example:



- If there is no negative weighted cycle, then the algorithm produces the shortest path weights.
- Bellman–Ford algorithm assumes that a vertex v can be reached from vertex u with at most |V|-1 edges which result, in no negative edge cycle.
- It finds the shortest path involving 1 edge at first, then two and so on |V| -1 edges.
- If the cost found in involving |V| passes and |V|-1 passes are the same, then there is no negative edge cycle present.
- The edges are relaxed repeatedly, decreasing the estimate v.d on the weight of the shortest path to all the vertices from the source s until the actual shortest-path weight $\delta(s,v)$ is achieved.

BELLMAN-FORD (G, w, s)

1. INITIALISE_SINGLE_SOURCE(G, s)
2. For i = 1 to (number of vertices) - 1
3. For every edge (u, v) $\in$ G.E
4. RELAX (u, v, w)
5. For every edge (u, v) $\in$ G.E (Edges of graph)
6. If v.d > u.d + w (u, v)
7. Return FALSE
8. Return TRUE

- The Bellman-ford algorithm runs in time O(VE), since the initialisation in line 1 takes $\theta(v)$ time, each of the |V|-1 passes over the edges in lines 2-4 takes $\theta(E)$ time, and the for loop of lines 5-7 takes O(E) time.

Previous Years' Question

What is the time complexity of Bellman-Ford single-source shortest path algorithm on a complete graph of n vertices?

(A) $0(n^2)$ (B) $0(n^2 \log n)$ (C) $0(n^3)$ (D) $0(n^3 \log n)$

Solution: (C)

Solved Examples

1. Consider the weights and values of items listed below.

Item number	Weight (in kgs)	Value (in Rupees)
1	1	2
2	4	28
3	5	25
4	3	18
5	3	9

The task is to pick a subset of these items by using greedy method such that their total weight is no more than 15 kgs and their total values is maximized.
The total value of items picked by the greedy algorithms is denoted by V_{greedy}"
The value of V_{greedy} is ------

Solution: 80

Now, sort the object in non-increasing order of value/weight ratio.

Item No.	Weight (in kg)	Value (in rupees)	Value/ weight
2	4	28	7
4	3	18	6
3	5	25	5
5	3	9	3
1	1	2	2

First we pick item 2 then item 4 then item 3 and at last item 5. Total capacity of this 4 object is 15 kg and total value is 80. So, overall profit is 80.

2. Consider the weights and values of items listed below.

Item number	Weight (in kgs)	Value (in Rupees)
1	4	8
2	8	4

The task is to pick a subset of these items by using greedy method such that their total weight is no more than 4 kgs and their total value is maximized.
The total value of items picked by the greedy algorithm is denoted by V_{greedy}.
The value of V_{greedy} is ----

Solution: 8

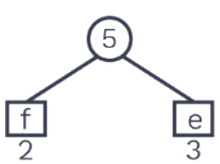
Here, the value / weight ratio of both the item is 2. So, we can take any item 1st.

Let's take item 1 first. Hence, the total capacity is going to be 4, and the profit is 8.

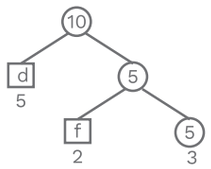
3. The characters a to f have the set of frequencies as shown below:
a:40, b:30, c:20, d:5, e:3, f:2
A Huffman code is used to represent the characters.
What is the weighted external path length?

Solution: 205

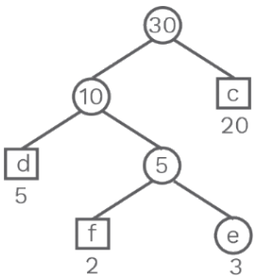
Step 1: $\overset{a}{40}, \overset{b}{30}, \overset{c}{20}, \overset{d}{5}, \overset{e}{3}, \overset{f}{2}$



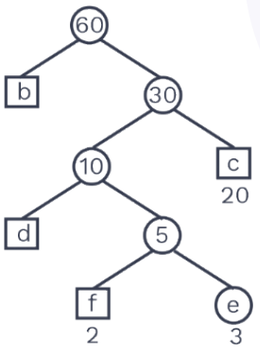
Step 2: a b c d
40, 30, 20, 5, 5



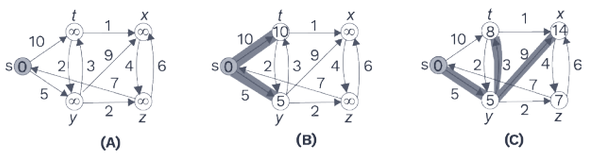
Step 3: a b c
40, 30, 20, 10



Step 4: a b
40, 30, 30



Step 5:



Weighted external path length = 1 × 40 + 30 × 2 + 4 × 5 + 20 × 3 + 2 × 5 + 3 × 5

= 40 + 60 + 20 + 60 + 10 + 15

= 205

4. The characters a to h have the set of frequencies given below:
a:2, b:2, c:3, d:4, e:6, f:9, g:14, h:22
A Huffman code is used to represent the characters.
What is the sequence of characters corresponding to the following code?
11000001011011000100101001 1

(A) hdegfcab

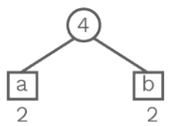
(B) abcdefgh

(C) hdegcfab

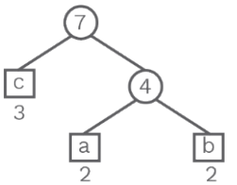
(D) hdgefcab

Solution: (A)

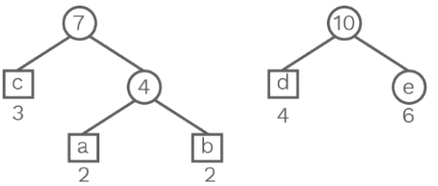
a b c d e f g h
2 2 3 4 6 9 14 22



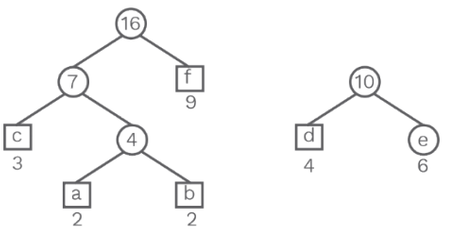
c d e f g h
4 3 4 6 9 14 22



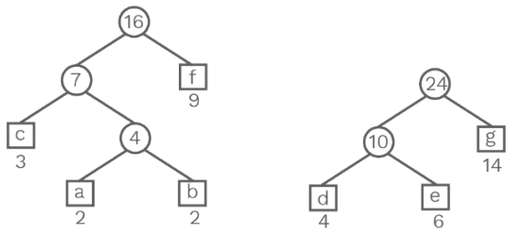
d e f g h
7 4 6 9 14 22



f g h
7 10 9 14 22

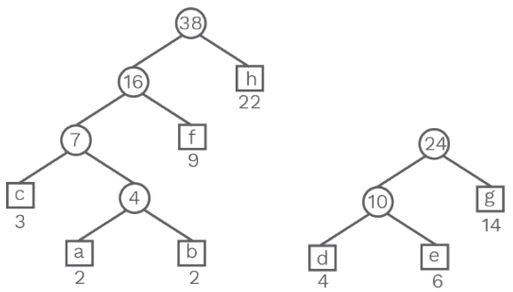


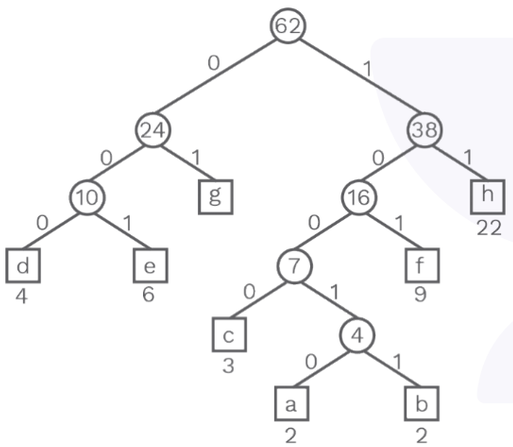
g h
10 16 14 22



h

16 24 22





So, the Huffman code is

d: 000
e: 001
g: 01
h: 11
f: 101
c: 1000
a: 10010
b: 10011

Group the string into characters from right to left:

11 000 001 01 101 1000 10010 10011

h d e g f c a b

5. If job J = (J_1, J_2 , J_3 , J_4) are given their processing time T_i = (1, 1, 1, 1) profit p; = (6, 8, 5, 10) and deadline are D_i = (2, 1, 1, 2). Maximum how many job can be done?

(A) 1

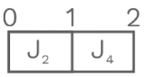
(B) 2

(C) 3

(D) All

Solution: (B)

The gantt chart is shown below:



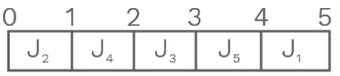
Profit = 8 + 10 = 18

6. If job J = (J_1, J_2 , J_3 , J_4 , J_5, J_6) are given their processing time T; = (1, 1, 1, 1, 1, 1) profit p, = (200, 180, 190, 300, 120, 100) and deadline are D, = (5, 3, 3, 2, 4, 2). What is the maximum profit?

Solution: 990

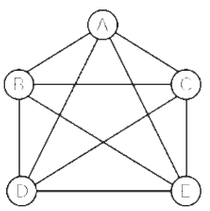
	J_1	J_2	J_3	J_4	J_5	J_6
Deadline	5	3	3	2	4	2
Profit	200	180	190	300	120	100

The gantt chart is shown below:



Profit = 990

7. What is the number of spanning tree possible from the given graph?



Solution: 125

Since it is a complete graph. So, the number of spanning trees for a complete graph k_n is $n^{(n-2)}$, (where n = no. of vertices)

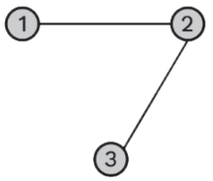
Here, n = 5

So, the number of spanning trees = $5^{(5-2)}$

= $5^{(3)}$

= 125

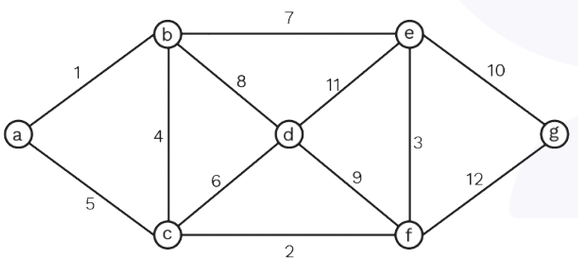
8. What is the number of spanning tree possible from the given graph?



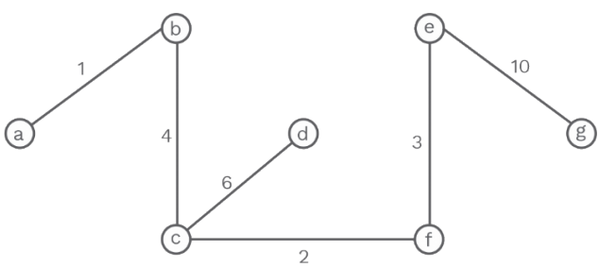
Solution: 1

The given graph is itself a spanning tree. So, only 1 spanning tree in possible.

9. Consider the undirected graph given below. What is the minimum possible weight of a spanning tree T ?



Solution: 26



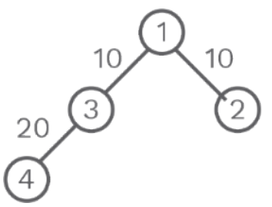
This is a minimum cost spanning tree.

10. Consider a complete undirected graph with vertex set {1, 2, 3, 4}. Entry W;i in the matrix W below is the weight of the edge {i,j}. What is the minimum possible weight of a spanning tree T?

$$W = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 10 & 10 & 50 \\ 10 & 0 & 40 & 30 \\ 10 & 40 & 0 & 20 \\ 50 & 30 & 20 & 0 \end{bmatrix} \end{array}$$

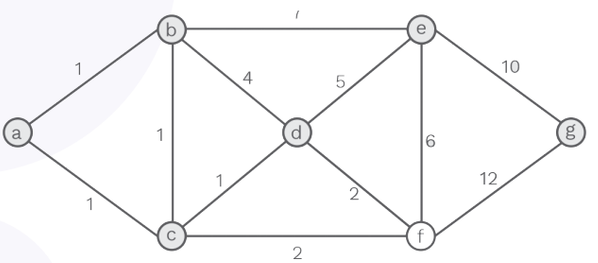
Solution: 40

The minimum spanning tree in tree is shown below:



Minimum cost of spanning tree = 40.

11. Consider the undirected graph below:



Using Prim's algorithms to construct a minimum spanning tree starting with node A, which one of the following sequences of node A, which one of the following sequences of edges represents a possible order in which the edges would be added to construct the minimum spanning tree? (MSQ)

(A) (a, b), (a, c), (c, d), (c, f), (d, e), (e, g)

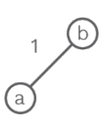
(B) (a, c), (a, b), (c, d), (c, f), (d, e), (e, g)

(C) (a, b), (a, c), (c, d), (d, f), (d, e), (e, g)

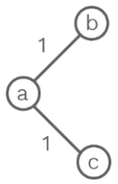
(D) (a, b), (a, c), (b, c), (d, f), (d, e), (e, g)

Solution: (A), (B) & (C)

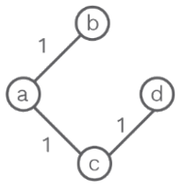
Step 1: We can either connect (a, b) or (a, c). We are first connecting ab.



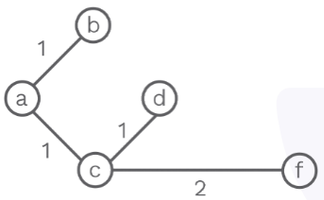
Step 2: Connecting (a, c) as shown below.



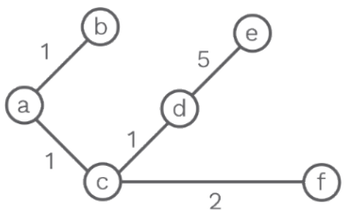
Step 3: Connect (c, d)



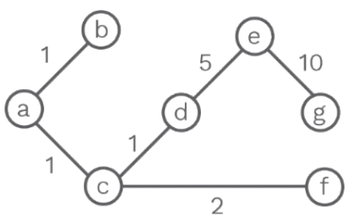
Step 4: Connect (c, f)



Step 5: Connect (d, e)

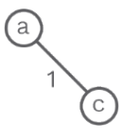


Step 6: Connect (e, g)

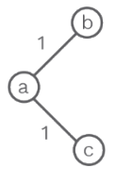


In this graph, we can get another sequences of edges as well to construct a minimum spanning tree, as shown below:

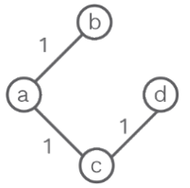
Step 1: First connect (a, c)



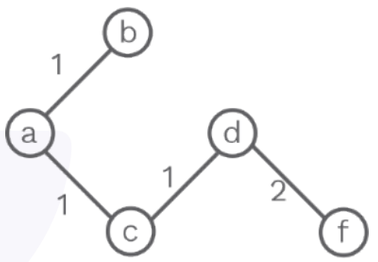
Step 2: Connect (a, b)



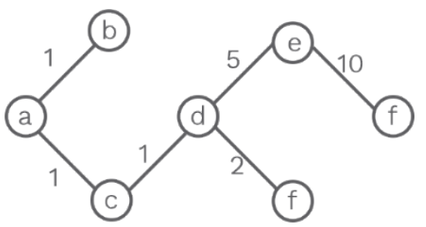
Step 3: Connect (c, d)



Step 4: Connect (d, f)



Step 5: Connect (d, e)



Similarly, we will get another sequence as

(a, b), (b, c), (c, d), (d, f), (d, e), (e, g).

12. What is the time complexity of Prim's algorithm with and without min heap respectively, (where E = edge and V = vertices)

(A) $O(ElogV^2)$ and $O(V2)$

(B) $O(V^2)$ and $O(V^3)$

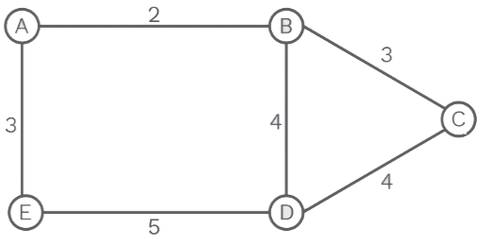
(C) $O(ElogV)$ and $O(V^3)$

(D) $O(VlogV + E)$ and $O(V^2)$

Solution: (A)

The time complexity of Prim's algorithm with and without min-heap is $O(ElogV)$ and $O(V^2)$.

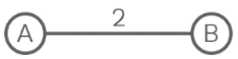
13. The number of distinct minimum spanning trees for the weighted graph below is ---



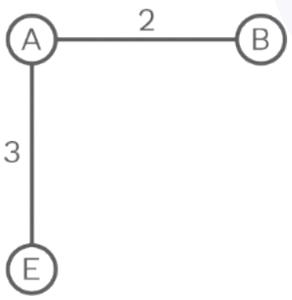
Solution: 2

By using Kruskal's algorithm we can add first (A, B) as show below:

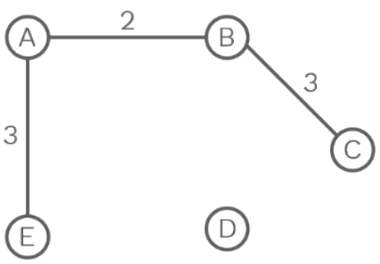
Step 1:



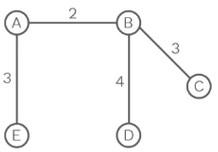
Step 2: Now add either (B, C) or (A, E). We are adding (A, E).



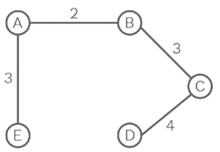
Step 3: Now, add (B, C)



Step 4: Now, we can add either (B, D) or (D, E). So, we will get two minimum spanning trees as shown below:



Or



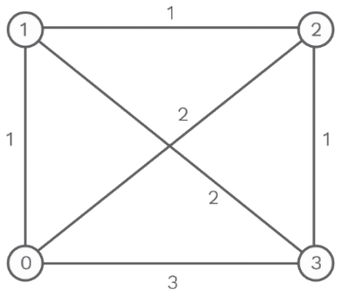
14. A complete, undirected, weighted graph 'G' is given on the vertex {O, 1, ... , n-1} for any fixed 'n'. Draw the minimum spanning tree of G if

(A) The weight of the edge (u, v) is lu - vi

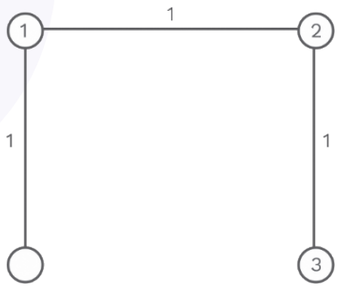
(B) The weight of the edge (u, v) is (u + v)

Solution: (A)

Let's take n = 4. So, the complete, undirected and weighted graph is shown below.

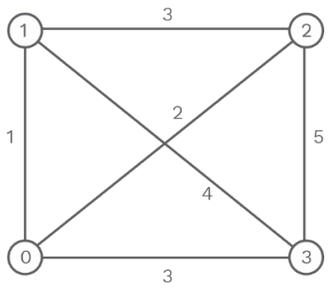


The minimum spanning tree is:

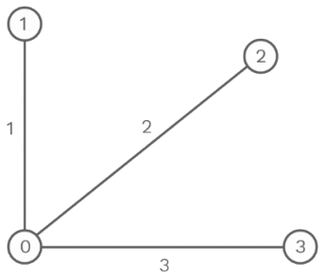


So, this is a line-graph. In this case, we will always get line graph.

(b) Similarly, for n = 4

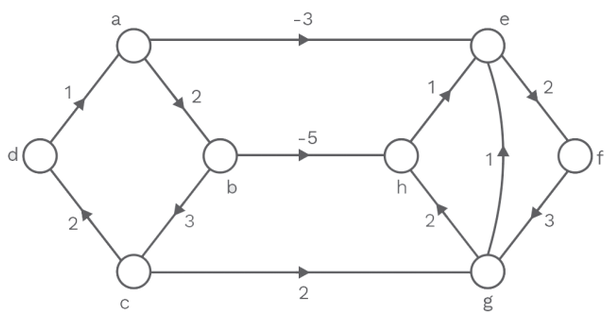


The minimum spanning tree for this graph is



This type of graph is called a star graph. Here, we will always get a star graph as a minimum spanning tree.

15. Find the single source shortest path node a to all other vertices. The graph is given below:



Solution:

	a	b	c	d	e	f	g	h
a	⓪	∞	∞	∞	∞	∞	∞	∞
e		2	∞	∞	(-3)	∞	∞	∞
f		2	∞	∞		(-1)	∞	∞
g		2	∞	∞			(2)	∞
b		(2)	∞	∞				4
h			(5)	∞				(-3)
c				(5)				
d								

The order in which single source shorted path computed is

a → e → f → g → b → h → c → d

Chapter Summary

Huffman Coding:

- Huffman's greedy algorithm builds an optimum manner of expressing each character as a binary string using a table that shows how often each character appears i.e., its frequency.

Applications of Huffman Coding:

- They're used by compression formats like gzip, and other.
- It is useful when there's a string of characters that appear frequently.

Fractional knapsack problem:

- Given n items worth value V; each and weight W; select items whose weight sums to a given value is w. The items can be taken in fractions.

Job sequencing problem:

- Given a list of jobs, each with its own deadline and associated profit if completed before the deadline.

Minimum spanning trees:

- MST is a subset of the set of Edges E in a connected and undirected graph G(V, E) such that it forms an acyclic graph.
- Difference between Prim's Algorithm and Kruskal's algorithm.

Prim's algorithm	Kruskal's algorithm
• It begins to construct the minimum spanning tree from any vertex in the graph. • The time complexity of Prim's algorithm is $O(V^2)$ where V is the number of vertices, and it can be improved to O(ElogV) using Fibonacci heaps. • Prim's algorithm returns a connected component and only works with connected graphs. • In dense graph, Prim's algorithm performs better	• The minimum spanning tree is built from the vertex in the graph with the least edge connected to it. • The time complexity of Kruskal's algorithm is O(ElogV), where V is the number of vertices • Kruskal's algorithm can both generate forests and work on disconnected components at any time • In sparse graphs, Kruskal's algorithm performs better.

Single-source shortest path:

- It is a problem of finding shortest path from a source to all the vertices in a graph.
- Difference between Dijkstra's algorithm and Bellman Ford's algorithm.

Prim's algorithm	Kruskal's algorithm
• When there is a negative weight edge, Dijkstra's Algorithms does not work properly all the time. • The time complexity is O(ElogV). • It follows greedy approach, performs better	• When there is a negative weight edge, Bellman Ford's algorithm detects the negative weight cycle. • ts time complexity is O(VE). • It follows dynamic programming approach.