ALGORITHM AND ANALYSIS Q/A'S
UNIT -2

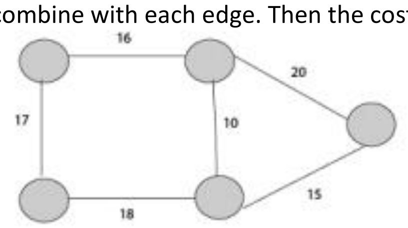
1. What is a minimum spanning tree and explain its properties and applications?

Answer:

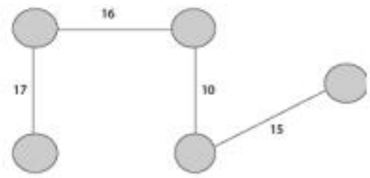
Spanning Tree: Given an undirected and connected graph, a spanning tree of the graph is a tree that spans (that is, it includes every vertex of) and is a sub-graph of (every edge in the tree belongs to)

Minimum Spanning Tree:

Minimum Spanning Tree is a Spanning Tree which has minimum total cost. If we have a linked undirected graph with a weight (or cost) combine with each edge. Then the cost of spanning tree would be the sum of the cost of its edges.



Connected , Undirected Graph



Minimum Cost Spanning Tree
Total Cost = 17+16+10+15=58

Properties of Minimal Spanning Tree:

1. There may be several minimum spanning trees of the same weight having the minimum number of edges.
2. If all the edge weights of a given graph are the same, then every spanning tree of that graph is minimum.
3. If each edge has a distinct weight, then there will be only one, unique minimum spanning tree.
4. A connected graph G can have more than one spanning trees.
5. A disconnected graph can't have to span the tree, or it can't span all the vertices.
6. Spanning Tree does not contain cycles.
7. Spanning Tree has (n-1) edges where n is the number of vertices.

Addition of even one single edge results in the spanning tree losing its property of Acyclicity and elimination of one single edge results in its losing the property of connectivity.

Application of Minimum Spanning Tree:

1. Consider n stations are to be linked using a communication network and laying of communication links between any two stations involves a cost. The ideal solution would be to extract a sub-graph termed as minimum cost spanning tree.
2. Suppose we want to construct highways or railroads spanning several cities then we can use the concept of minimum spanning trees.
3. Designing Local Area Networks.
4. Laying pipelines connecting offshore drilling sites, refineries and consumer markets.
5. Suppose we want to apply a set of houses with
   - Electric Power
   - Water
   - Telephone lines
   - Sewage lines

   To reduce cost, we can connect houses with minimum cost spanning trees.
6. Minimum spanning tree has direct application in the design of networks. It is used in algorithms approximating the travelling salesman problem, multi-terminal minimum cut problem and minimum-cost weighted perfect matching. Other practical applications are:
   - Cluster Analysis
   - Handwriting recognition
   - Image segmentation

~~~~~~

2. Discuss Edmond's Blossom algorithm with example?

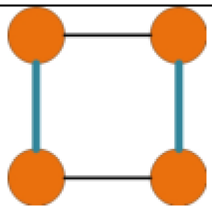
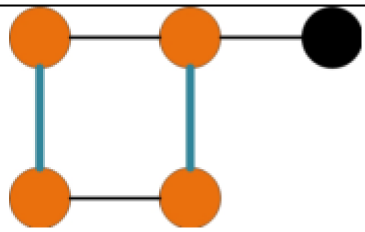
Answer:

EDMOND'S BLOSSOM ALGORITHM:

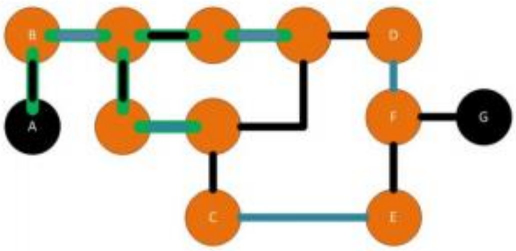
The blossom algorithm, created by Jack Edmunds in 1961, was the first polynomial time algorithm that could produce a maximum matching on any graph. Previous to the discovery of this algorithm, there were no fast algorithms that were able to
~~~~~~

find a maximum matching on a graph with odd length cycles. In contrast to some other matching algorithms, the graph need not be bipartite. Edmonds's Blossom Algorithm uses and extends the essential ideas of the Hopcroft-Karp algorithm, which computes a maximum matching for bipartite graphs.
This algorithm uses the ideas of finding augmenting and alternating paths in a graph. An augmenting path is a path along within a graph where the edges alternate between unmatched and matched edges and it ultimately ends with an unmatched edge.

ALGORITHM:
The algorithm works by repeatedly trying to find augmenting paths (specifically paths starting and ending with free vertices) in the graph until no more augmenting paths can be found. This occurs when either all vertices are matched with a mate or when a free vertex is only connected to match vertices which recursively only match to other matched vertices.

All Vertices Matched and a maximum matching	Not All Vertices Matched and a maximum matching
	

Furthermore, the process of finding augmented paths can be slightly modified so that it simultaneously can check to see if there is a blossom in the graph. This can be accomplished by labelling each vertex in an alternating manner O (for odd) and E (for even) depending on how far away a vertex is from the root. To start, label the original free vertex E, and then alternate labeling from there.
The algorithm that finds augmented paths will always search for neighbouring vertices from a vertex labelled E (the matched vertex's mate always is two edges further away from the root) and thus if an adjacent vertex also has the label E, there is a blossom that contains both of those vertices. This blossom is then contracted into one supernode with a label E and then the search for augmenting paths continues.
All together, this algorithm starts by finding augmenting paths within any graph while labeling vertices E or O and then inverting these paths to increase the size of the matching by one. If a blossom is found, it is contracted into a single node and the augmenting path algorithm continues as normal. This repeats until all vertices of the graph have been either searched or are already part of the matching set.



Blossoms can be nested within each other, and therefore, supernodes may end up containing other supernodes.

FOR DETAILED ALGORITM CHECK THE PDF

~~~~~~

3. Write an algorithm to compute maximal weight maximum independent set in graph?

Answer:

Maximal Weight Maximum Independent Set In Graph: [refer sona notes]

The maximum (weighted) independent set (MIS(MWIS)) is one of the most important optimization problems. In several heuristic methods for optimization problems, the greedy strategy is the most natural and simplest one. For MIS, two simple greedy algorithms have been investigated. One is called GMIN, which selects a vertex of minimum degree, removes it and its neighbors from the graph and iterates this process on the remaining graph until no vertex remains. (the set of selected vertices is an
~~~~~~

independent set). The other is called GMAX, which deletes a vertex of maximum degree until no edge remains (the set of remaining vertices is an independent set).

Divide and Conquer:

Divide and Conquer is an algorithmic paradigm. A typical Divide and Conquer algorithm solves a problem using following three steps.

1. **Divide:** Break the given problem into sub-problems of same type.
2. **Conquer:** Recursively solve these sub-problems.
3. **Combine:** Appropriately combine the answers.

Divide and Conquer algorithm:

```
DAC(a, i, j)
 {
if(small(a, i, j))
 return(Solution(a, i, j))
else
    m = divide(a, i, j)                    // f1(n)
    b = DAC(a, i, mid)                     // T(n/2)
    c = DAC(a, mid+1, j)                   // T(n/2)
   d = combine(b, c)                       // f2(n)
  return(d)
}
```

Divide/Break:

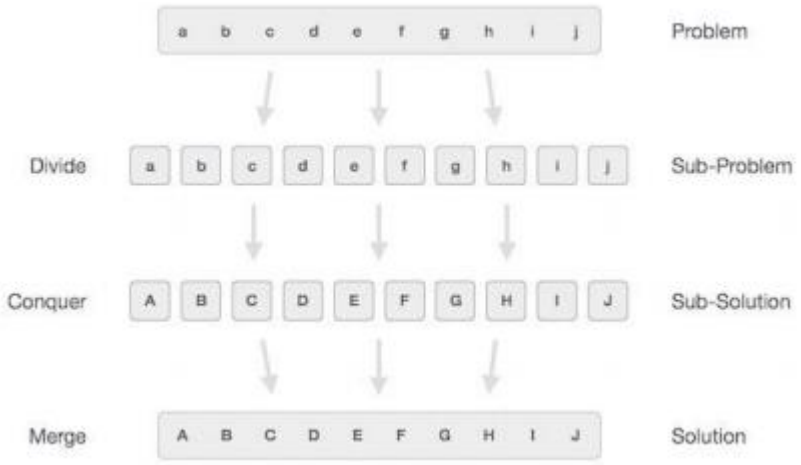
This step involves breaking the problem into smaller sub-problems. Sub-problems should represent a part of the original problem. This step generally takes a recursive approach to divide the problem until no sub-problem is further divisible. At this stage, sub-problems become atomic in nature but still represent some part of the actual problem.

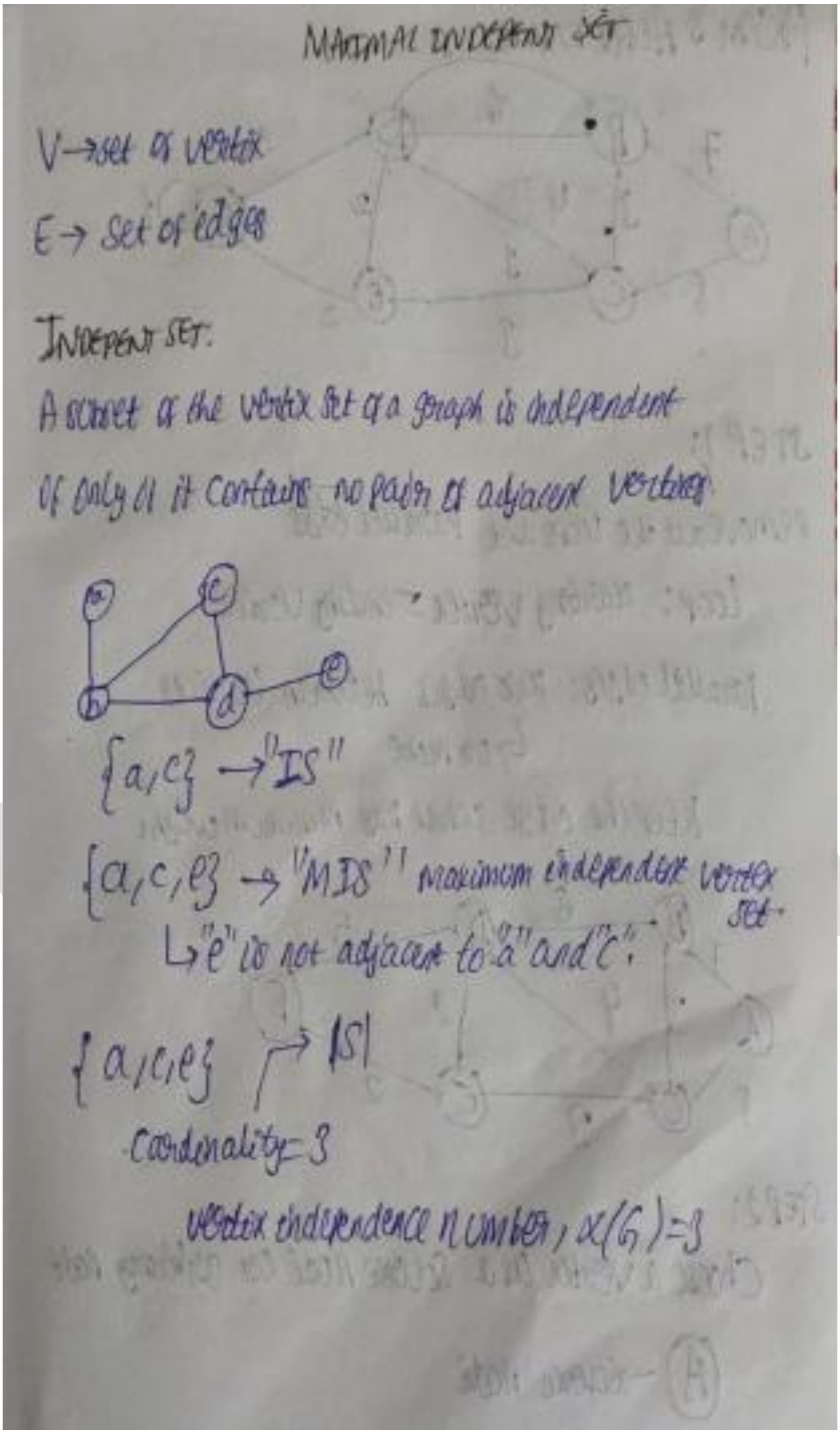
Conquer/Solve:

This step receives a lot of smaller sub-problems to be solved. Generally, at this level, the problems are considered 'solved' on their own.

Merge/Combine:

When the smaller sub-problems are solved, this stage recursively combines them until they formulate a solution of the original problem. This algorithmic approach works recursively conquers and merge steps works so close that they appear as one. Below diagram indicates this:





~~~~~~

4. Discuss in detail Prims algorithm with an example?

Answer:

Spanning-tree is a set of edges forming a tree and connecting all nodes in a graph. The **minimum spanning tree** is the spanning tree with the lowest cost (sum of edge weights). Also, it's worth noting that since it's a tree, MST is a term used when talking about undirected connected graphs.

Prim's Algorithm:

It is a greedy algorithm. It starts with an empty spanning tree. The idea is to maintain two sets of vertices:
~~~~~~

- Contain vertices already included in MST.
- Contain vertices not yet included.

At every step, it considers all the edges and picks the minimum weight edge. After picking the edge, it moves the other endpoint of edge to set containing MST.

Algorithm:

1. Create MST set that keeps track of vertices already included in MST.
2. Assign key values to all vertices in the input graph. Initialize all key values as INFINITE (∞). Assign key values like 0 for the first vertex so that it is picked first.
3. While MST set doesn't include all vertices.
    a. Pick vertex u which is not is MST set and has minimum key value. Include 'u' to MST set.
    b. Update the key value of all adjacent vertices of u. To update, iterate through all adjacent vertices. For every adjacent vertex v, if the weight of edge u.v less than the previous key value of v, update key value as a weight of u.v.

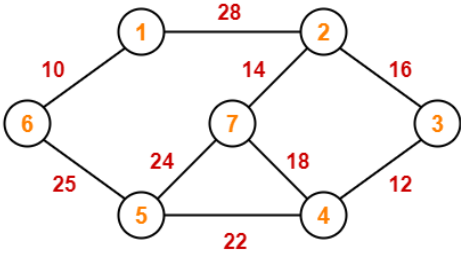
4.2. Algorithm

Consider the following pseudocode for Prim's algorithm.

Algorithm 2: Prim's Algorithm

```
Data: G: The given graph
      source: The node to start from
Result: Returns the cost of the MST
totalCost ← 0;
included ← {false };
Q.addOrUpdate(source, 0, Φ);
while ¬ Q.empty() do
    u ← Q.getNodeWithLowestWeight();
    totalCost ← totalCost + u.weight;
    if u.edge ≠ Φ then
        mst.add(u.edge);
    end
    included[u.node] ← true;
    for v ∈ G.neighbors(u.node) do
        if ¬included[v.node] then
            Q.addOrUpdate(v.node, weight(u.node, v.node),
              v.edge);
        end
    end
end
return totalCost, mst;
```

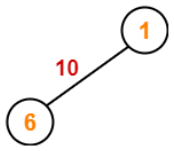
Example: Construct the minimum spanning tree (MST) for the given graph using Prim's Algorithm-



Solution-

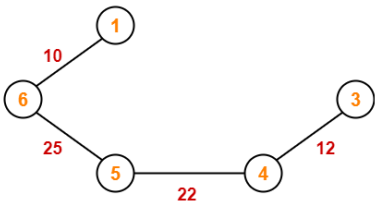
The above discussed steps are followed to find the minimum cost spanning tree using Prim's Algorithm-

Step-01:

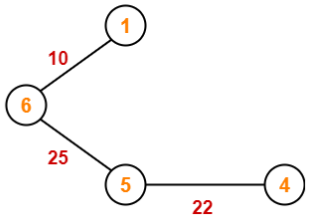


STEP-02:

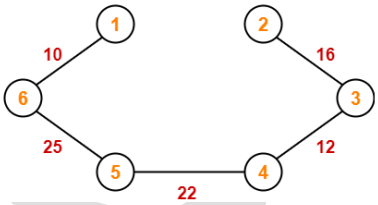
Step-04:



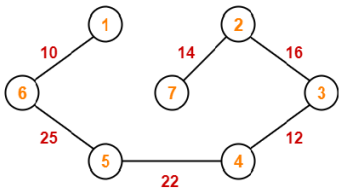
Step-03:



Step-05:



Step-06:



Since all the vertices have been included in the MST, so we stop.

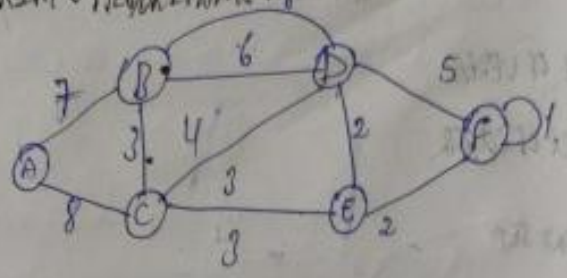
Now, Cost of Minimum Spanning Tree

= Sum of all edge weights

= 10 + 25 + 22 + 12 + 16 + 14

= 99 units

PRIM'S ALGORITHM:



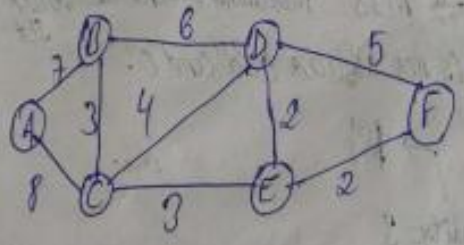
STEP 1:

Remove all the loops and Parallel edges

Loop: starting vertex = ending vertex

parallel edges: two edges between vertices (2 or more)

keep the edge which has minimum weight



STEP 2:

Choose a vertex as a source node or arbitrary node

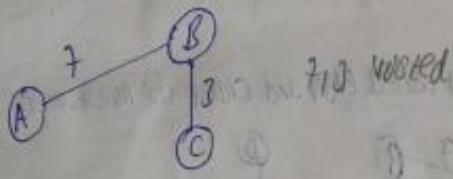
(A) → source node

Check the out going edges (incident edges).

Choose the edge from A which has less weight

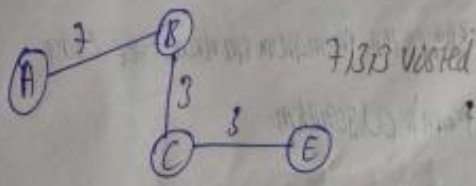
STEP 3:

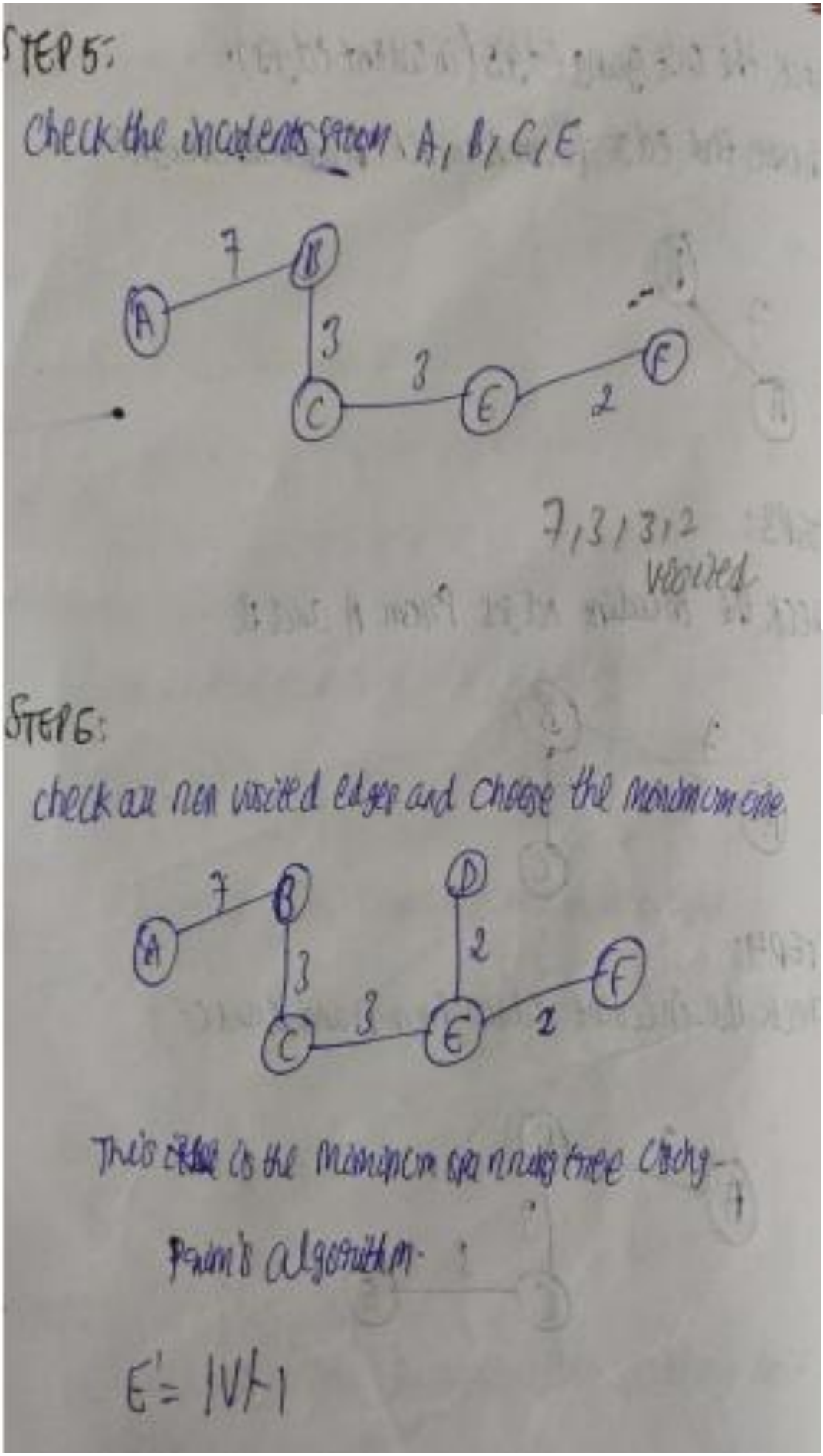
check the incident edges from A and B:



STEP 4:

check the incident edges from A and B and C:





The advantage of Prim's algorithm:

is its complexity, which is better than Kruskal's algorithm. Therefore, Prim's algorithm is helpful when dealing with dense graphs

that have lots of edges.

However, Prim's algorithm doesn't allow us much control over the chosen edges when multiple edges with the same weight occur. The reason is that only the edges discovered so far are stored inside the queue, rather than all the edges like in Kruskal's algorithm.

The disadvantage of Prim's algorithm:

Also, unlike Kruskal's algorithm, Prim's algorithm is a little harder to implement.

The Application of Prim's algorithm:

Prim's Algorithm can be used in lots of different applications. For example, the traveling salesman problem. Another example is to build a computer network system

The Comparision of Prim's algorithm[extra info]:

Let's highlight some key differences between the two algorithms.

	Kruskal	Prim
Multiple MSTs	Offers a good control over the resulting MST	Controlling the MST might be a little harder
Implementation	Easier to implement	Harder to implement
Requirements	Disjoint set	Priority queue
Time Complexity	$O(E \cdot log(V))$	$O(E + V \cdot log(V))$

As we can see, **the Kruskal algorithm is better to use regarding the easier implementation and the best control over the resulting MST**. However, **Prim's algorithm offers better complexity**.

Reference: https://www.baeldung.com/cs/kruskals-vs-prims-algorithm#:~:text=The%20advantage%20of%20Prim's%20algorithm,with%20the%20same%20weight%20occur

~~~~~~

5. Discuss in detail Kruskal's algorithm with an example?

Answer:

Spanning-tree is a set of edges forming a tree and connecting all nodes in a graph. The **minimum spanning tree** is the spanning tree with the lowest cost (sum of edge weights). Also, it's worth noting that since it's a tree, MST is a term used when talking about undirected connected graphs.

Kruskal's Algorithm:

This is an algorithm to construct a Minimum Spanning Tree for a connected weighted graph. It is a Greedy Algorithm. If the graph is not linked, then it finds a Minimum Spanning Tree.

Kruskal's Algorithm Implementation-

The implementation of Kruskal's Algorithm is explained in the following steps-

Step-01:

- Sort all the edges from low weight to high weight.

Step-02:

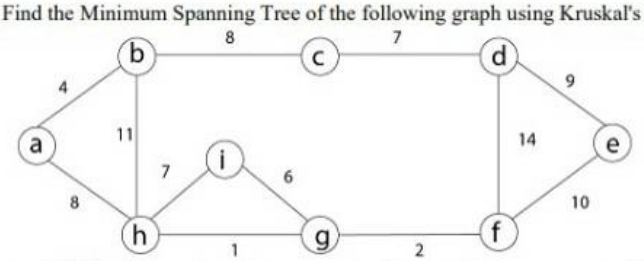
- Take the edge with the lowest weight and use it to connect the vertices of graph.
- If adding an edge creates a cycle, then reject that edge and go for the next least weight edge.

Step-03:

- Keep adding edges until all the vertices are connected and a Minimum Spanning Tree (MST) is obtained.

PROBLEM BASED ON KRUSKAL'S ALGORITHM:
~~~~~~

For Example: Find the Minimum Spanning Tree of the following graph using Kruskal's algorithm.



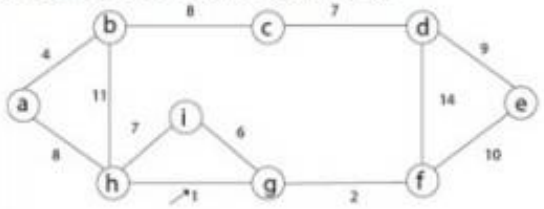
Solution: First we initialize the set A to the empty set and create |v| trees, one containing each vertex with MAKE-SET procedure. Then sort the edges in E into order by non-decreasing weight.
There are 9 vertices and 12 edges. So MST formed (9-1) = 8 edges

Weight	Source	Destination
1	H	g
2	G	F
4	A	B
6	I	G
7	H	I
7	C	D
8	B	C
8	A	H
9	D	E
10	E	F
11	B	H
14	D	F

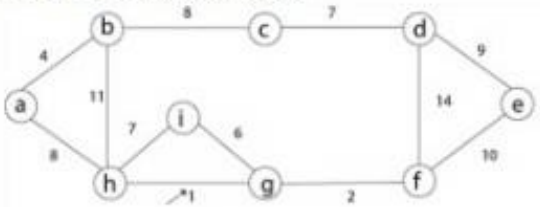
Now, check for each edge (u, v) whether the endpoints u and v belong to the same tree. If they do then the edge (u, v) cannot be supplementary. Otherwise, the two vertices belong to different trees, and the edge (u, v) is added to A and the vertices in two trees are merged in by union procedure.

Steps to find Minimum Spanning Tree using Kruskal's algorithm

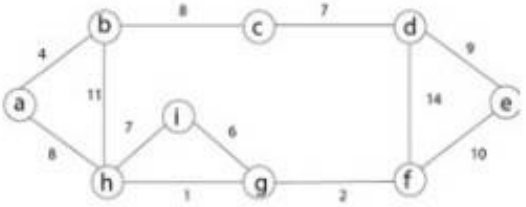
Step1: So, first take (h, g) edge



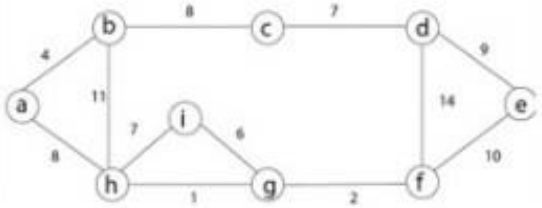
Step 2: then (g, f) edge.



Step 3: then (a, b) and (i, g) edges are considered, and the forest becomes

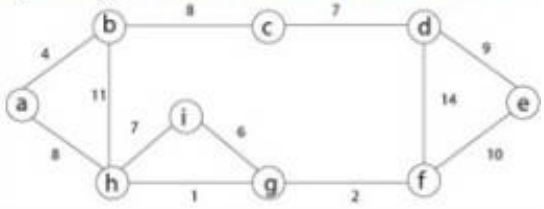


Step 4: Now, edge (h, i). Both h and i vertices are in the same set. Thus it creates a cycle. So this edge is discarded.
Then edge (c, d), (b, c), (a, h), (d, e), (e, f) are considered, and the forest becomes.



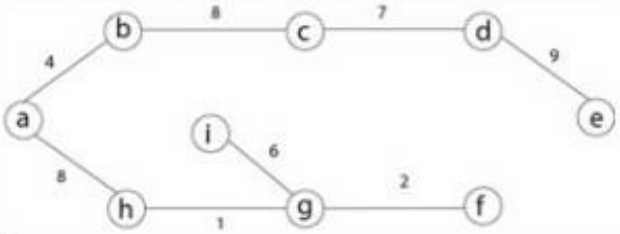
Step 5: In (e, f) edge both endpoints e and f exist in the same tree so discarded this edge. Then (b, h) edge, it also creates a cycle.

Step 6: After that edge (d, f) and the final spanning tree is shown as in dark lines.



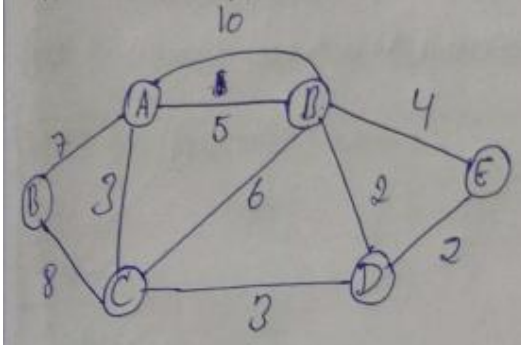
Step 7: This step will be required Minimum Spanning Tree because it contains all the 9 vertices and (9 - 1) = 8 edges

e → f, b → h, d → f [cycle will be formed]



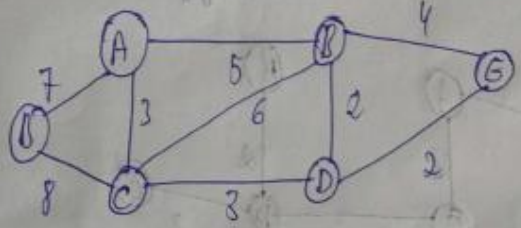
Minimum Cost MST

KRUSKAL'S ALGORITHM:



STEP1:

Remove loops and edges.



STEP 2:

Arrange all the edges according to edge weight in increasing order.

BD=2

DE=2

AC=3

CD=3

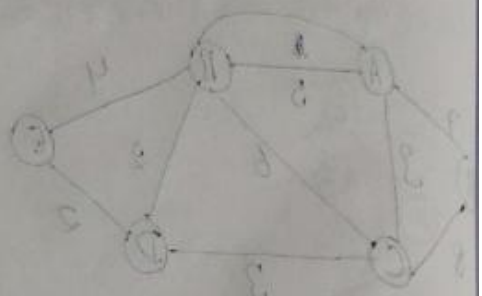
BE=4

AB=5

BC=6

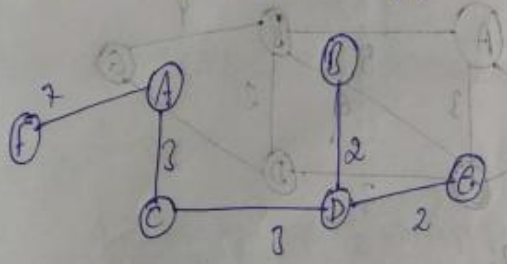
AF=7

FC=8



STEP 3:

Choose the minimum edge from above edges.



This is the minimum Spanning tree.

The Comparision of Kruskal's algorithm[extra info]:

Let's highlight some key differences between the two algorithms.

	Kruskal	Prim
Multiple MSTs	Offers a good control over the resulting MST	Controlling the MST might be a little harder
Implementation	Easier to implement	Harder to implement
Requirements	Disjoint set	Priority queue
Time Complexity	$O(E \cdot log(V))$	$O(E + V \cdot log(V))$

As we can see, **the Kruskal algorithm is better to use regarding the easier implementation and the best control over the resulting MST**. However, **Prim's algorithm offers better complexity**.

~~~~~~

6. Explain the components, advantages and disadvantages of greedy approach?

Answer:

Greedy is an algorithmic paradigm that builds up a solution piece by piece, always choosing the next piece that offers the most obvious and immediate benefit. So the problems where choosing locally optimal also leads to global solution are best fit for Greedy. An algorithm is designed to achieve optimum solution for a given problem. In greedy algorithm approach, decisions are made from the given solution domain. As being greedy, the closest solution that seems to provide an optimum solution is chosen. Greedy algorithms try to find a localized optimum solution, which may eventually lead to globally optimized solutions. However, generally greedy algorithms do not provide globally optimized solutions.

Components of Greedy Approach:

Greedy approach has the following five components:

- A candidate set − A solution is created from this set.
- A selection function − Used to choose the best candidate to be added to the solution.
- A feasibility function − Used to determine whether a candidate can be used to contribute to the solution.
- An objective function − Used to assign a value to a solution or a partial solution.
- A solution function − Used to indicate whether a complete solution has been reached

Advantages and Disadvantages of Greedy Approach:

Advantages And Disadvantages

Advantages:

- They are easier to implement.
- They require much less computing resources.
- They are much faster to execute.
- Greedy algorithms are used to solve optimization problems

Disadvantages:

- Their only disadvantage being that they not always reach the global optimum solution.
- On the other hand, even when the global optimum solution is not reached, most of the times the reached sub-optimal solution is a very good solution.

Examples of Greedy Algorithm

Most networking algorithms use the greedy approach. Here is a list of few of them −

- Travelling Salesman Problem
- Prim's Minimal Spanning Tree Algorithm
- Kruskal's Minimal Spanning Tree Algorithm
- Dijkstra's Minimal Spanning Tree Algorithm
~~~~~~

- Graph - Map Coloring
- Graph - Vertex Cover
- Knapsack Problem
- Job Scheduling Problem

There are lots of similar problems that uses the greedy approach to find an optimum solution.

Reference: https://www.tutorialspoint.com/data_structures_algorithms/greedy_algorithms.htm
~~~~~~

7. Discuss the use of Graph matching algorithm to find vertex cover of a graph?

Answer:

The matching algo is in page 2 &
theorems are there write that

Reference: https://www.cs.dartmouth.edu/~ac/Teach/CS105-Winter05/Handouts/vempala-blossom.pdf
~~~~~~