Problem Solving Strategies and Computational Approaches SCS1304

Handout 6 : Greedy Algorithms

Prof Prasad Wimalaratne PhD(Salford),SMIEEE

Ref

- **Prims and Kruskals Algorithms - Greedy Method**
- https://www.youtube.com/watch?v=4ZlRH0eK-qQ
- **Greedy Method - Introduction**
- https://www.youtube.com/watch?v=ARvQcqJ_-NY&list=PLfFeAJ-vQopt_S5XlayyvDFL_mi2pGJE3
- **Prim's Algorithm for Minimum Spanning Tree**
- https://www.youtube.com/watch?v=ZtZaR7EcI5Y

Greedy Algorithms

https://miro.medium.com/v2/resize:fit:640/format:webp/1*mKvyFosEdJxv5oZpRMIQOw.gif

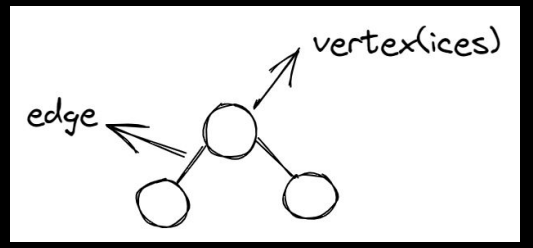
History of Greedy algorithm

- The concept of a greedy algorithm has a long history in various fields, going back in some form to ancient times.
- Although the term "greedy algorithm" may not have been used explicitly, the principles of making local optimal choices have been applied to problem-solving for centuries.
- Ancient Mathematics and Coin Tossing: The earliest known application of the greedy approach is the coin-tossing problem. Ancient civilizations needed to exchange goods and services, and their challenge was to make the change with as few coins as possible.
  - This problem can be solved by a simple greedy strategy of always choosing the largest currency denomination that is not greater than the remaining convertible amount.

```
While W > 0
   pick the largest coin c that is <= W
   W <- W - c
```

Greedy Algorithms?

- According to the Oxford English Dictionary, "greedy" means having excessive desire for something without considering the effect or damage done.
- In computer science, a greedy algorithm is an algorithm that finds a solution to problems in the shortest time possible. It picks the path that seems optimal at the moment without regard for the overall optimization of the solution that would be formed.
- Edsger Dijkstra, a computer scientist and mathematician who wanted to calculate a minimum spanning tree, introduced the term "Greedy algorithm". Prim and Kruskal came up with optimization techniques for minimizing cost of graphs.
- Many Greedy algorithms were developed to solve graph problems. A graph is a structure made up of edges and vertices.



https://www.freecodecamp.org/news/greedy-algorithms/

Greedy algorithms

- Greedy algorithms are often used when solving optimization problems, like finding the maximum or the minimum of a certain quantity, under certain conditions.
- Solutions that satisfy those extrema are called optimal solutions
- Greedy algorithms are attractive as a way to get approximate solutions, as they are usually simple to implement and fast, and they sometimes get good solutions (can be proved are within some reasonable factor of the optimal solution).

Greedy algorithms

- A greedy algorithm is an algorithm that follows the problem-solving heuristic of making the locally optimal choice at each stage.
  - Heuristics are used to make informed but biased decisions when information and time are lacking. Overreliance on heuristics in some real-world situations can potentially lead to serious consequences.

- In many problems, a greedy strategy does not produce an optimal solution, but a greedy heuristic can yield locally optimal solutions that approximate a globally optimal solution in a reasonable amount of time.

Coin Change Problem

- For example, with C={1,2,5} and W=13, you will pick 5, 5, 2 and 1 , and you can show that the minimum number of coins required is indeed 4 .

- However, Greedy algorithm does NOT always provide an optimal solution. For example, if C={1,4,5} and W=8 , the greedy algorithm will choose coins 5,1,1,1 when there is a solution 4,4 using less coins.

Further Example: Coin Change Problem

- Let us say you have a set of coins with values {1, 2, 5, 10, 20, 50, 100} and you need to give minimum number of coin to someone change for 36 .
- The greedy algorithm for making change would work as follows:
  - Start with the largest coin value that is less than or equal to the amount to be changed. In this case, the largest coin less than 36 is 20 .
  - Subtract the largest coin value from the amount to be changed, and add the coin to the solution. In this case, subtracting 20 from 36 gives 16 , and we add a 20 coin to the solution.
  - Repeat steps 1 and 2 until the amount to be changed becomes 0.
  - So, using the greedy algorithm, the solution for making change for 36 would be one 20 coins, one 10 coin, one 5 coins and one 1 coin needed.

https://www.geeksforgeeks.org/introduction-to-greedy-algorithm-data-structures-and-algorithm-tutorials/

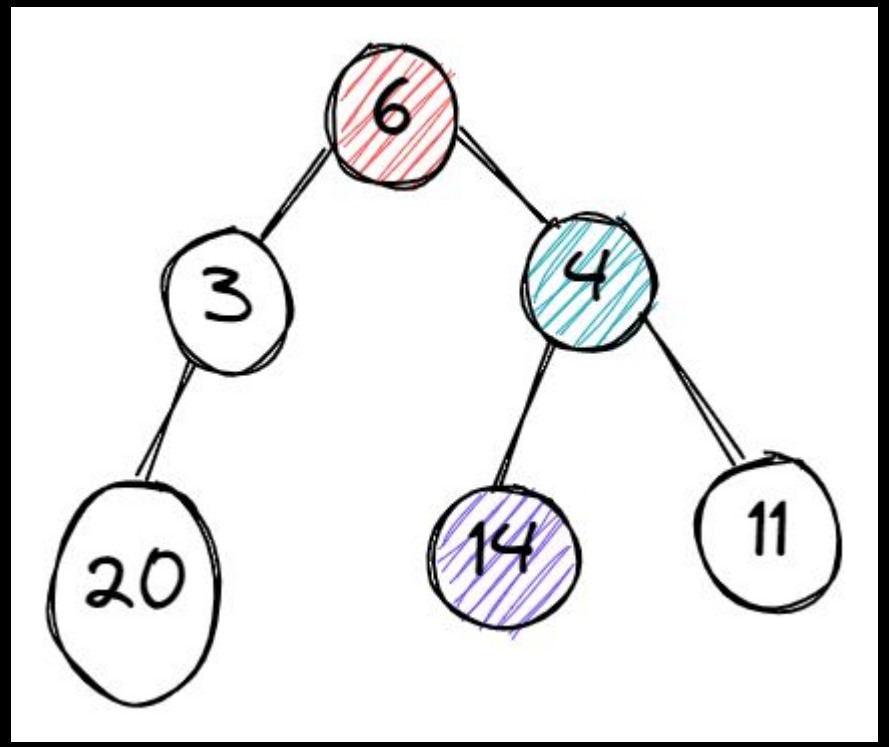
Greedy vs Not Greedy Algorithms

- An algorithm is greedy when the path picked is regarded as the best option based on a specific criterion without considering future consequences. But it typically evaluates feasibility before making a final decision. The correctness of the solution depends on the problem and criteria used.
- Example: A graph has various weights and you are to determine the maximum value in the tree. You would start by searching each node and checking its weight to see if it is the largest value.
- There are two approaches to solving this problem: greedy approach or not greedy.

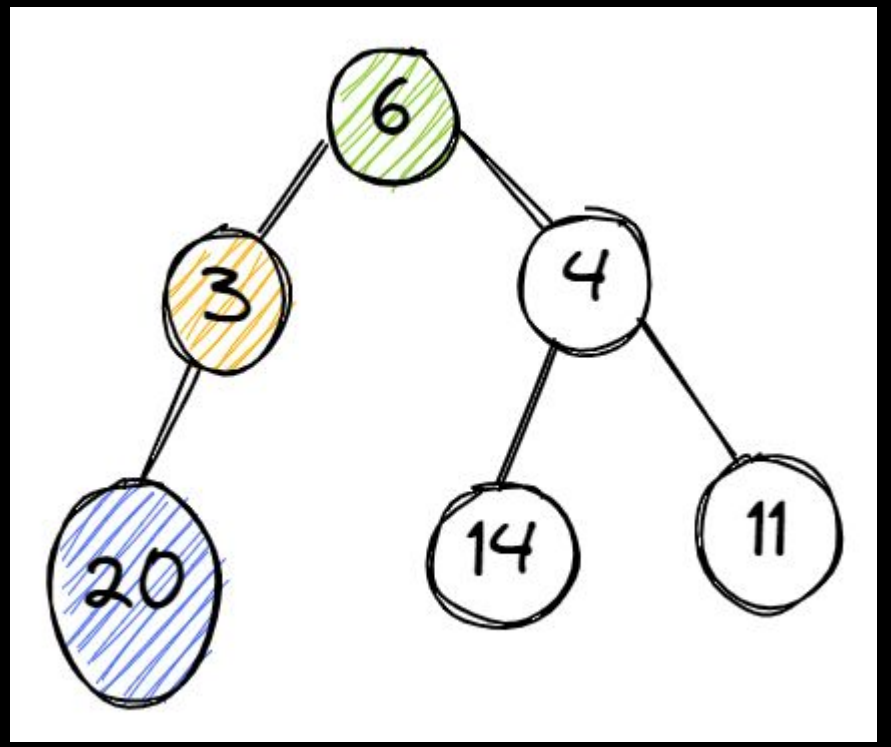
https://www.freecodecamp.org/news/greedy-algorithms/

Greedy vs Not Greedy Algorithms

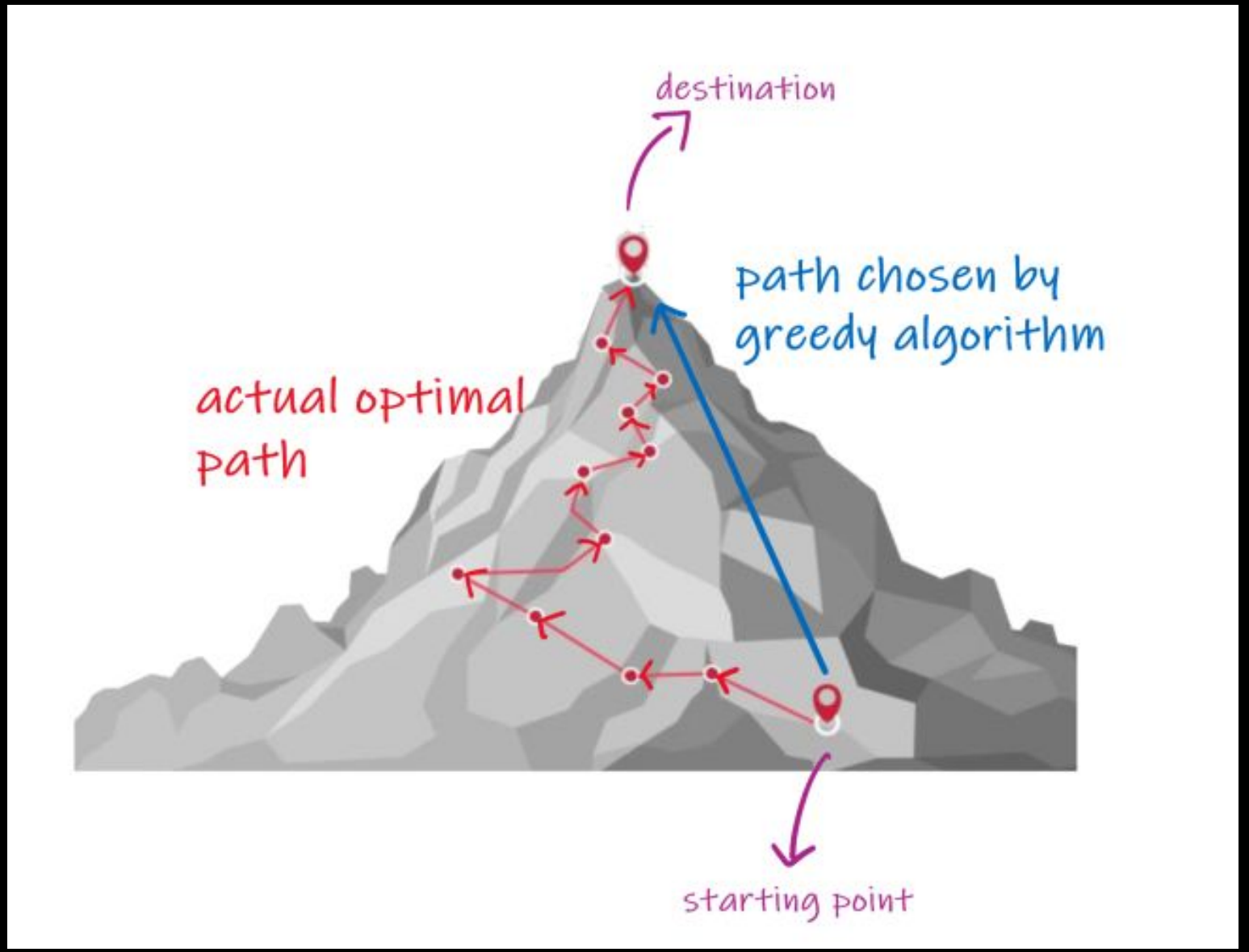
- Greedy Algo



Not Greedy Algo



https://www.freecodecamp.org/news/greedy-algorithms/



https://medium.com/@benkaddourmed54/is-it-necessary-to-use-the-greedy-algorithm-19b97843d60c

Characteristics of a Greedy Algorithm

- The algorithm solves its problem by finding an optimal solution.
- This solution can be a maximum or minimum value. It makes choices based on the best option available.
- The algorithm is fast and efficient with time complexity of O(n log n) or O(n). Therefore applied in solving large-scale problems.
- The search for optimal solution is done without repetition – the algorithm runs once.
- It is straightforward and easy to implement.

https://www.freecodecamp.org/news/greedy-algorithms/

Greedy Algorithm Examples: Activity Selection Problem

- This problem contains a set of activities or tasks that need to be completed.
- Each one has a start and finish time.
- The algorithm finds the maximum number of activities that can be done in a given time without them overlapping.

Starting Time	Finish Time	Task
2	5	Homework
6	10	Presentation
4	8	Term Paper
10	12	Volleyball practice
13	14	Biology lecture
7	15	Hangout

https://www.freecodecamp.org/news/greedy-algorithms/

Activity Selection Problem : Approach to the Problem

We have a list of activities. Each has a start time and finish time.

1. First, sort the activities and start time in ascending order using the finish time of each.
2. Then we start by picking the first activity. We create a new list to store the selected activity.
3. To choose the next activity, we compare the finish time of the last activity to the start time of the next activity. If the start time of the next activity is greater than the finish time of the last activity, it can be selected. If not we skip this and check the next one.
4. This process is repeated until all activities are checked. The final solution is a list containing the activities that can be done

https://www.freecodecamp.org/news/greedy-algorithms/

Activity Selection Problem

Starting Time	Finish Time	Task
2	5	Homework
4	8	Term Paper
6	10	Presentation
10	12	Volleyball practice
13	14	Biology lecture
7	15	Hangout

- The first step is to sort the finish time in ascending order and arrange the activities with respect to the result.

final result is the list of selected activities that we can do without time overlapping: **{Homework, Presentation, Volleyball practice, Biology lecture}.**

https://www.freecodecamp.org/news/greedy-algorithms/

```
data = {
  "start_time": [2 , 6 , 4 , 10 , 13 , 7],
  "finish_time": [5 , 10 , 8 , 12 , 14 , 15],
  "activity": ["Homework" , "Presentation" , "Term paper" , "Volleyball practice" , "Biolog
}

selected_activity =[]
start_position = 0
# sorting the items in ascending order with respect to finish time
tem = 0
for i in range(0 , len(data['finish_time'])):
   for j in range(0 , len(data['finish_time'])):
       if data['finish_time'][i] < data['finish_time'][j]:
           tem = data['activity'][i] , data['finish_time'][i] , data['start_time'][i]
           data['activity'][i] , data['finish_time'][i] , data['start_time'][i] = data['act
           data['activity'][j] , data['finish_time'][j] , data['start_time'][j] = tem

# by default, the first activity is inserted in the list of activities to be selected.

selected_activity.append(data['activity'][start_position])
for pos in range(len(data['finish_time'])):
   if data['start_time'][pos] >= data['finish_time'][start_position]:
       selected_activity.append(data['activity'][pos])
       start_position = pos

print(f"The student can work on the following activities: {selected_activity}")
# Results
# The student can work on the following activities: ['Homework', 'Presentation', 'Volleybal
```

https://www.freecodecamp.org/news/greedy-algorithms/

What is Greedy Algorithm?

- A greedy algorithm is a problem-solving technique that makes the best local choice at each step in the hope of finding the global optimum solution.
- It prioritizes immediate benefits over long-term consequences, making decisions based on the current situation without considering future implications. While this approach can be efficient and straightforward, it doesn't guarantee the best overall outcome for all problems.

- However, it is important to note that not all problems are suitable for greedy algorithms. They work best when the problem exhibits the following properties:
  1. **Greedy Choice Property**: The optimal solution can be constructed by making the best local choice at each step.
  2. **Optimal Substructure**: The optimal solution to the problem contains the optimal solutions to its subproblems.

https://www.geeksforgeeks.org/introduction-to-greedy-algorithm-data-structures-and-algorithm-tutorials/

1. Greedy choice property

- Whichever choice seems best at a given moment can be made and then (recursively) solve the remaining sub-problems.
- The choice made by a greedy algorithm may depend on choices made so far, but not on future choices or all the solutions to the subproblem.
- It iteratively makes one greedy choice after another, reducing each given problem into a smaller one. In other words, a greedy algorithm never reconsiders its choices.
- This is the main difference from dynamic programming, which is exhaustive and is guaranteed to find the solution.
- After every stage, dynamic programming makes decisions based on all the decisions made in the previous stage and may reconsider the previous stage's algorithmic path to the solution

https://en.wikipedia.org/wiki/Greedy_algorithm

dynamic programming vs Greedy Approach

- Dynamic programming focus on the Principle of Optimality, "An optimal solution to any instance of an optimization problem is composed of optimal solutions to its sub-instances".

- Whereas, the greedy approach focuses on expanding partially constructed solutions until you arrive at a solution for a complete problem.
  - It must be "the best local choice among all feasible choices available on that step".

Greedy approach: design process

- Dynamic Programming vs. Greedy Approach
  - DP: Recursive property to divide an instance into smaller ones
  - GA: No division into smaller instances, but making a sequence of greedy choices to arrive at a solution.
- Selection procedure
  - Chooses the next item to add to the solution set.
  - Selection is made according to a greedy criterion that satisfies locally optimal consideration at the time.
- Feasibility check
  - Sees if the new set is feasible.
- Solution check
  - Determines if the new set constitutes a solution to the instance.

Characteristics of Greedy Algorithm

- Here are the characteristics of a greedy algorithm:
  1. Greedy algorithms are simple and easy to implement.
  2. They are efficient in terms of time complexity, often providing quick solutions.
  3. Greedy algorithms are used for optimization problems where a locally optimal choice leads to a globally optimal solution.
  4. These algorithms do not reconsider previous choices, as they make decisions based on current information without looking ahead.
  5. Greedy algorithms are suitable for problems for optimal substructure.

https://www.geeksforgeeks.org/introduction-to-greedy-algorithm-data-structures-and-algorithm-tutorials/

Examples of Greedy Algorithm

- Several well-known algorithms fall under the category of greedy algorithms. Here are a few examples:
  1. Dijkstra's Algorithm: This algorithm finds the shortest path between two nodes in a graph. It works by repeatedly choosing the shortest edge available from the current node.
  2. Kruskal's Algorithm: This algorithm finds the minimum spanning tree of a graph. It works by repeatedly choosing the edge with the minimum weight that does not create a cycle.
  3. Fractional Knapsack Problem: This problem involves selecting items with the highest value-to-weight ratio to fill a knapsack with a limited capacity. The greedy algorithm selects items in decreasing order of their value-to-weight ratio until the knapsack is full.
  4. Scheduling and Resource Allocation : The greedy algorithm can be used to schedule jobs or allocate resources in an efficient manner.
  5. Coin Change Problem : The greedy algorithm can be used to make change for a given amount with the minimum number of coins, by always choosing the coin with the highest value that is less than the remaining amount to be changed.
  6. Huffman Coding : The greedy algorithm can be used to generate a prefix-free code for data compression, by constructing a binary tree in a way that the frequency of each character is taken into consideration.

https://www.geeksforgeeks.org/introduction-to-greedy-algorithm-data-structures-and-algorithm-tutorials/

Knapsack Problem-

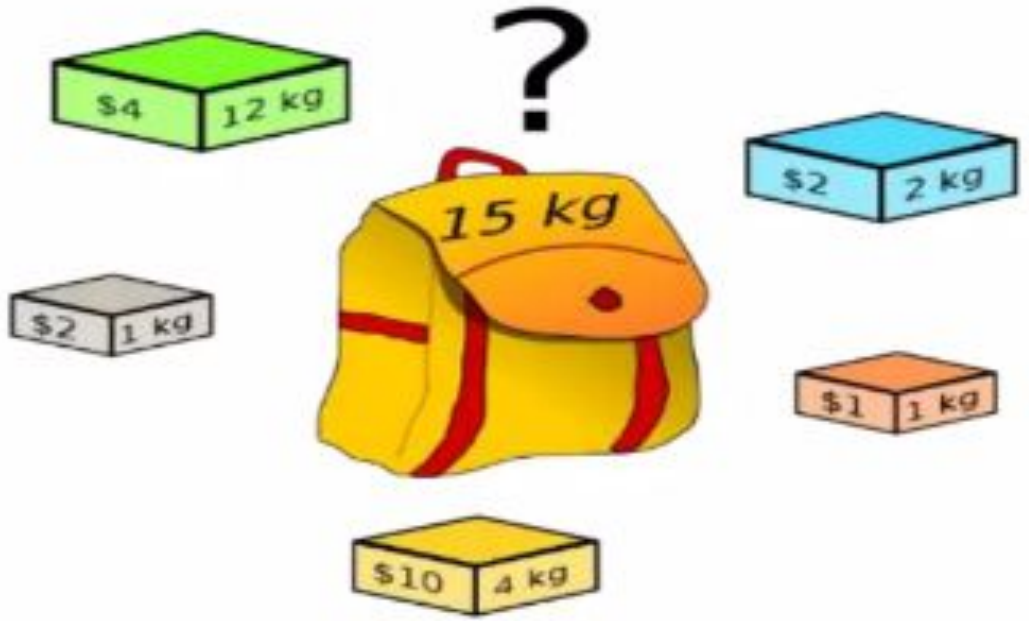
You are given the following-

- A knapsack (kind of shoulder bag) with limited weight capacity.
- Few items each having some weight and value.

The problem states-

Which items should be placed into the knapsack such that-

- **The value or profit obtained by putting the items into the knapsack is maximum.**
- **And the weight limit of the knapsack does not exceed.**



Why use Greedy Approach?

- Here are some reasons why you might use the Greedy Approach:
  i. Simple and easy to understand: The Greedy Approach is straightforward and easy to implement, making it a good choice for beginners.
  ii. Fast and efficient: It usually finds a solution quickly, making it suitable for problems where time is a constraint.
  iii. Provides a good enough solution: While not always optimal, the Greedy Approach often finds a solution that is close to the best possible solution.
  iv. Can be used as a building block for other algorithms: The Greedy Approach can be used as a starting point for developing more complex algorithms.
  v. Useful for a variety of problems: The Greedy Approach can be applied to a wide range of optimization problems, including knapsack problems, scheduling problems, and routing problems.

However, it is important to remember that the Greedy Approach doesn't always find the **optimal solution** .

There are cases where it can lead to suboptimal solutions.

Therefore, it is necessary to carefully consider the problem and the potential drawbacks before using the Greedy Approach.

https://www.geeksforgeeks.org/introduction-to-greedy-algorithm-data-structures-and-algorithm-tutorials/

How does the Greedy Algorithm works?

- **Greedy Algorithm** solve optimization problems by making the best local choice at each step in the hope of finding the global optimum. It is similar to taking the best option available at each moment, hoping it will lead to the best overall outcome.
  1. Start with the initial state of the problem. This is the starting point from where you begin making choices.
  2. Evaluate all possible choices you can make from the current state. Consider all the options available at that specific moment.
  3. Choose the option that seems best at that moment, regardless of future consequences. This is the "greedy" part – you take the best option available now, even if it might not be the best in the long run.
  4. Move to the new state based on your chosen option. This becomes your new starting point for the next iteration.
  5. Repeat steps 2-4 until you reach the goal state or no further progress is possible. Keep making the best local choices until you reach the end of the problem or get stuck..

https://www.geeksforgeeks.org/introduction-to-greedy-algorithm-data-structures-and-algorithm-tutorials/

Greedy Algorithm Vs Dynamic Programming

Criteria	Greedy Algorithm	Dynamic Programming
Basic Idea	Makes the locally optimal choice at each stage	Solves subproblems and builds up to the optimal solution
Optimal Solution	Not always guaranteed to provide the globally optimal solution	Guarantees the globally optimal solution
Time Complexity	Typically faster; often linear or polynomial time	Usually slower due to solving overlapping subproblems
Space Complexity	Requires less memory; often constant or linear space	Requires more memory due to storing intermediate results
Subproblems Overlapping	Does NOT handle overlapping subproblems	Handles overlapping subproblems efficiently
Examples	Finding minimum spanning tree, Huffman coding	Matrix chain multiplication, shortest path problems
Global Solution	In general, greedy algorithms **may not** yield a global optimal solution, but they may produce good locally optimal solutions in a reasonable time and with less computational effort.	yield a global optimal solution

https://www.geeksforgeeks.org/introduction-to-greedy-algorithm-data-structures-and-algorithm-tutorials/

Advantages of Greedy Algorithms:

- Simple and easy to understand: Greedy algorithms are often straightforward to implement and reason about.
- Efficient for certain problems: They can provide optimal solutions for specific problems, like finding the shortest path in a graph with non-negative edge weights.
- Fast execution time: Greedy algorithms generally have lower time complexity compared to other algorithms for certain problems.
- Intuitive and easy to explain : The decision-making process in a greedy algorithm is often easy to understand and justify.
- Can be used as building blocks for more complex algorithms: Greedy algorithms can be combined with other techniques to design more sophisticated algorithms for challenging problems.

https://www.geeksforgeeks.org/introduction-to-greedy-algorithm-data-structures-and-algorithm-tutorials/

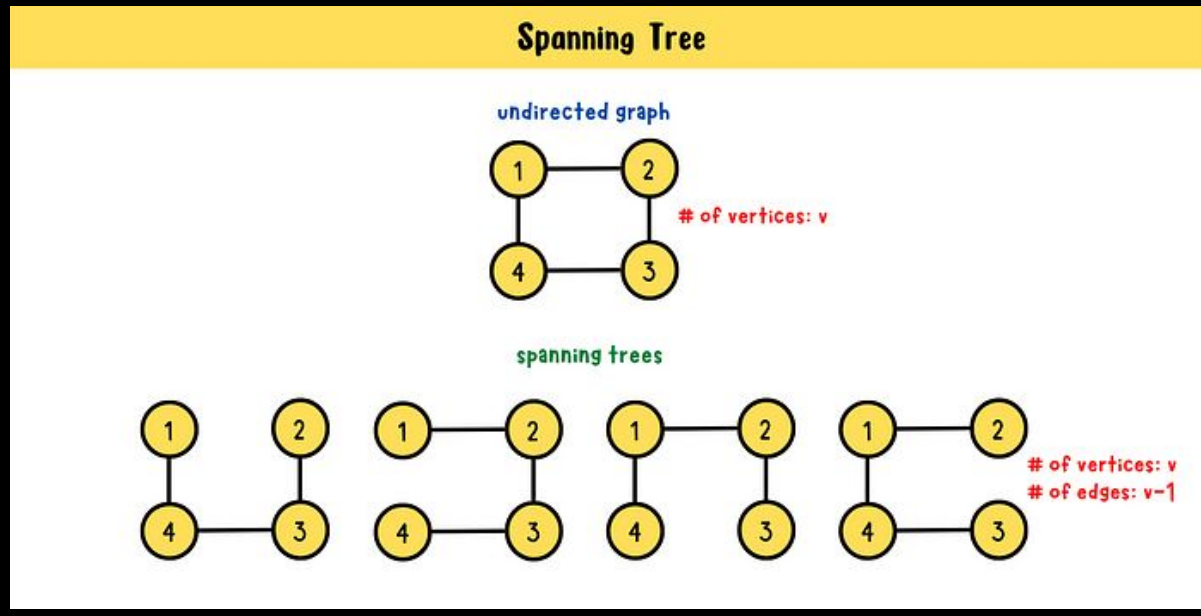
Disadvantages of the Greedy Approach:

- Not always optimal: Greedy algorithms prioritize local optima over global optima, leading to suboptimal solutions in some cases.
- Difficult to prove optimality: Proving the optimality of a greedy algorithm can be challenging, requiring careful analysis.
- Sensitive to input order: The order of input data can affect the solution generated by a greedy algorithm.
- Limited applicability: Greedy algorithms are not suitable for all problems and may not be applicable to problems with complex constraints.

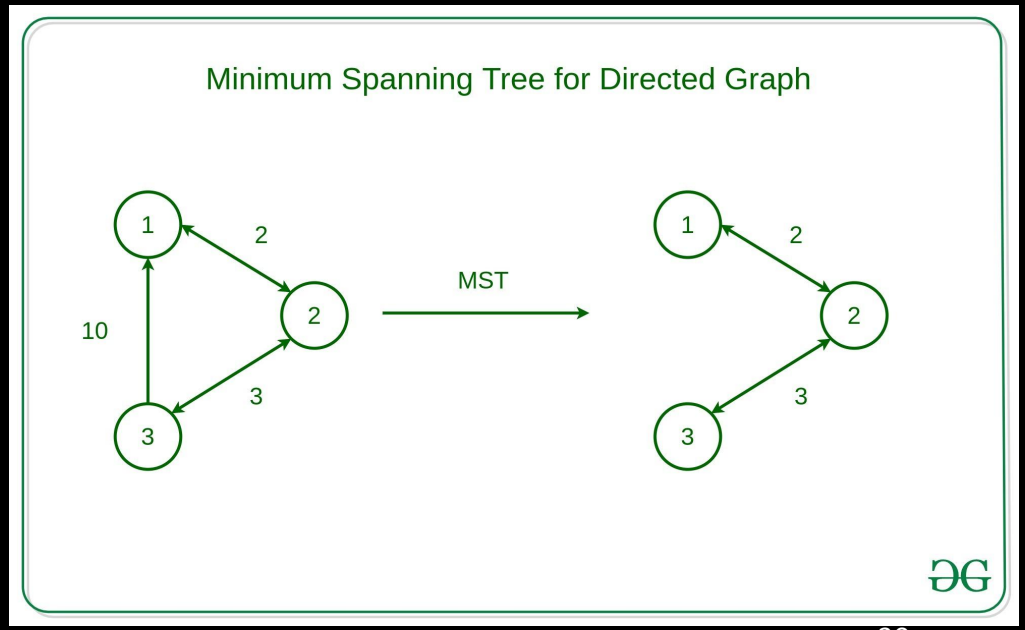
https://www.geeksforgeeks.org/introduction-to-greedy-algorithm-data-structures-and-algorithm-tutorials/

Spanning Tree

- A spanning tree is defined as a tree-like subgraph of a connected, undirected graph that includes all the vertices of the graph.
- i.e. it is a subset of the edges of the graph that forms a tree (acyclic) where every node of the graph is a part of the tree.
- The minimum spanning tree has all the properties of a spanning tree with an added constraint of having the minimum possible weights among all possible spanning trees.
- Like a spanning tree, there can also be many possible MSTs for a graph. (same min total wight)

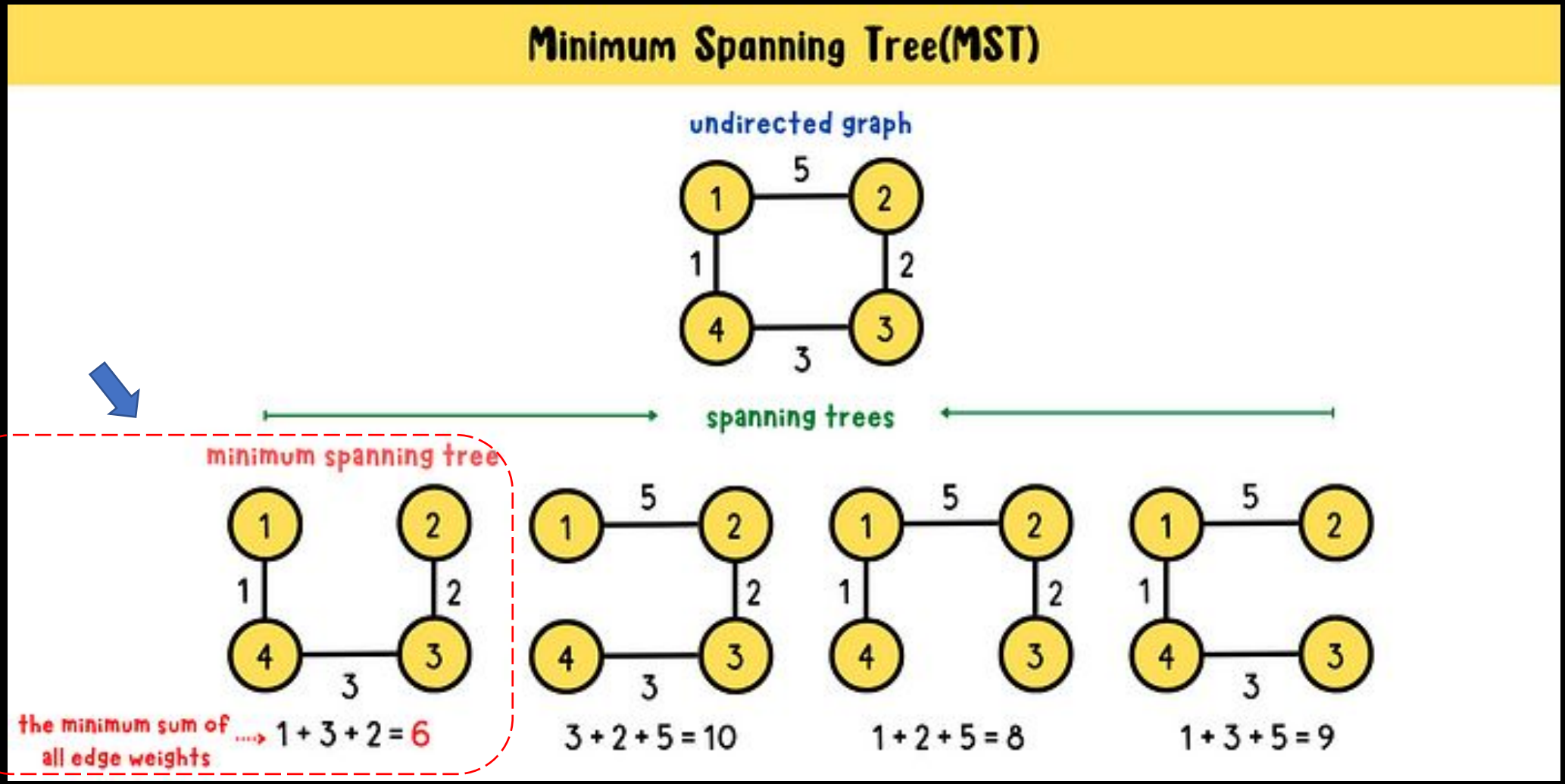


https://yuminlee2.medium.com/kruskals-algorithm-minimum-spanning-tree-db96e91d0aed



https://www.geeksforgeeks.org/what-is-minimum-spanning-tree-mst/

Minimum Spanning Tree:



https://yuminlee2.medium.com/kruskals-algorithm-minimum-spanning-tree-db96e91d0aed

Applications of Minimum Spanning Trees:

- Network design: Spanning trees can be used in network design to find the minimum number of connections required to connect all nodes. Minimum spanning trees, in particular, can help minimize the cost of the connections by selecting the cheapest edges.
- Image processing: Spanning trees can be used in image processing to identify regions of similar intensity or color, which can be useful for segmentation and classification tasks.
- Biology: Spanning trees and minimum spanning trees can be used in biology to construct phylogenetic trees to represent evolutionary relationships among species or genes.
- Social network analysis: Spanning trees and minimum spanning trees can be used in social network analysis to identify important connections and relationships among individuals or groups.

https://www.geeksforgeeks.org/what-is-minimum-spanning-tree-mst/

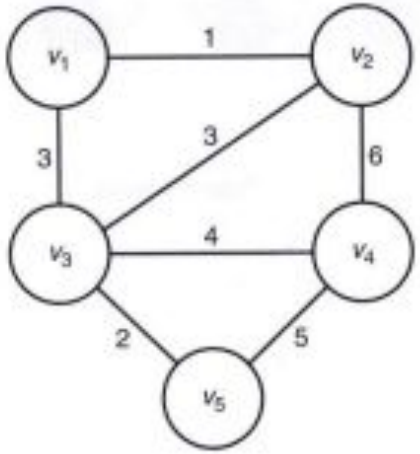
Algorithms to find Minimum Spanning Tree:

- There are several algorithms to find the minimum spanning tree from a given graph, some of them are listed below:
- Kruskal's Minimum Spanning Tree Algorithm:
- This is one of the popular algorithms for finding the minimum spanning tree from a connected, undirected graph. This is a greedy algorithm. The algorithm workflow is as below:
  1. First, it sorts all the edges of the graph by their weights,
  2. Then starts the iterations of finding the spanning tree.
  3. At each iteration, the algorithm adds the next lowest-weight edge one by one, such that the edges picked until now does not form a cycle.
- This algorithm can be implemented efficiently using a Disjoint-Set data structure to keep track of the connected components of the graph. This is used in a variety of practical applications such as network design, clustering, and data analysis.

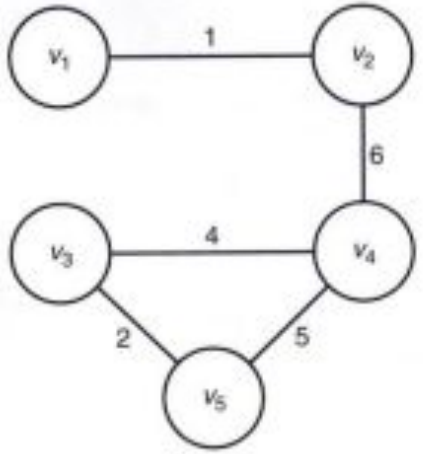
https://www.geeksforgeeks.org/what-is-minimum-spanning-tree-mst/

Example: A weighted graph and three subgraphs.

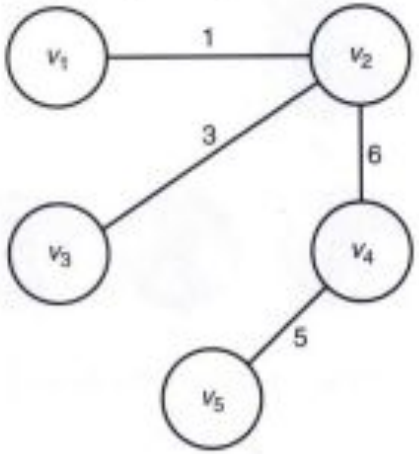
(a) A connected, weighted, undirected graph G.



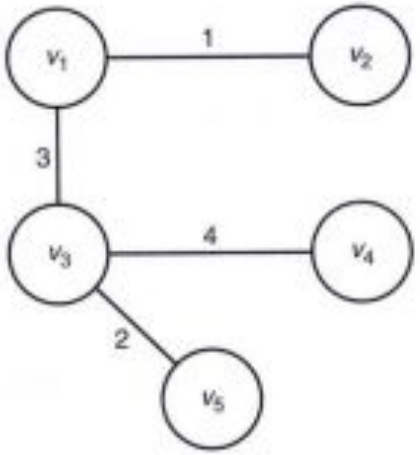
(b) If (v_4, v_5) were removed from this subgraph, the graph would remain connected.

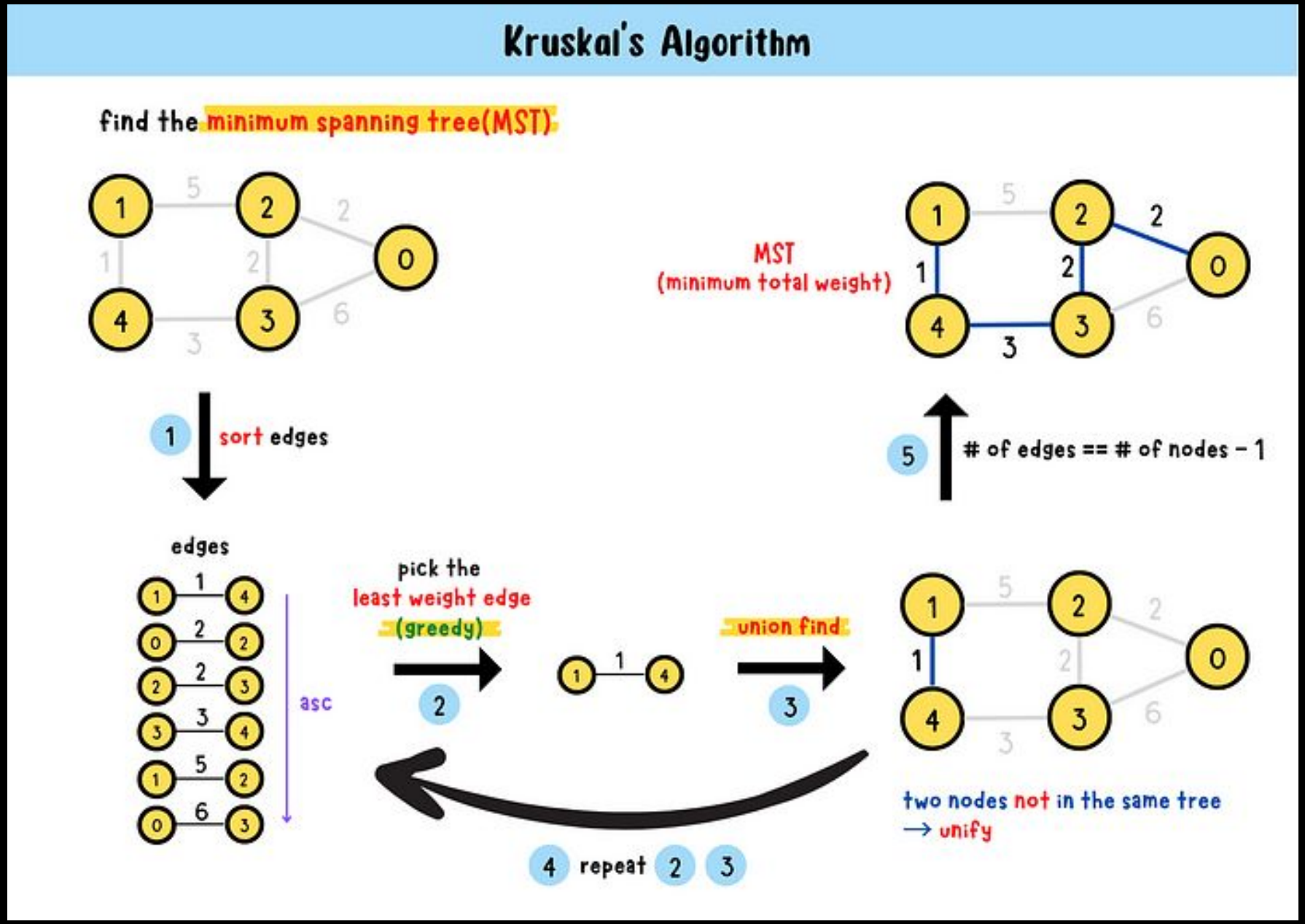


(c) A spanning tree for G.



(d) A minimum spanning tree for G.





algorithm is a **union find algorithm** because it implements **find operation** to check if nodes are in the same tree/set and perform **union operation** to merge two disjoint trees/sets

https://yuminlee2.medium.com/kruskals-algorithm-minimum-spanning-tree-db96e91d0aed

Algorithms to find Minimum Spanning Tree: ctd..

- Prim's Minimum Spanning Tree Algorithm:
- This is also a greedy algorithm. This algorithm has the following workflow:
  1. It starts by selecting an arbitrary vertex and then adding it to the MST.
  2. Then, it repeatedly checks for the minimum edge weight that connects one vertex of MST to another vertex that is not yet in the MST.
  3. This process is continued until all the vertices are included in the MST.
- To efficiently select the minimum weight edge for each iteration, this algorithm uses priority_queue to store the vertices sorted by their minimum edge weight currently.
- It also simultaneously keeps track of the MST using an array or other data structure suitable considering the data type it is storing.
- This algorithm can be used in various scenarios such as image segmentation based on color, texture, or other features. For Routing, as in finding the shortest path between two points for a delivery truck to follow.
- Both Kruskal's and Prim's algorithms have a time complexity of **O(ElogE)**, where E is the number of edges in the graph.

https://www.geeksforgeeks.org/what-is-minimum-spanning-tree-mst/

Kruskal's Minimum Spanning Tree (MST) Algorithm

- A minimum spanning tree (MST) or minimum weight spanning tree for a weighted, connected, undirected graph is a spanning tree with a weight less than or equal to the weight of every other spanning tree.
- In Kruskal's algorithm, sort all edges of the given graph in increasing order.
- Then it keeps on adding new edges and nodes in the MST if the newly added edge does not form a cycle.
- It picks the minimum weighted edge at first and the maximum weighted edge at last.
- Thus we can say that it makes a locally optimal choice in each step in order to find the optimal solution. Hence this is a Greedy Algorithm.

https://www.geeksforgeeks.org/what-is-minimum-spanning-tree-mst/

How to find MST using Kruskal's algorithm?

- Below are the steps for finding MST using Kruskal's algorithm:
  1. Sort all the edges in non-decreasing order of their weight.
  2. Pick the smallest edge. Check if it forms a cycle with the spanning tree formed so far. If the cycle is not formed, include this edge. Else, discard it.
  3. Repeat step #2 until there are (V-1) edges in the spanning tree.

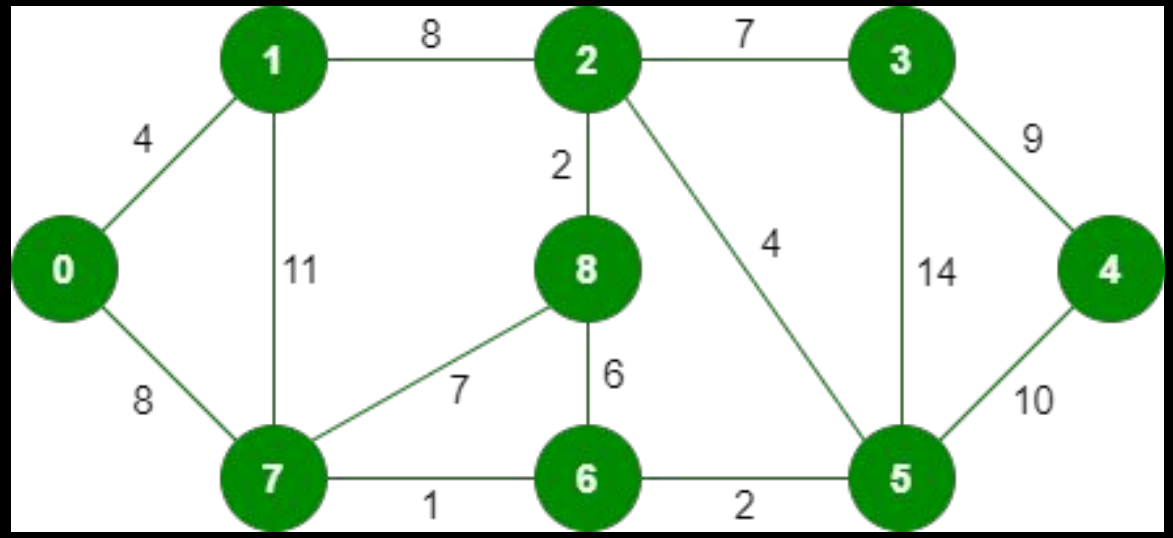
https://www.geeksforgeeks.org/what-is-minimum-spanning-tree-mst/

- *Step 2 uses the Union-Find algorithm to detect cycles.*
- *So the following post as a prerequisite.*
- *Union-Find Algorithm | Set 1 (Detect Cycle in a Graph)*
- *Union-Find Algorithm | Set 2 (Union By Rank and Path Compression)*

https://www.geeksforgeeks.org/what-is-minimum-spanning-tree-mst/

Kruskal's algorithm

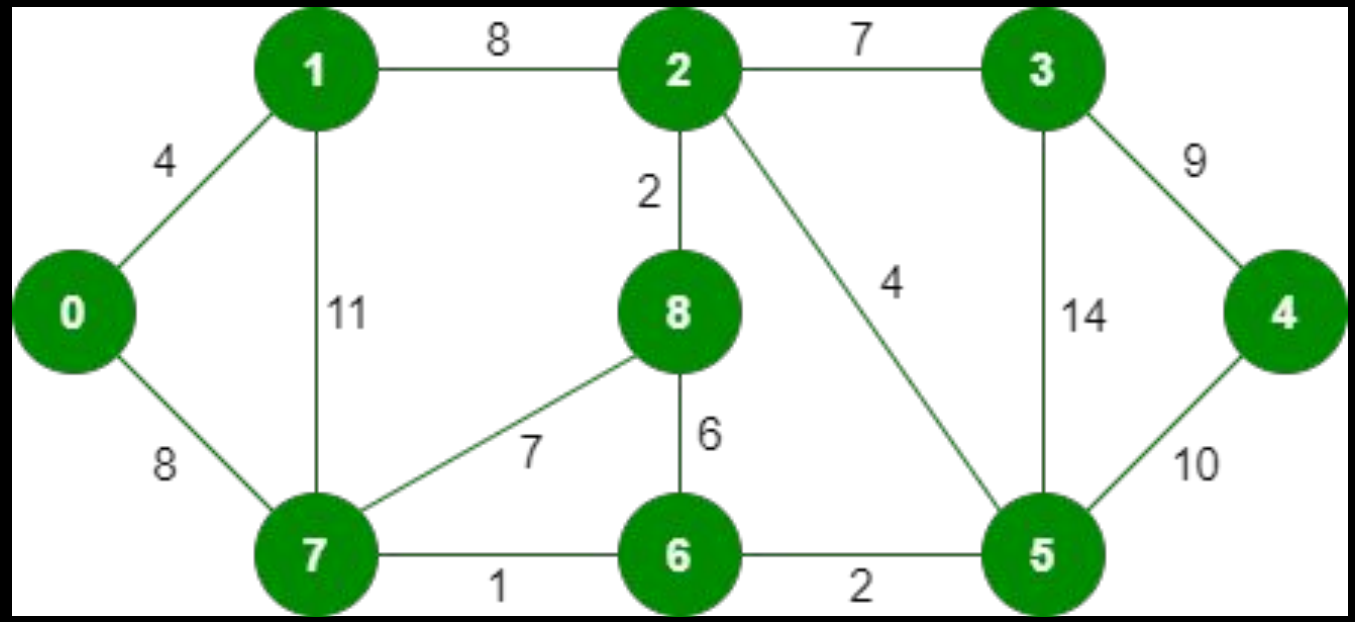
- Kruskal's algorithm to find the minimum cost spanning tree uses the greedy approach.
- The Greedy Choice is to pick the smallest weight edge that does not cause a cycle in the MST constructed so far. Let us understand it with an example: Below is the illustration of the above approach:



https://www.geeksforgeeks.org/kruskals-minimum-spanning-tree-algorithm-greedy-algo-2/

Weight	Source Vertex	Destination Vertex
1	7	6
2	8	2
2	6	5
4	0	1
4	2	5
6	8	6
7	2	3
7	7	8
8	0	7
8	1	2
9	3	4
10	5	4
11	1	7
14	3	5

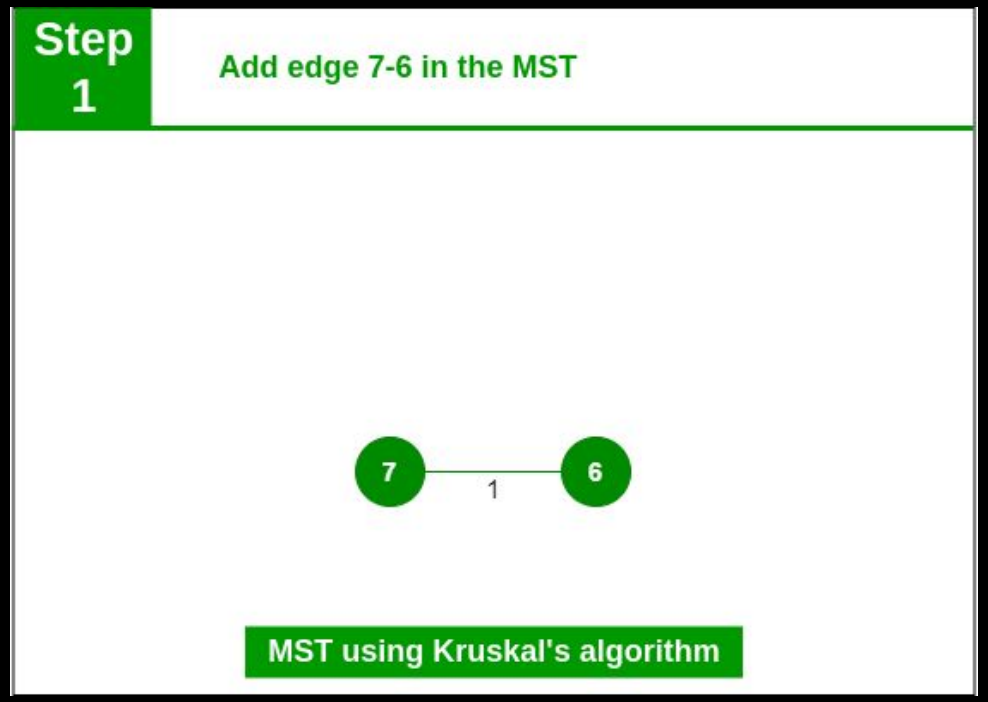
Kruskal's algorithm example



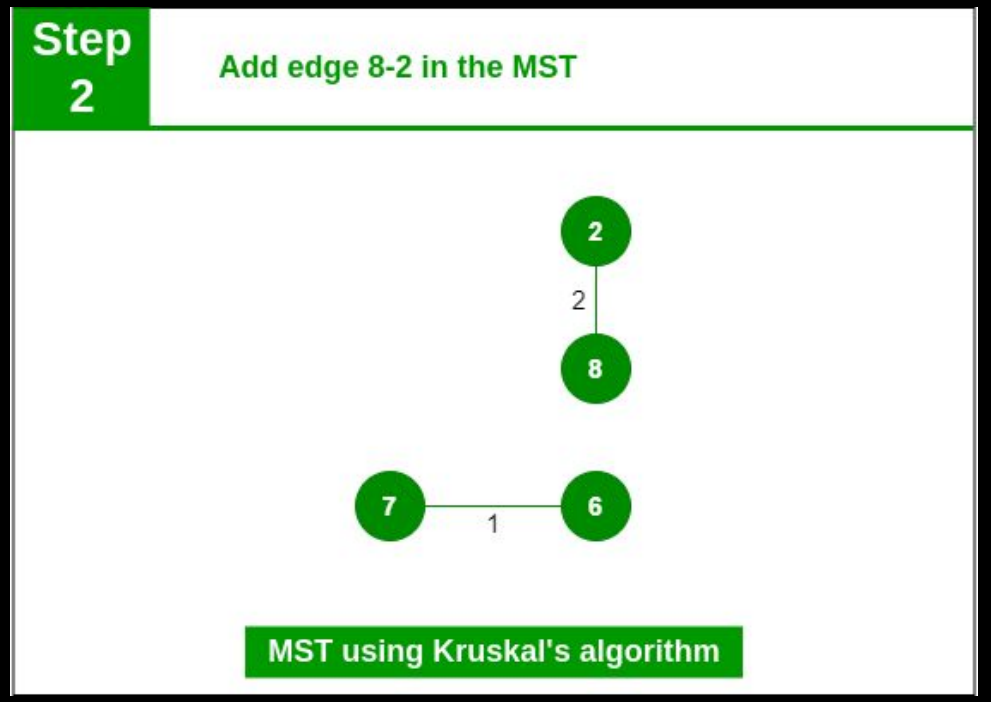
https://www.geeksforgeeks.org/kruskals-minimum-spanning-tree-algorithm-greedy-algo-2/

Kruskal's algorithm example ctd..

- *Now pick all edges one by one from the sorted list of edges*
- ***Step 1:** Pick edge 7-6. No cycle is formed, include it.*



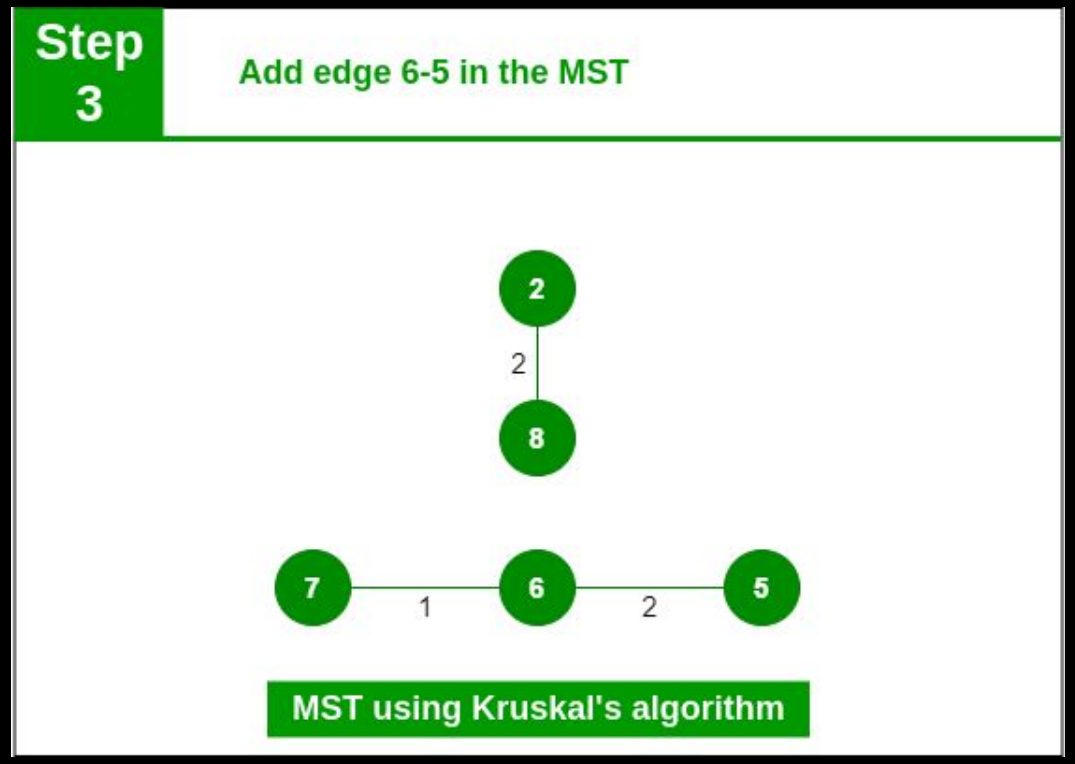
- ***Step 2:** Pick edge 8-2. No cycle is formed, include it.*



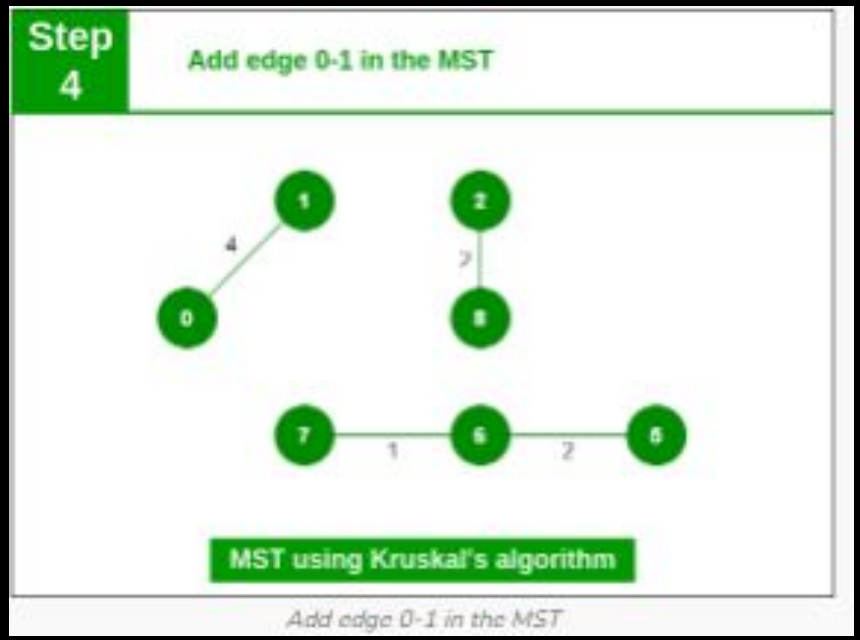
https://www.geeksforgeeks.org/kruskals-minimum-spanning-tree-algorithm-greedy-algo-2/

Kruskal's algorithm example ctd..

- Step 3: Pick edge 6-5. No cycle is formed, include it.



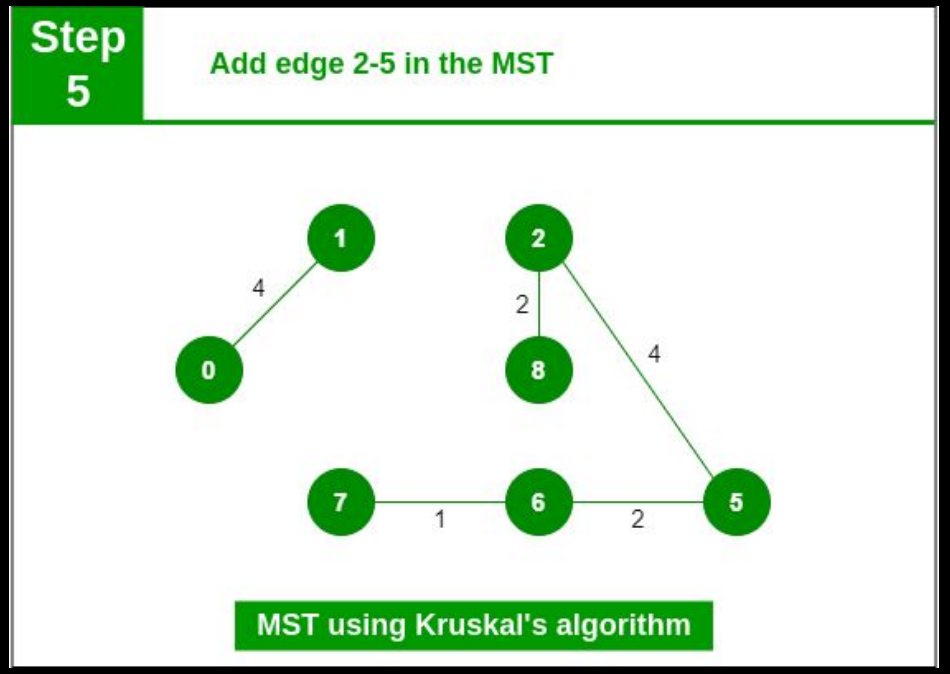
- Step 4: Pick edge 0-1. No cycle is formed, include it.



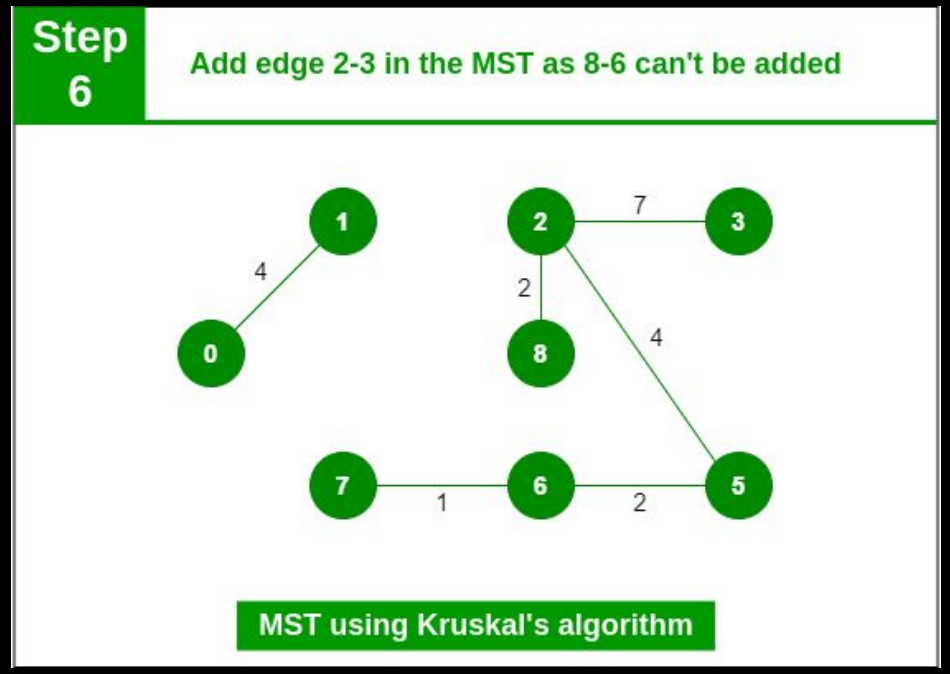
https://www.geeksforgeeks.org/kruskals-minimum-spanning-tree-algorithm-greedy-algo-2/

Kruskal’s algorithm example ctd..

- Step 5: Pick edge 2-5. No cycle is formed, include it.



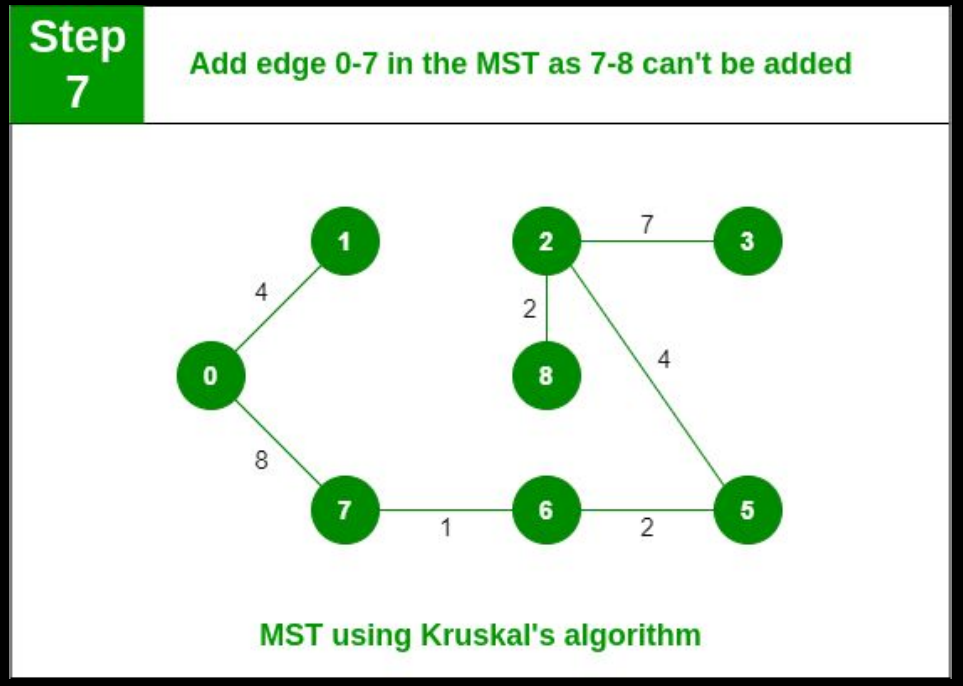
- Step 5: Pick edge 2-5. No cycle is formed, include it.



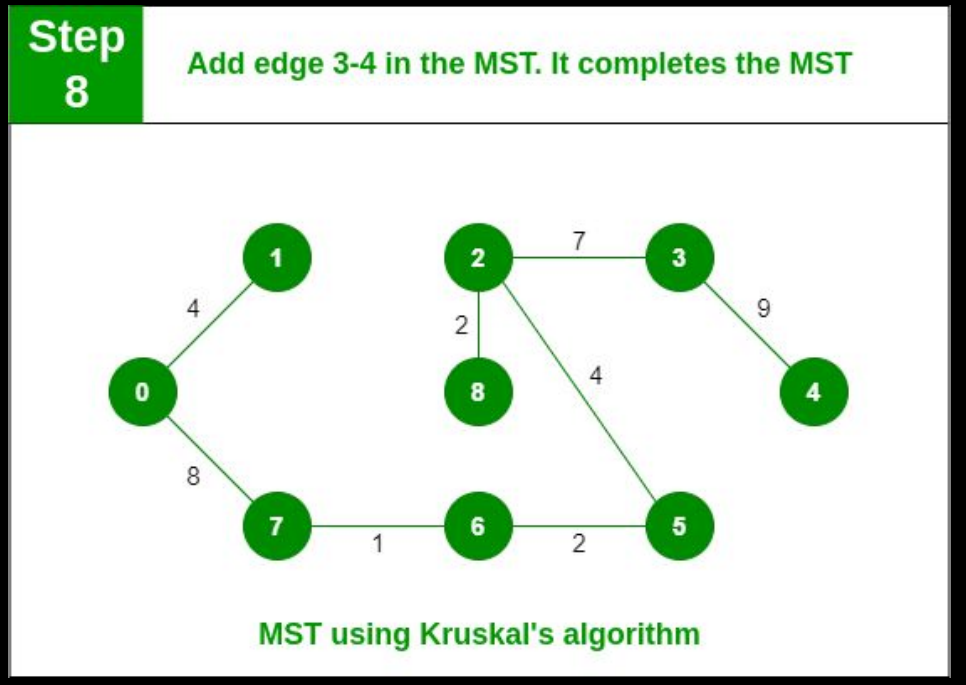
https://www.geeksforgeeks.org/kruskals-minimum-spanning-tree-algorithm-greedy-algo-2/

Kruskal's algorithm example ctd..

- ***Step 7:*** *Pick edge 7-8. Since including this edge results in the cycle, discard it. Pick edge 0-7. No cycle is formed, include it.*



- *Step 8: Pick edge 1-2. Since including this edge results in the cycle, discard it. Pick edge 3-4. No cycle is formed, include it.*



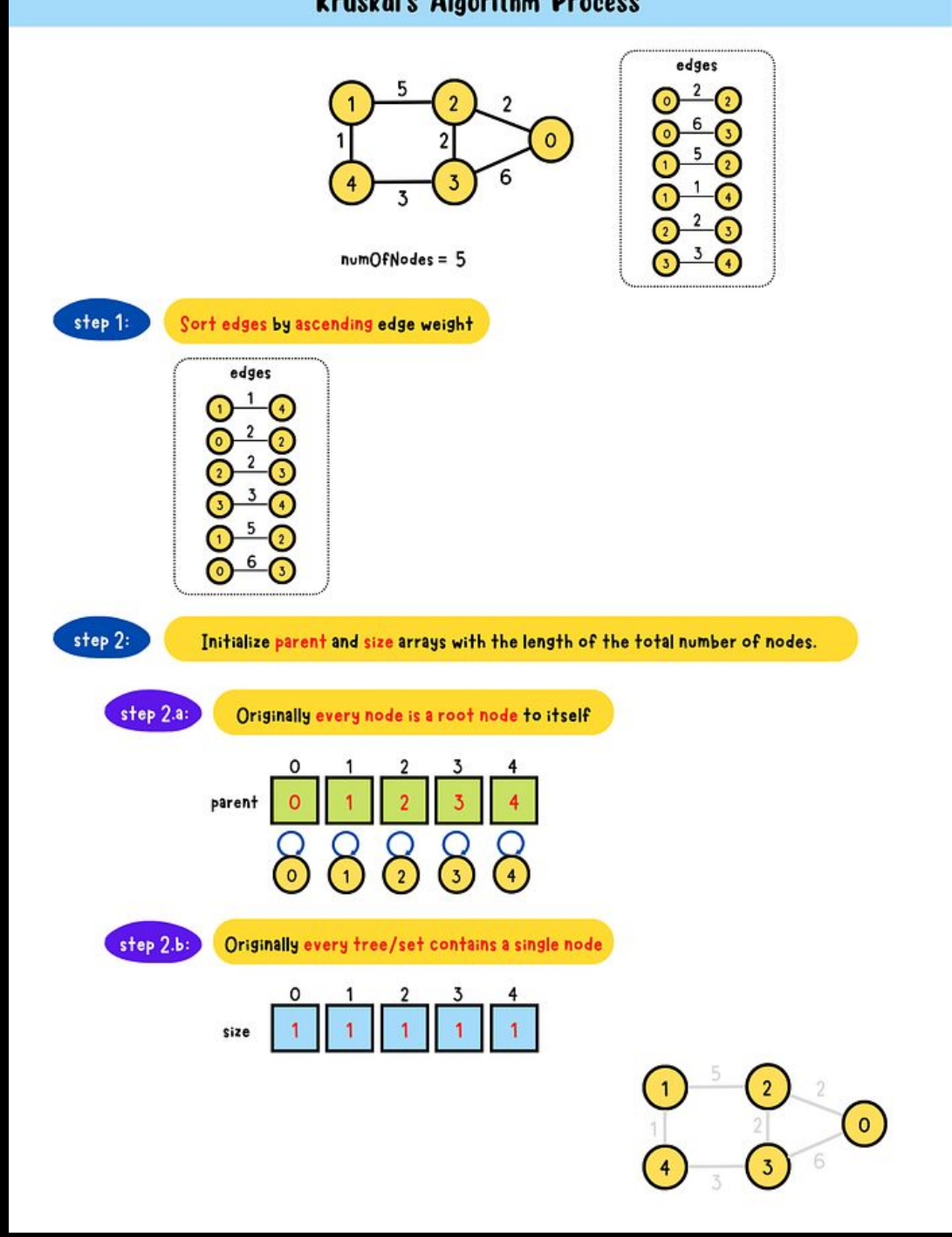
Note: *Since the number of edges included in the MST equals to (V – 1), so the algorithm stops here*
*** the number of edges in the MST is always V - 1, where V is the number of vertices in a connected undirected graph*

https://www.geeksforgeeks.org/kruskals-minimum-spanning-tree-algorithm-greedy-algo-2/

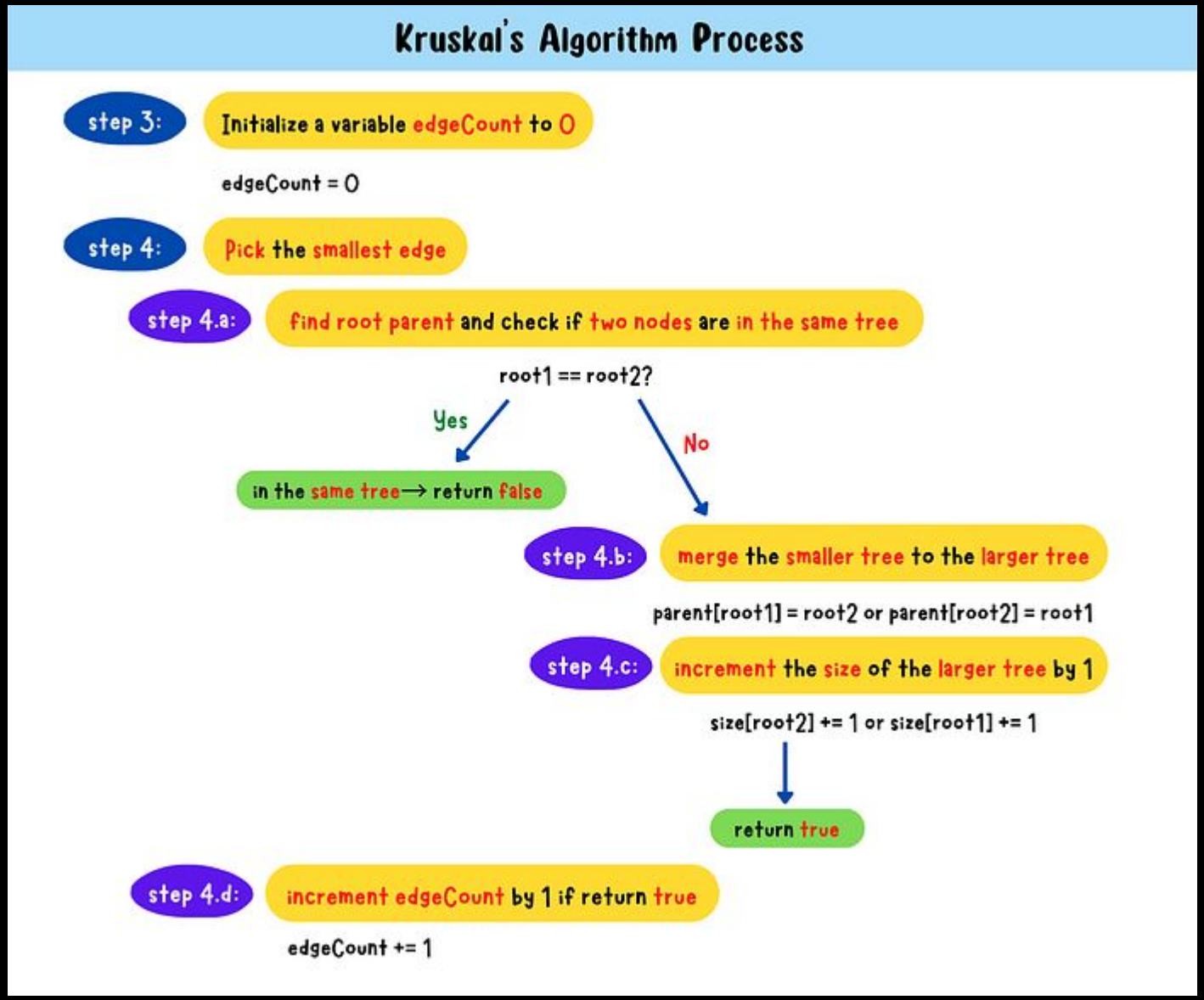
Another Example

- Ref **: Kruskal's Algorithm: Minimum Spanning Tree**
- https://yuminlee2.medium.com/kruskals-algorithm-minimum-spanning-tree-db96e91d0aed

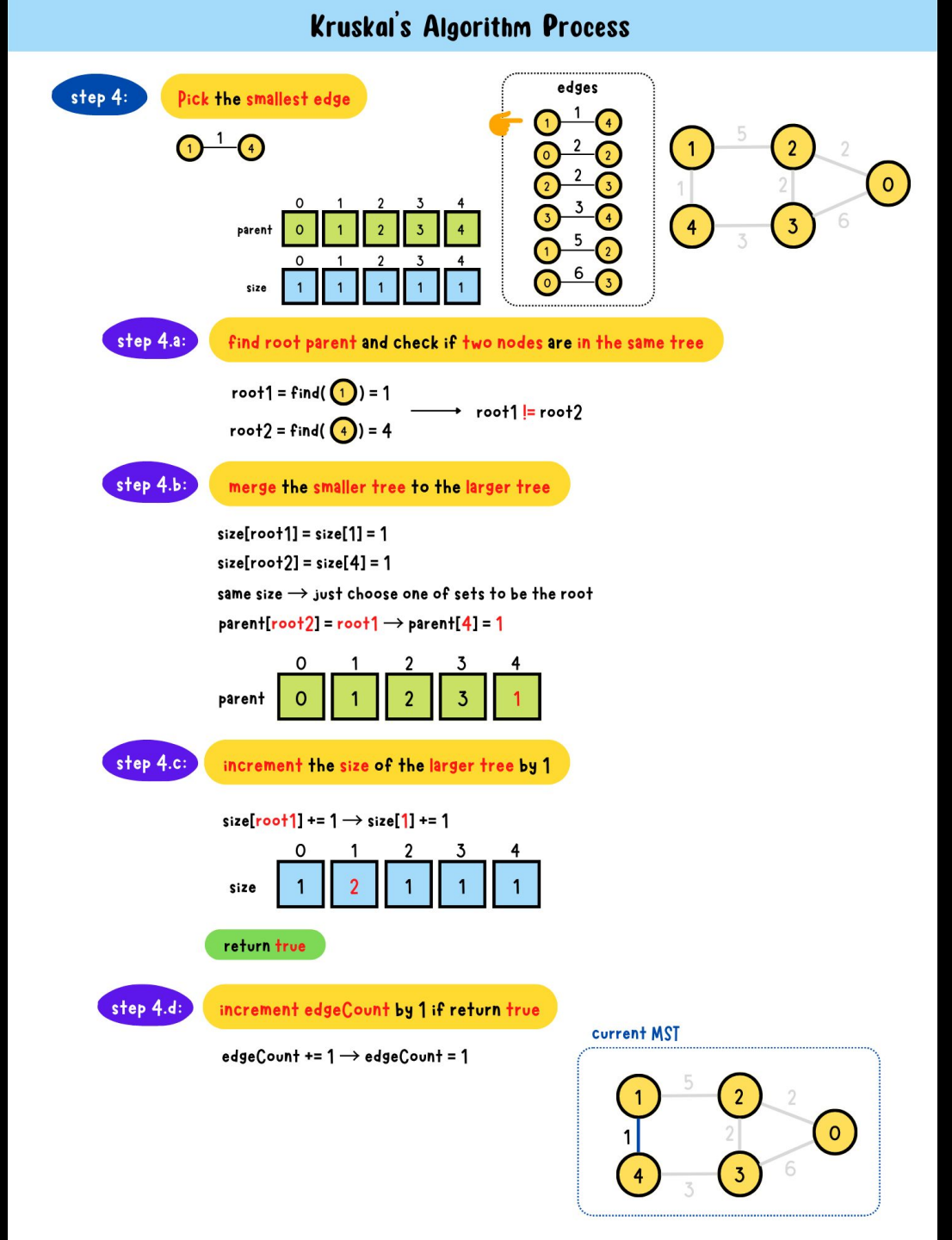
- ***step1***: **Sort edges** by **ascending** edge weight
- ***step2***:
  Initialize **parent** and **size** arrays with the length of the total number of nodes.



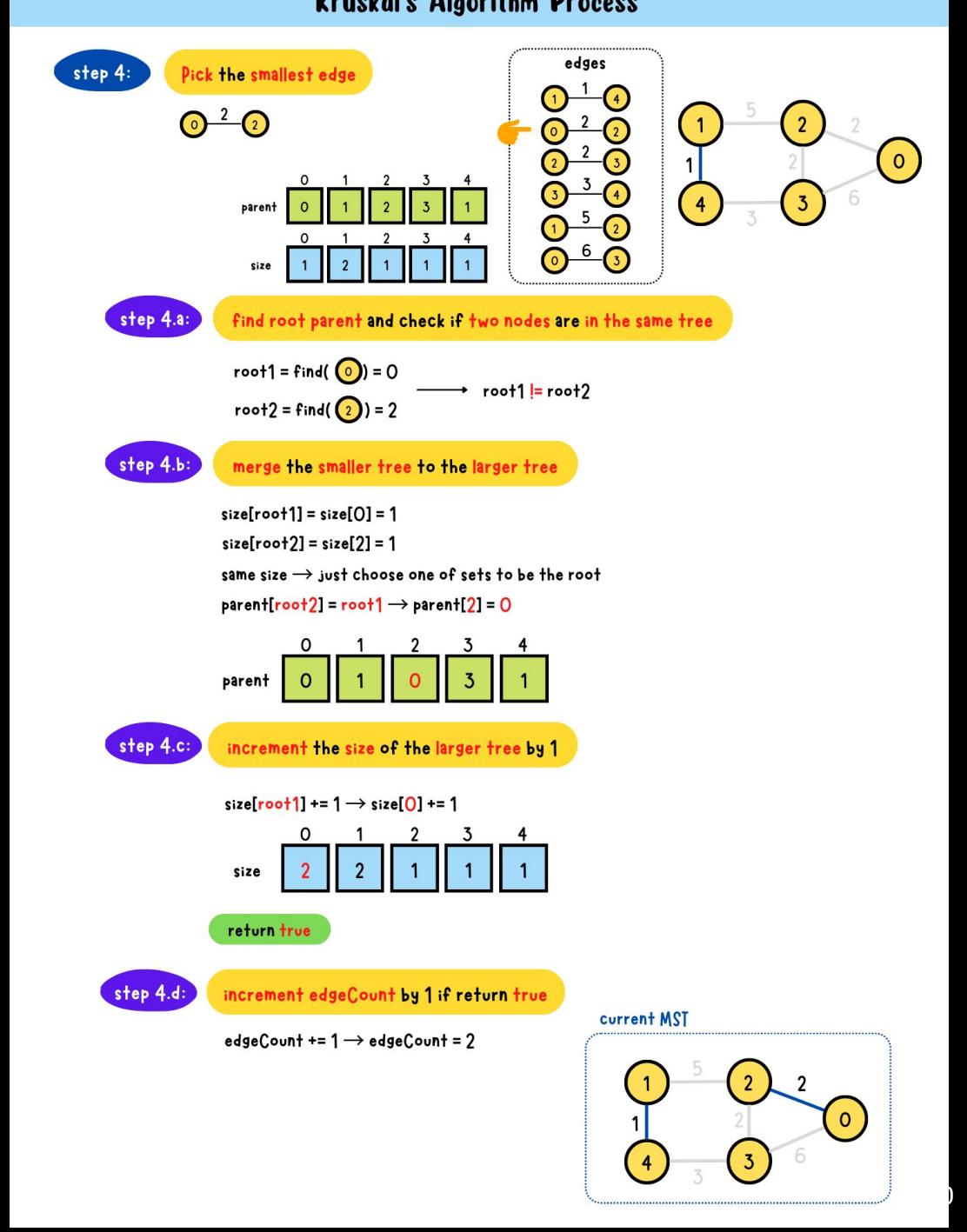
- ***step3***: Initialize a variable **edgeCount** to **0**
- ***step4***: **Pick** the **smallest edge**



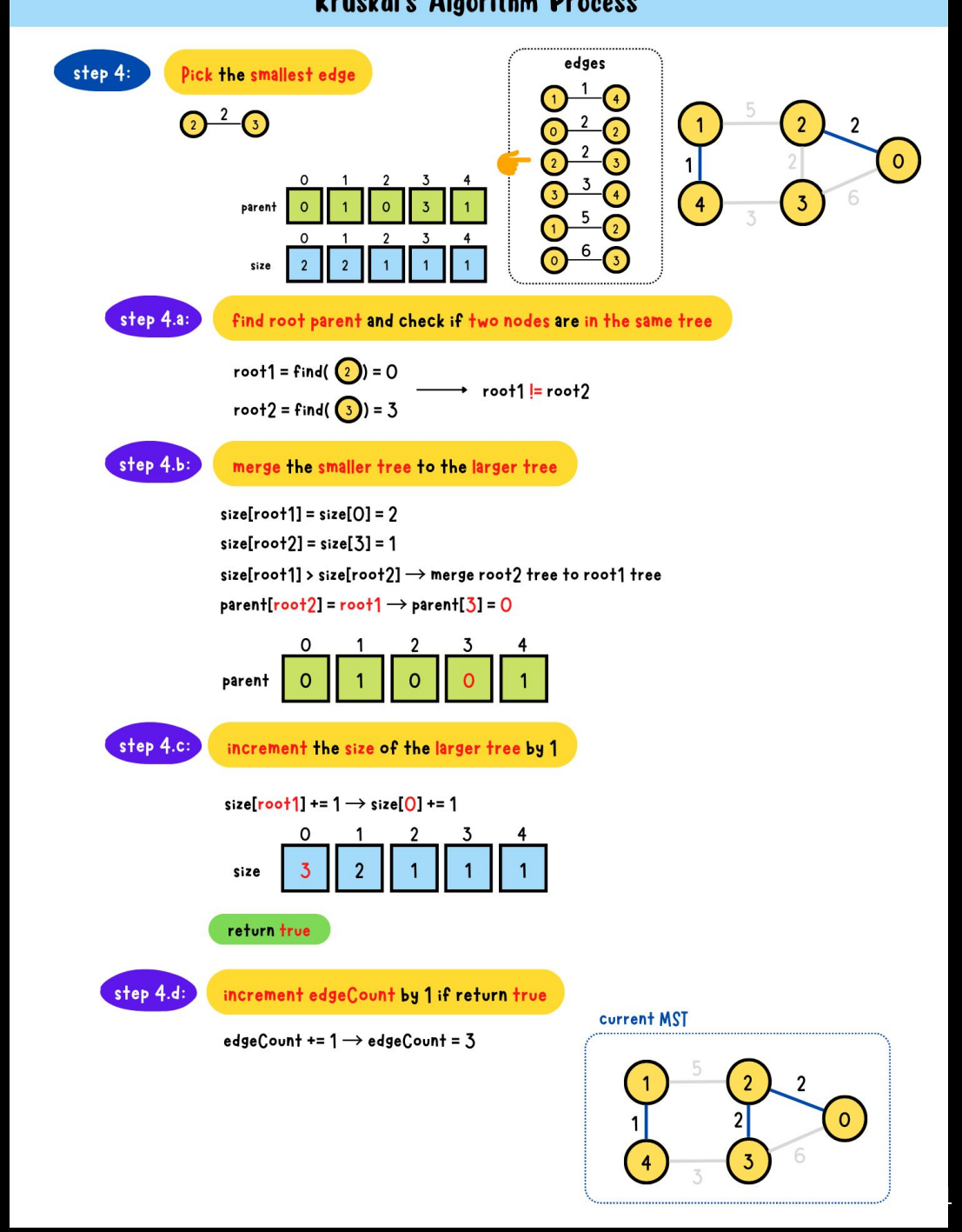
- ***step5***: repeat ***step4*** until it includes (**the number of nodes -1) edges** in the MST



- ***step5***: repeat ***step4*** until it includes (**the number of nodes -1) edges** in the MST



- ***step5***: repeat ***step4*** until it includes (**the number of nodes -1) edges** in the MST



Complexity

- Time: O(elog(e) + elog(v))
  - elog(e): sort all the edges
  - elog(v): iterate through all edges(e) and perform union find (find operation: log(v))

- Space: O(v) parent and size array
  - v: the total number of vertices
  - e: the total number of edges

https://yuminlee2.medium.com/kruskals-algorithm-minimum-spanning-tree-db96e91d0aed

Prim's Algorithm for Minimum Spanning Tree (MST)

- Like Kruskal's algorithm, Prim's algorithm is also a Greedy algorithm. This algorithm always starts with a single node and moves through several adjacent nodes, in order to explore all of the connected edges along the way.

1. The algorithm starts with an empty spanning tree.
2. The idea is to maintain two sets of vertices. The first set contains the vertices already included in the MST, and the other set contains the vertices not yet included.
3. At every step, it considers all the edges that connect the two sets and picks the minimum weight edge from these edges
4. After picking the edge, it moves the other endpoint of the edge to the set containing MST.

https://www.geeksforgeeks.org/prims-minimum-spanning-tree-mst-greedy-algo-5/

Prim’s Algorithm

- Prim's algorithm was invented in 1930 by the Czech mathematician Vojtěch Jarník.
- The algorithm was then rediscovered by Robert C. Prim in 1957, and also rediscovered by Edsger W. Dijkstra in 1959. Therefore, the algorithm is also sometimes called "Jarník's algorithm", or the "Prim-Jarník algorithm"
- A group of edges that connects two sets of vertices in a graph is called cut in graph theory.
- So, at every step of Prim’s algorithm, find a cut, pick the minimum weight edge from the cut, and include this vertex in MST Set (the set that contains already included vertices).

https://www.w3schools.com/dsa/dsa_algo_mst_prim.php

https://www.geeksforgeeks.org/prims-minimum-spanning-tree-mst-greedy-algo-5/

How does Prim's Algorithm Work?

- The working of Prim's algorithm can be described by using the following steps:

  Step 1: Determine an arbitrary vertex as the starting vertex of the MST.

  Step 2: Follow steps 3 to 5 till there are vertices that are not included in the MST (known as fringe vertex).

  Step 3: Find edges connecting any tree vertex with the fringe vertices.

  Step 4: Find the minimum among these edges.

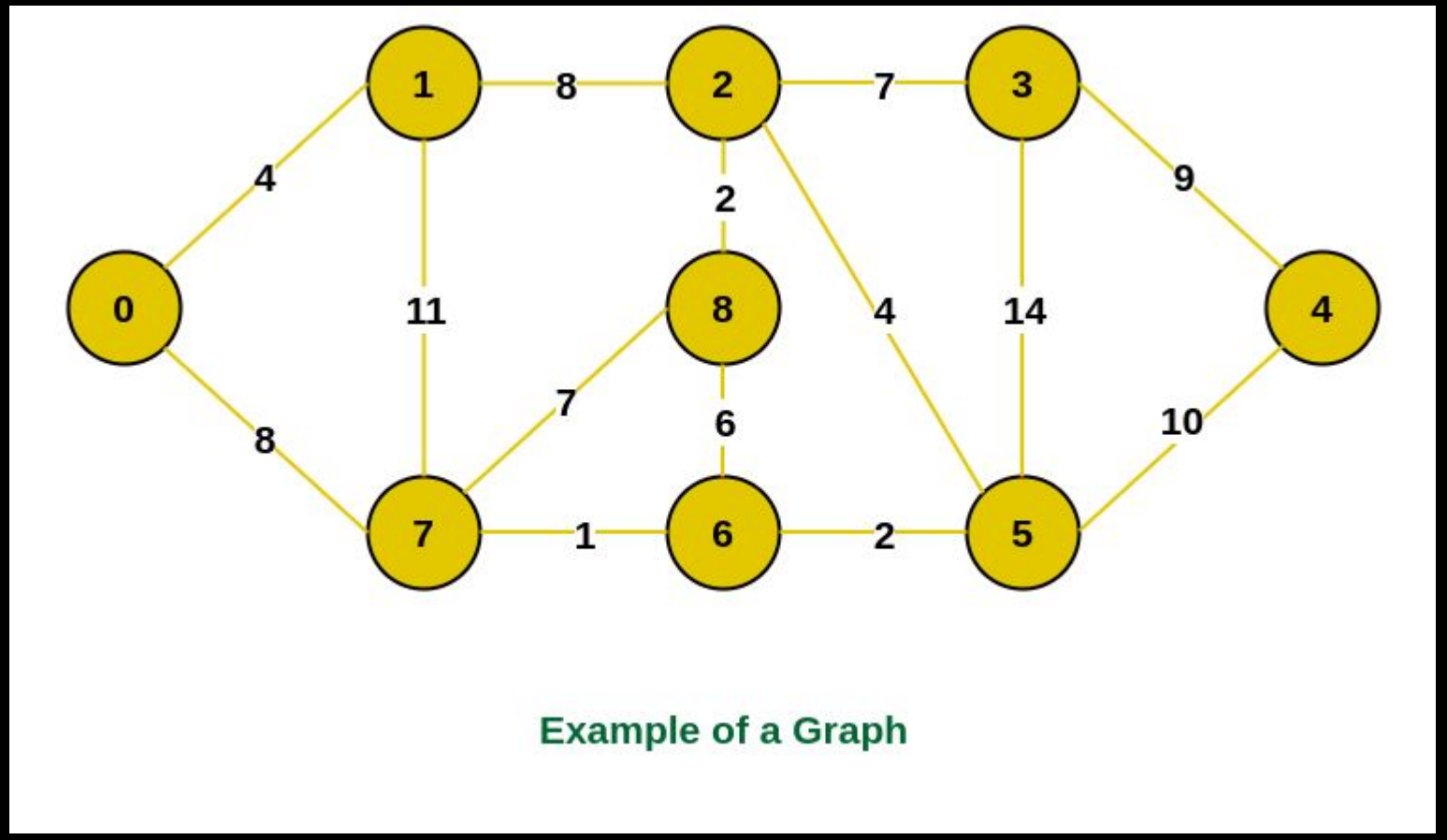
  Step 5: Add the chosen edge to the MST if it does not form any cycle.

  Step 6: Return the MST and exit

https://www.geeksforgeeks.org/prims-minimum-spanning-tree-mst-greedy-algo-5/

Illustration of Prim's Algorithm:

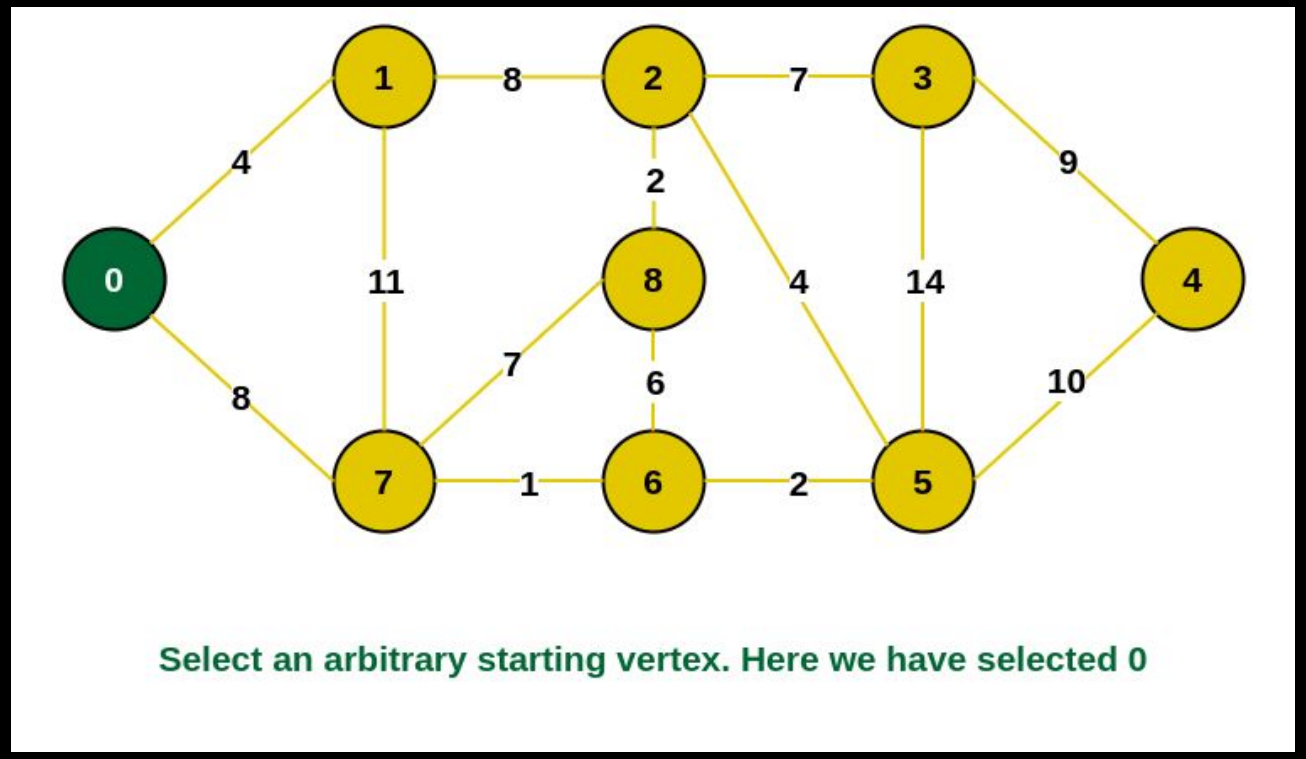
- Consider the following graph as an example for which we need to find the Minimum Spanning Tree (MST).



https://www.geeksforgeeks.org/prims-minimum-spanning-tree-mst-greedy-algo-5/

Prim's Algorithm

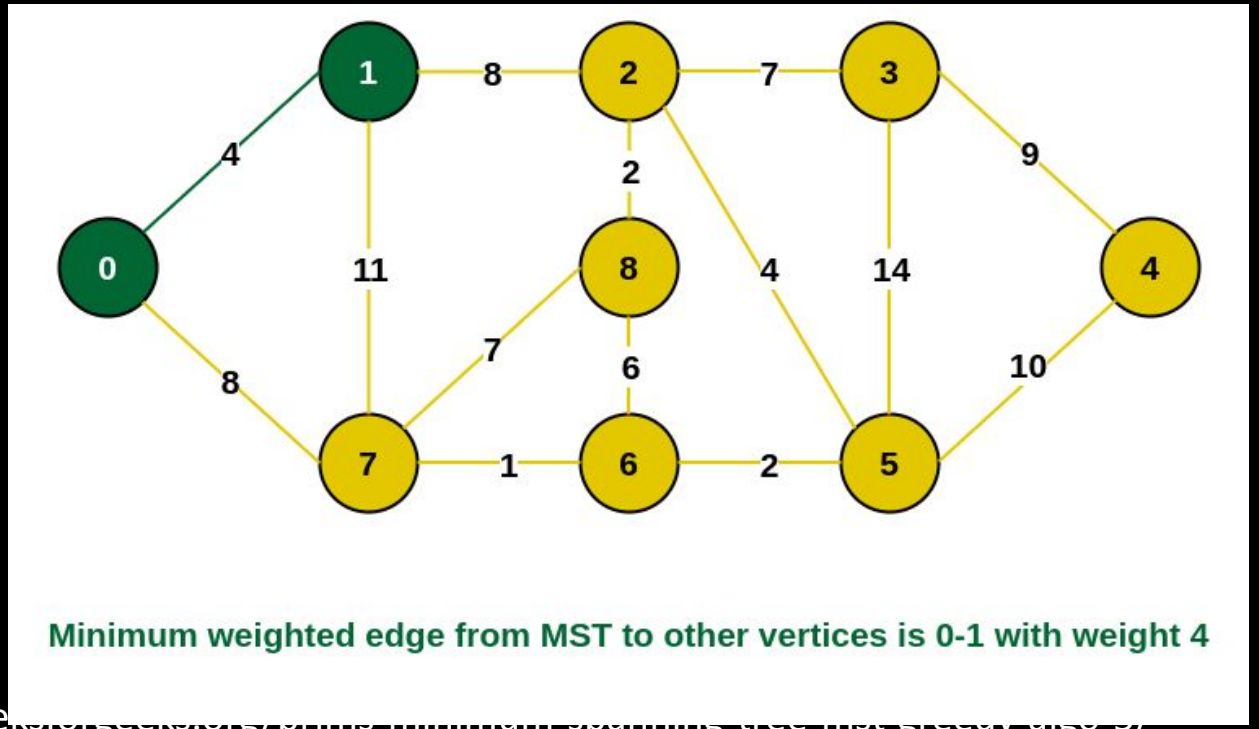
- Step 1: Firstly, we select an arbitrary vertex that acts as the starting vertex of the Minimum Spanning Tree. Here we have selected vertex 0 as the starting vertex.



https://www.geeksforgeeks.org/prims-minimum-spanning-tree-mst-greedy-algo-5/

Prim's Algorithm ctd..

- Step 2: All the edges connecting the incomplete MST and other vertices are the edges {0, 1} and {0, 7}. Between these two the edge with minimum weight is {0, 1}. So include the edge and vertex 1 in the MST.

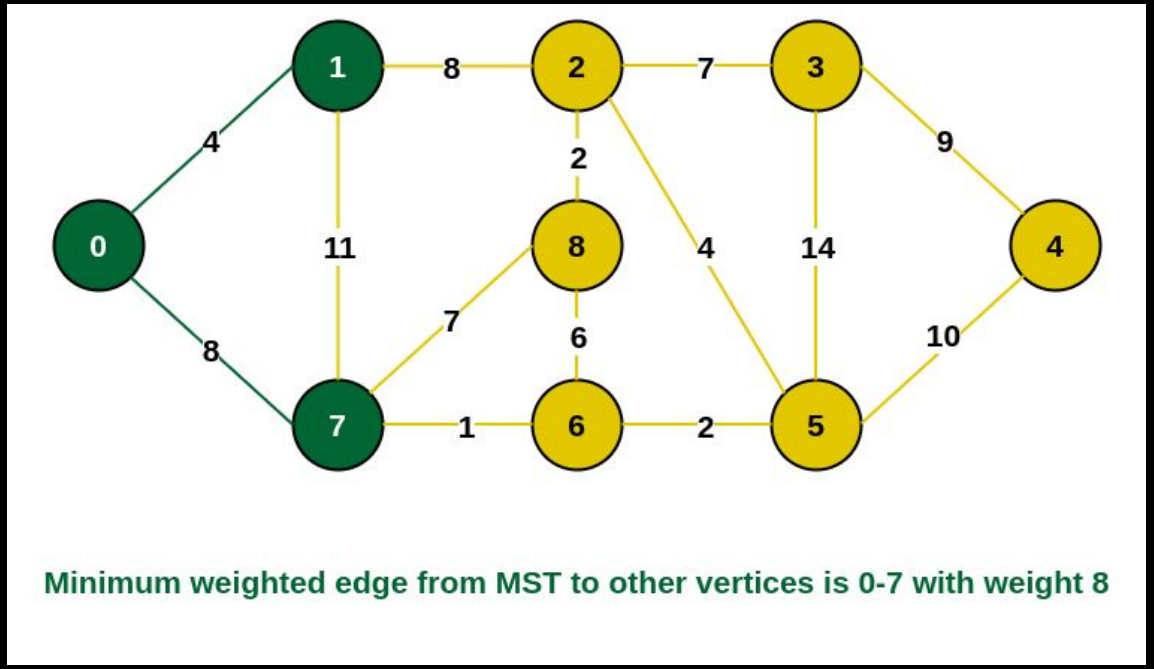


Minimum weighted edge from MST to other vertices is 0-1 with weight 4

https://www.geeksforgeeks.org/prims-minimum-spanning-tree-mst-greedy-algo-5/

Prim's Algorithm ctd..

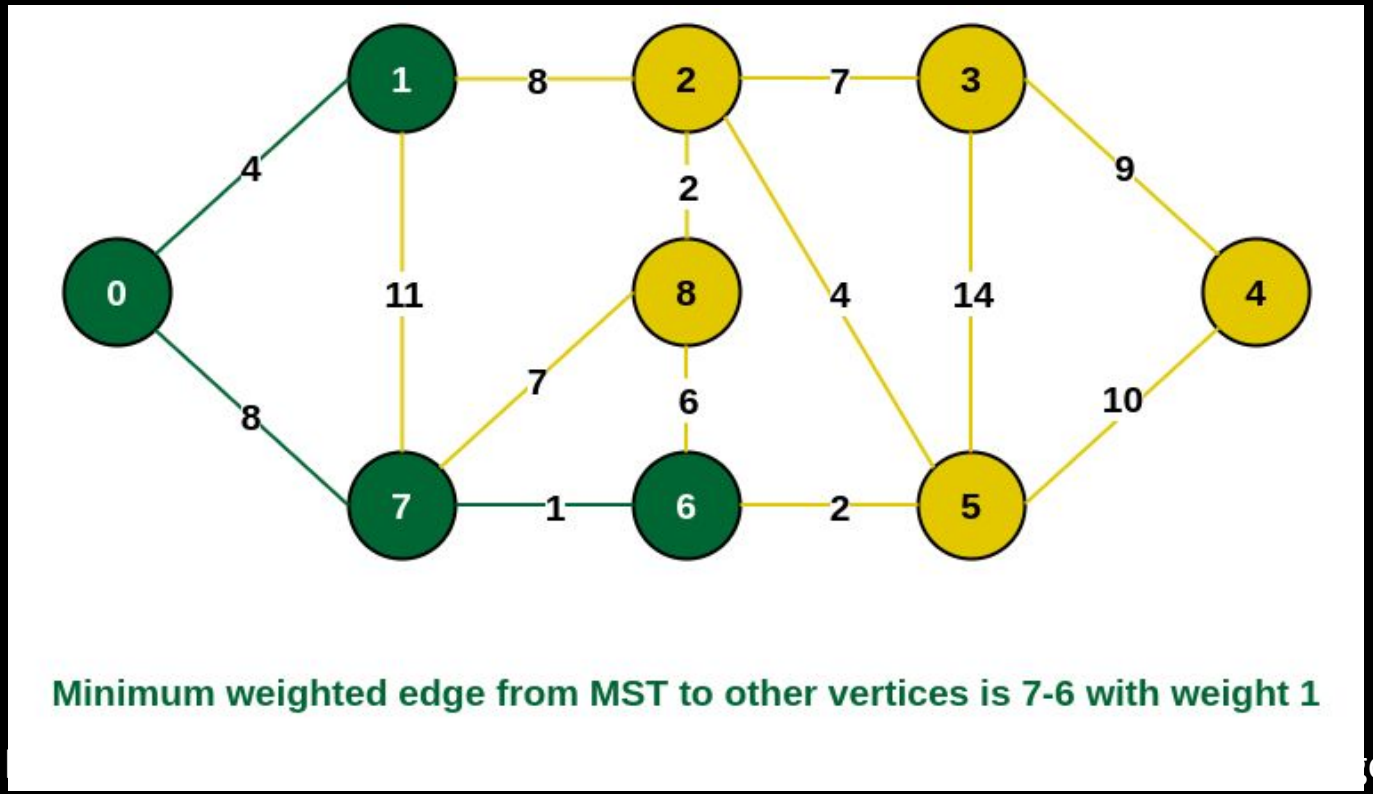
- Step 3: The edges connecting the incomplete MST to other vertices are {0, 7}, {1, 7} and {1, 2}. Among these edges the minimum weight is 8 which is of the edges {0, 7} and {1, 2}. Let us here include the edge {0, 7} and the vertex 7 in the MST. [We could have also included edge {1, 2} and vertex 2 in the MST].



https://www.geeksforgeeks.org/prims-minimum-spanning-tree-mst-greedy-algo-5/

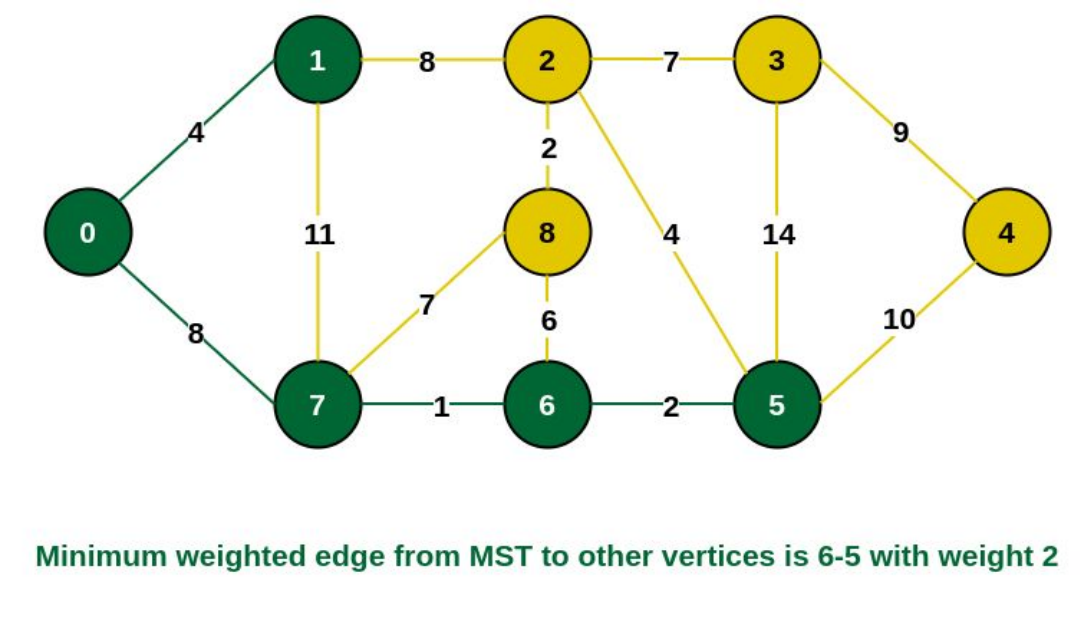
Prim's Algorithm ctd..

- Step 4: The edges that connect the incomplete MST with the fringe vertices are {1, 2}, {7, 6} and {7, 8}. Add the edge {7, 6} and the vertex 6 in the MST as it has the least weight (i.e., 1).

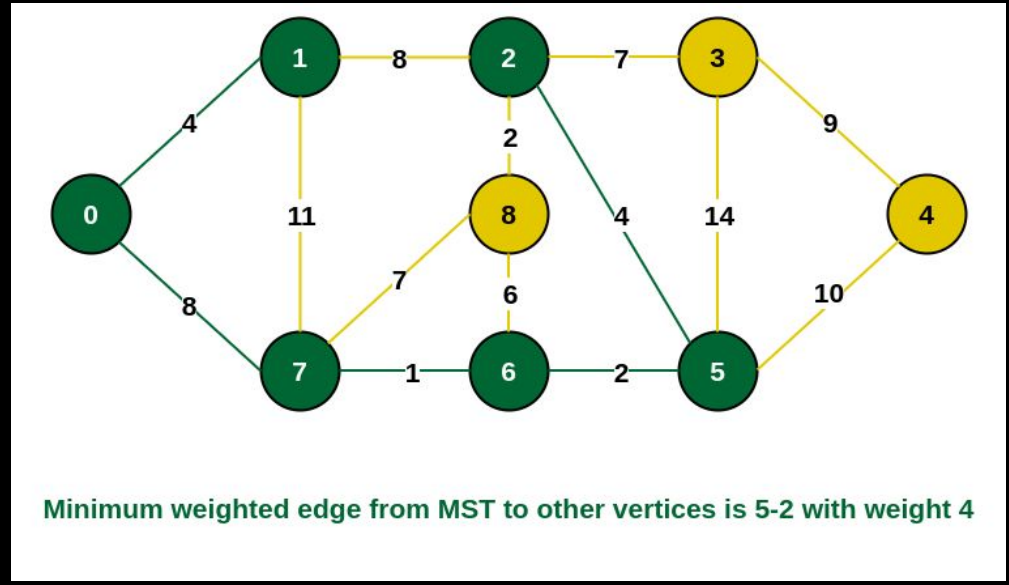


Prim's Algorithm ctd..

- Step 5: The connecting edges now are {7, 8}, {1, 2}, {6, 8} and {6, 5}. Include edge {6, 5} and vertex 5 in the MST as the edge has the minimum weight (i.e., 2) among them.



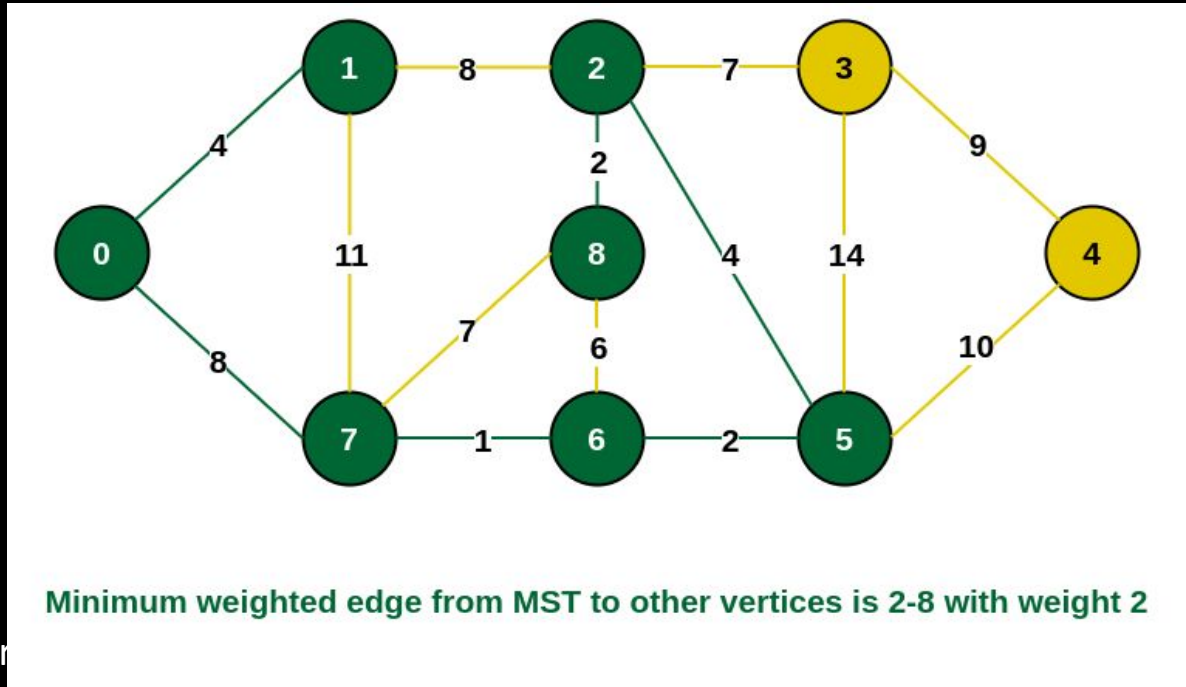
- Step 6: Among the current connecting edges, the edge {5, 2} has the minimum weight. So include that edge and the vertex 2 in the MST.



https://www.geeksforgeeks.org/prims-minimum-spanning-tree-mst-greedy-algo-5/

Prim's Algorithm ctd..

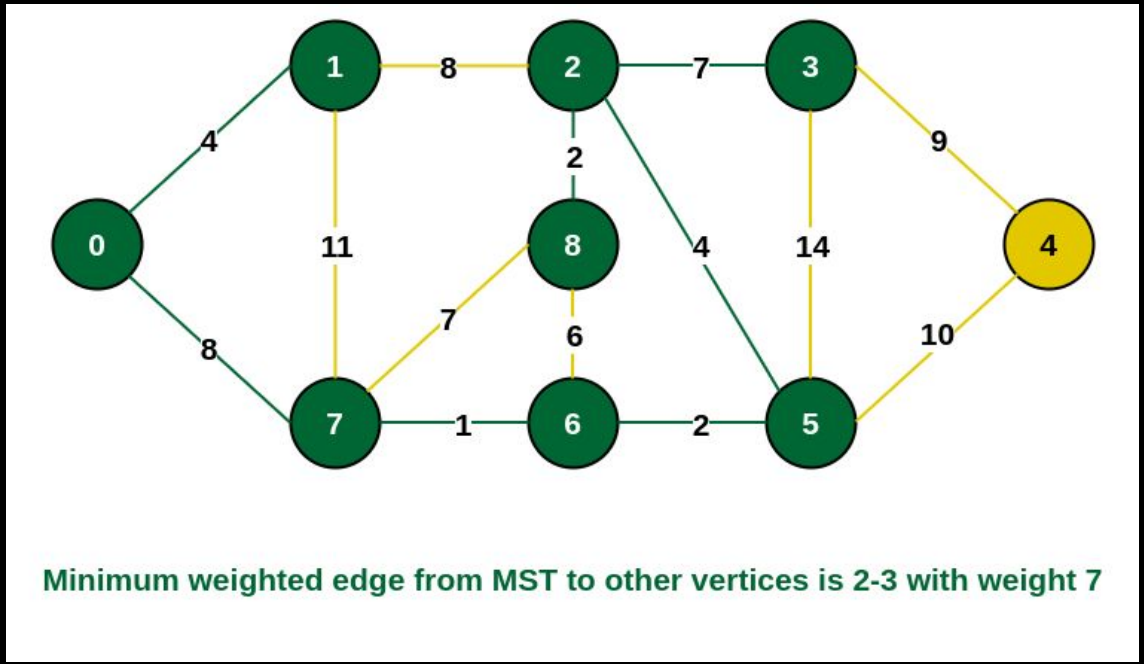
- Step 7: The connecting edges between the incomplete MST and the other edges are {2, 8}, {2, 3}, {5, 3} and {5, 4}. The edge with minimum weight is edge {2, 8} which has weight 2. So include this edge and the vertex 8 in the MST.



https://www.geeksfor

Prim's Algorithm ctd..

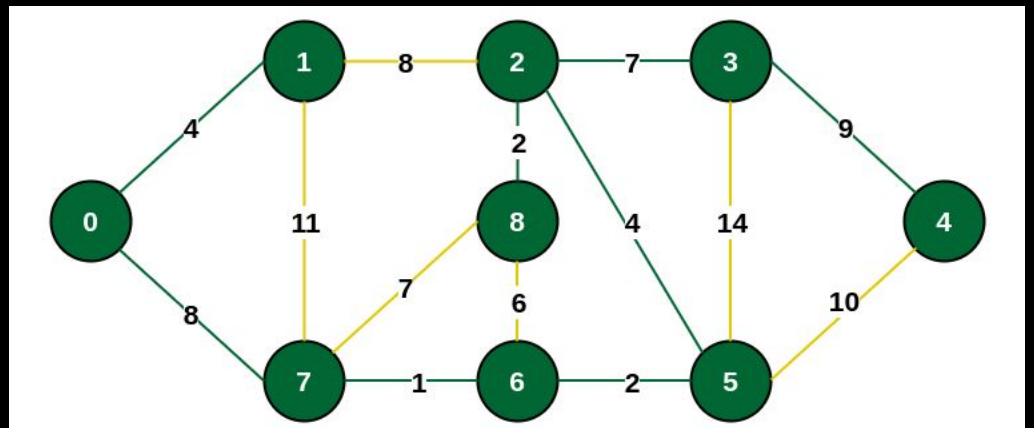
- Step 8: See here that the edges {7, 8} and {2, 3} both have same weight which are minimum. But 7 is already part of MST. So we will consider the edge {2, 3} and include that edge and vertex 3 in the MST.



https://www.geeksforgeeks.org/prims-minimum-spanning-tree-mst-greedy-algo-5/

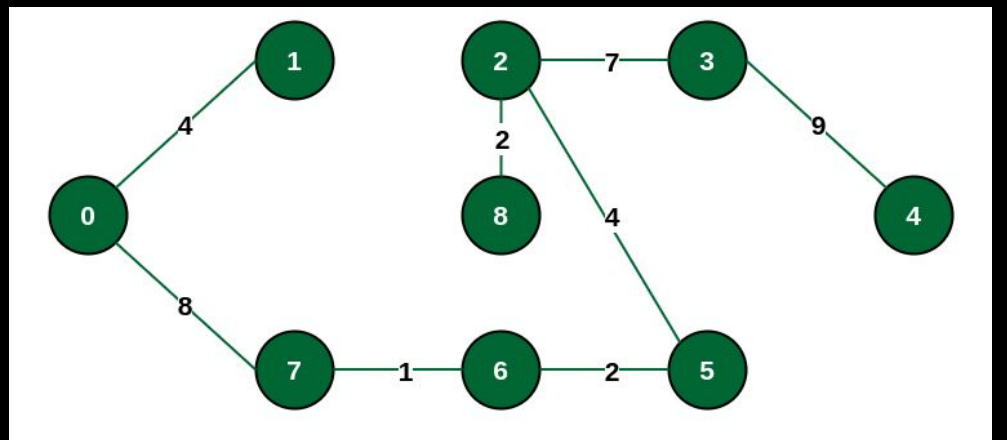
Prim's Algorithm ctd..

- Step 9: Only the vertex 4 remains to be included. The minimum weighted edge from the incomplete MST to 4 is {3, 4}.



Minimum weighted edge from MST to other vertices is 3-4 with weight 9

- The final structure of the MST is as follows and the weight of the edges of the MST is (4 + 8 + 1 + 2 + 4 + 2 + 7 + 9) = 37.

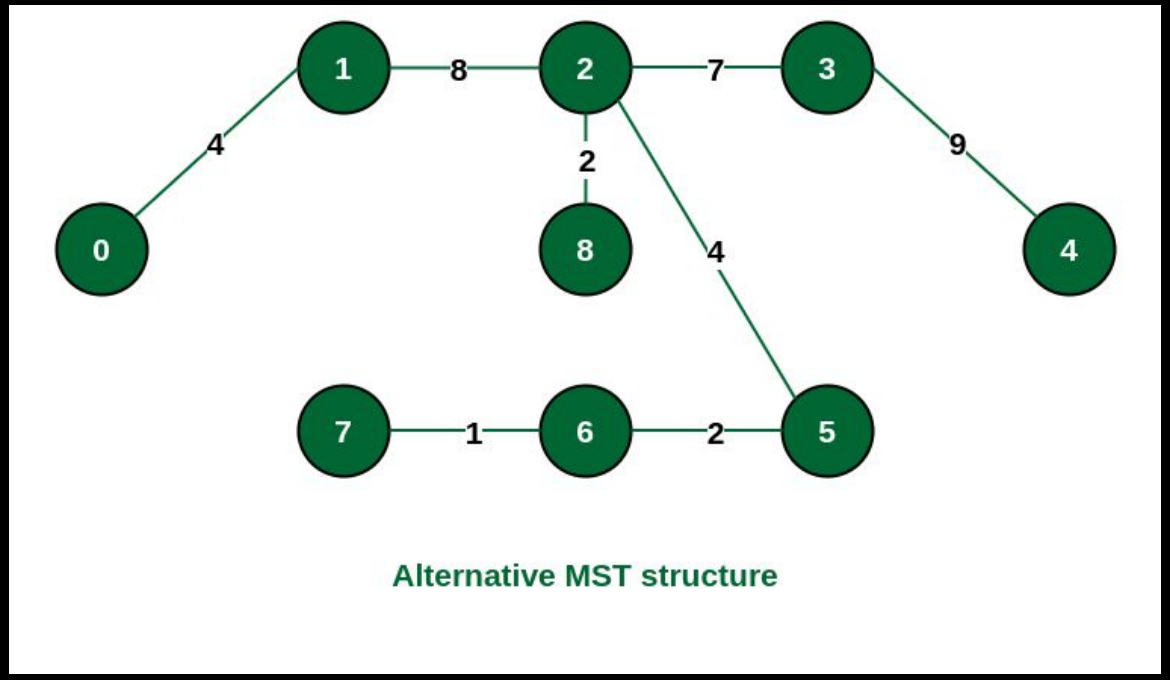


The final structure of MST

minimum-spanning-tree-mst-greedy-algo-5/

Prim’s Algorithm ctd..

- Note: If we had selected the edge {1, 2} in the third step then the MST would look like the following.



https://www.geeksforgeeks.org/prims-minimum-spanning-tree-mst-greedy-algo-5/

How to implement Prim's Algorithm?

- Follow the given steps to utilize the Prim's Algorithm mentioned above for finding MST of a graph:
  - Create a set mstSet that keeps track of vertices already included in MST.
  - Assign a key value to all vertices in the input graph. Initialize all key values as INFINITE. Assign the key value as 0 for the first vertex so that it is picked first.
  - While mstSet doesn't include all vertices
    1. Pick a vertex u that is not there in mstSet and has a minimum key value.
    2. Include u in the mstSet.
    3. Update the key value of all adjacent vertices of u. To update the key values, iterate through all adjacent vertices.
       - For every adjacent vertex v, if the weight of edge u-v is less than the previous key value of v, update the key value as the weight of u-v.
- The idea of using key values is to pick the minimum weight edge from the cut. The key values are used only for vertices that are not yet included in MST, the key value for these vertices indicates the minimum weight edges connecting them to the set of vertices included in MST.

https://www.geeksforgeeks.org/prims-minimum-spanning-tree-mst-greedy-algo-5/

Complexity Analysis of Prim's Algorithm:

- Time Complexity: $O(V^2)$, If the input graph is represented using an adjacency list, then the time complexity of Prim's algorithm can be reduced to O(E * logV) with the help of a binary heap.
- In this implementation, we are always considering the spanning tree to start from the root of the graph
- Auxiliary Space: O(V)

https://www.geeksforgeeks.org/prims-minimum-spanning-tree-mst-greedy-algo-5/

Complexity of Prim's algorithm

The running time of the prim's algorithm depends upon using the data structure for the graph and the ordering of edges. Below table shows some choices -

Time Complexity

Data structure used for the minimum edge weight	Time Complexity
Adjacency matrix, linear searching	$O(\|V\|^2)$
Adjacency list and binary heap	$O(\|E\| \log \|V\|)$
Adjacency list and Fibonacci heap	$O(\|E\|+ \|V\| \log \|V\|)$

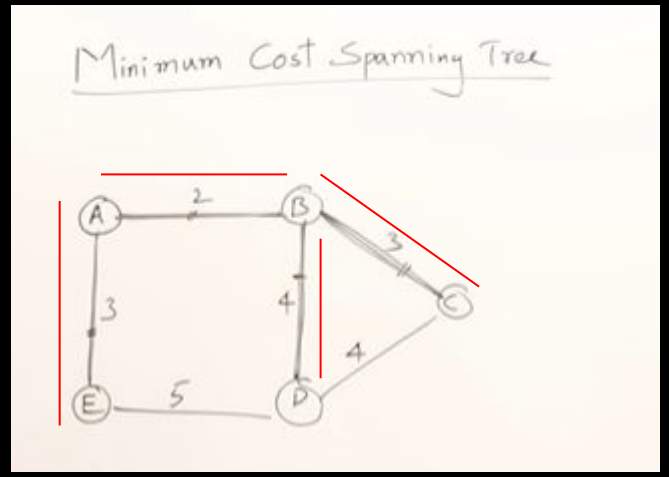
Advantages and Disadvantages of Prim's algorithm

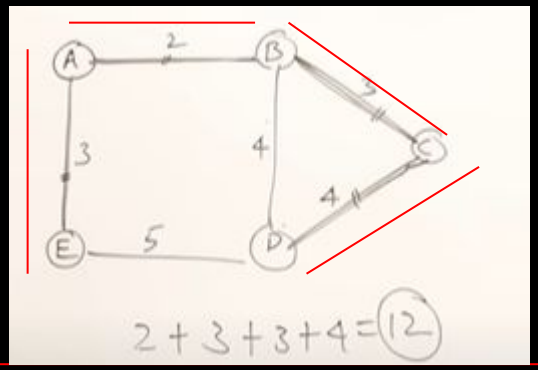
- Advantages:
  - Prim's algorithm is guaranteed to find the MST in a connected, weighted graph.
  - It has a time complexity of O(E log V) using a binary heap or Fibonacci heap, where E is the number of edges and V is the number of vertices.
  - It is a relatively simple algorithm to understand and implement compared to some other MST algorithms.
- Disadvantages:
  - Like Kruskal's algorithm, Prim's algorithm can be slow on dense graphs with many edges, as it requires iterating over all edges at least once.
  - Prim's algorithm relies on a priority queue, which can take up extra memory and slow down the algorithm on very large graphs.
  - The choice of starting node can affect the MST output, which may not be desirable in some applications.

https://www.geeksforgeeks.org/prims-minimum-spanning-tree-mst-greedy-algo-5/

MST

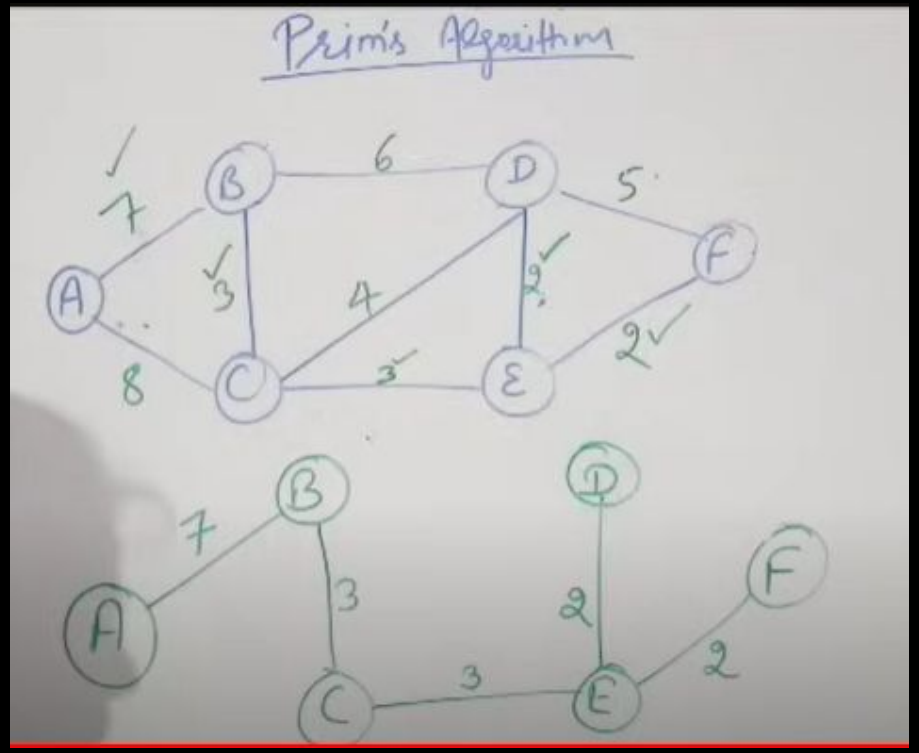
- MST is an optimization problems. Hence each tree can only have one solution for cost of MST is 3+2+3+4 =12. There can be more than one tree but min cost is one eg below (Same cost). Two MST with same minimal cost below





MST for Prims Algorithm

- Kruskals Algo selected min cost vertex first. Prims Algo selects any vertex.



Problem 1:

- You are given an array of coins (e.g., 1 cent, 5 cents, 10 cents, 25 cents) and a target in cents. Find the minimum number of coins needed to make the target amount.

- Greedy access:

i. Start with the largest value less than or equal to the target amount.

ii. Continue subtracting the largest possible multiple of that denomination from the target amount if the target amount is greater than 0.

iii. Move to the next smaller denomination and repeat the process until the target amount is 0.

https://www.javatpoint.com/greedy-algorithm-example

Problem 1:

- Example:
- Let's say we have nominal coins: 1 cent, 5 cents, 10 cents, and 25 cents, and we want to change 63 cents.

- Greedy solution:

1. Start with the largest value, 25 cents. We can use two of the 25 cents, which leaves us with 13 cents in change.
2. The next largest value, 10 cents, can be used once, leaving us with 3 cents.
3. The remaining amount is less than the smallest value, 5 cents, so we use three 1 cents to complete the exchange.

https://www.javatpoint.com/greedy-algorithm-example

Problem 1:

- **Result:**
- The minimum amount to make 63 cents is 6 coins (two 25 cents, one 10 cents, and three 1 cents).
- It is important to note that the greedy algorithm only sometimes works for all optimization problems and may not find the globally optimal solution in all cases.

- For certain problems, such as the coin toss problem, where the greedy choice property holds (i.e., making a locally optimal choice at each step leads to a globally optimal solution), the greedy approach works well and provides an efficient solution.

https://www.javatpoint.com/greedy-algorithm-example

Problem 2:

- You are given a series of tasks, each with a start and end time. The goal is to select the maximum number of non-overlapping tasks that can be completed.

- Greedy access:

i. Sort tasks by their due dates in ascending order.

ii. Scroll through the sorted task list and select the task with the earliest due date that does not overlap with previously selected tasks.

https://www.javatpoint.com/greedy-algorithm-example

Problem 2:

- Example:
- Let's have the following tasks with start and end times.
  - Task 1: (start time: 1, end time: 4)
  - Task 2: (start time: 3, end time: 5)
  - Task 3: (start time: 0, end time: 6)
  - Task 4: (start time: 5, end time: 7)
  - Task 5: (start time: 3, end time: 9)
  - Task 6: (start time: 5, end time: 9)
  - Task 7: (start time: 6, end time: 10)

https://www.javatpoint.com/greedy-algorithm-example

Problem 2:

- Greedy solution:

i. Sort tasks by due date in ascending order: Task 1, Task 2, Task 5, Task 3, Task 4, Task 6, Task 7.

ii. Start with task 1 (earliest finish time: 4). Go to task 5 (earliest finish time after task 1: 9) because task 2 overlaps with task 1.

iii. Go to task 7 (earliest finish time after task 5: 10) because tasks 3 and 4 overlap with tasks 1 and 5.

- Result:
- A maximum of 3 non-overlapping missions can be completed, and the selected missions are Mission 1, Mission 5, and Mission 7. The greedy algorithm for the interval scheduling problem works because choosing the tasks with the earliest finish times ensures more time for other tasks. By repeatedly selecting the task with the earliest completion time that does not coincide with the previously selected tasks, we can find an optimal solution to this problem.

https://www.javatpoint.com/greedy-algorithm-example

Problem 3:

- You get a set of items, each with weight and value, and a backpack with the largest carrying capacity. The goal is to fill the backpack with items to maximize the total value. Unlike the classic 0/1 backpack problem, you can still pick up some items.

- Greedy access:

1. Calculate the value-to-weight ratio (value divided by weight) of each product.
2. Sort products by their value-to-weight ratio in descending order.
3. Add items to the backpack in the order of the ranked list until the backpack is full.
4. If an item doesn't quite fit, take part of it to fill the remaining capacity.

https://www.javatpoint.com/greedy-algorithm-example

Problem 3:

- Example:
- Suppose you have the following items with weights and values:
- Item 1. (Weight: 10, Value: 60)
- Item 2: (Weight: 20, Value: 100)
- Item 3: (Weight: 30, Value: 120)
- And the maximum carrying capacity of the backpack is 50.

https://www.javatpoint.com/greedy-algorithm-example

Problem 3:

- Greedy solution:

i. Calculate the value and weight ratios.
ii. Sort the products in descending order by the value-to-weight ratio: item 1, item 2, item 3.
iii. Add items to your backpack.
iv. Take Full Item 1 (Weight: 10, Value: 60).
v. Remaining capacity: 50 - 10 = 40.
vi. Take 40% of item 2 (weight: 20, value: 100) that fits in the remaining capacity. Remaining capacity: 40 - 40% * 20 = 32.
vii. No point 3 fits in the remaining capacity (30 > 32), so we stop here.

- Result:
  - The maximum total value obtained by filling the backpack with partial goods is 60 (from 1) 40% * 100 (from 2) = 60 40 = 100.
  - The greedy algorithm of the fractional knapsack problem works because it focuses on selecting items with the highest value-to-weight ratio, ensuring that the most valuable items are included as much as possible, even in fractions, if necessary.

https://www.javatpoint.com/greedy-algorithm-example

- Refer :
- Prim's Algorithm

https://www.w3schools.com/dsa/dsa_algo_mst_prim.php

- Prims Algorithm to Find Minimum Spanning Tree of a Graph | Algorithm with Pseudo Code | Logic First

  https://www.youtube.com/watch?v=lg-HC6Q7kEE