

Content

1. Union-find algorithm
 - a. Description
 - b. Implementation
2. Kruskal's Algorithm
 - a. What is Minimum Spanning Tree?
 - b. Steps for finding MST using Kruskal's algorithm
 - c. Implementation
3. Prim's algorithm
 - a. How does Prim's Algorithm Work?
 - b. Algorithm
 - c. How to implement?
 - d. Implementation

1. Union-find algorithm

A disjoint-set data structure is a data structure that keeps track of a set of elements partitioned into a number of disjoint (non-overlapping) subsets. A union-find algorithm is an algorithm that performs two useful operations on such a data structure:

Find: Determine which subset a particular element is in. This can be used for determining if two elements are in the same subset.

Union: Join two subsets into a single subset. Here first we have to check if the two subsets belong to same set. If no, then we cannot perform union.

In this post, we will discuss the application of Disjoint Set Data Structure. The application is to check whether a given graph contains a cycle or not.

Union-Find Algorithm can be used to check whether an undirected graph contains cycle or not. Note that we have discussed an algorithm to detect cycle.

This is another method based on Union-Find. This method assumes that the graph doesn't contain any self-loops.

We can keep track of the subsets in a 1D array, let's call it parent[[]].

Let us consider the following graph:

For each edge, make subsets using both the vertices of the edge. If both the vertices are in the same subset, a cycle is found.

Initially, all slots of parent array are initialized to -1 (means there is only one item in every subset).

```
0  1  2
```

```
-1 -1 -1
```

Now process all edges one by one.

Edge 0-1: Find the subsets in which vertices 0 and 1 are. Since they are in different subsets, we take the union of them. For taking the union, either make node 0 as parent of node 1 or vice-versa.

```
0  1  2  <----- 1 is made parent of 0 (1 is now representative of subset {0, 1})
```

```
1 -1 -1
```

Edge 1-2: 1 is in subset 1 and 2 is in subset 2. So, take union.

```
0  1  2  <----- 2 is made parent of 1 (2 is now representative of subset {0, 1, 2})
```

```
1  2 -1
```

Edge 0-2: 0 is in subset 2 and 2 is also in subset 2. Hence, including this edge forms a cycle.

How subset of 0 is same as 2?

0->1->2 // 1 is parent of 0 and 2 is parent of 1

Implementation

<https://github.com/lavishabhambri/Weekly-Algo-Newsletter/blob/main/Graphs/Codes/unionFindAlgorithm.cpp>

2. Kruskal's Algorithm

What is Minimum Spanning Tree?

Given a connected and undirected graph, a spanning tree of that graph is a subgraph that is a tree and connects all the vertices together. A single graph can have many different spanning trees. 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. The weight of a spanning tree is the sum of weights given to each edge of the spanning tree.

How many edges does a minimum spanning tree has?

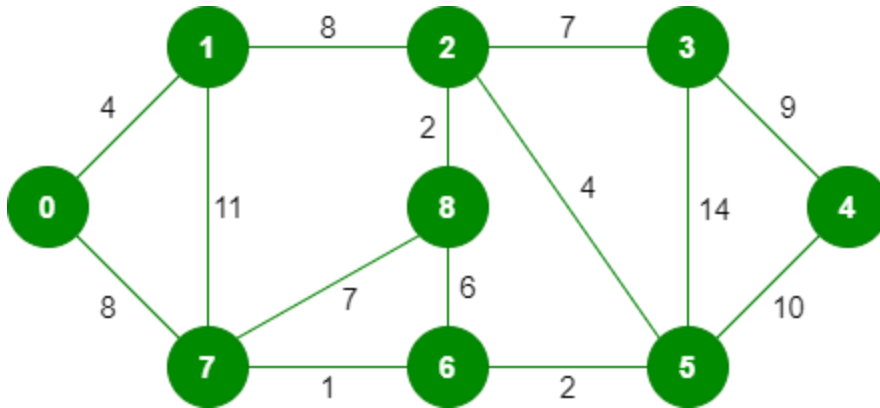
A minimum spanning tree has $(V - 1)$ edges where V is the number of vertices in the given graph.

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 cycle is not formed, include this edge. Else, discard it.
3. Repeat step#2 until there are $(V-1)$ edges in the spanning tree.

Step #2 uses the Union-find algorithm to detect cycles. So we recommend reading the following post as a prerequisite.

The algorithm is a Greedy Algorithm. 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: Consider the below input graph.



The graph contains 9 vertices and 14 edges. So, the minimum spanning tree formed will have $(9 - 1) = 8$ edges.

After sorting:

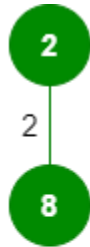
Weight	Src	Dest
1	7	6
2	8	2
2	6	5
4	0	1
4	2	5
6	8	6
7	2	3
7	7	8
8	0	7
8	1	2
9	3	4
10	5	4
11	1	7
14	3	5

Now pick all edges one by one from the sorted list of edges

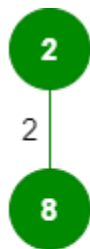
1. Pick edge 7-6: No cycle is formed, include it.



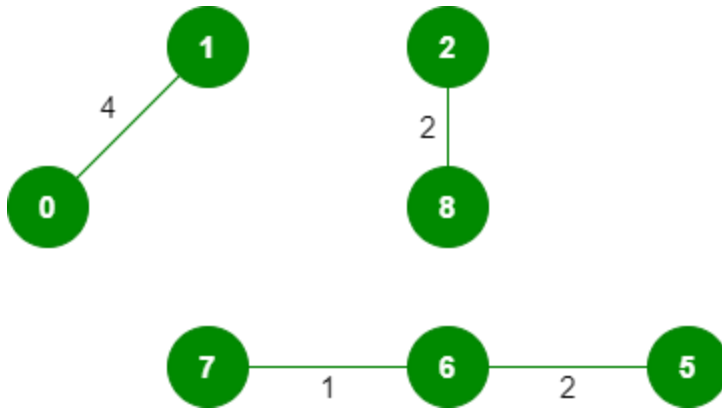
2. Pick edge 8-2: No cycle is formed, including it.



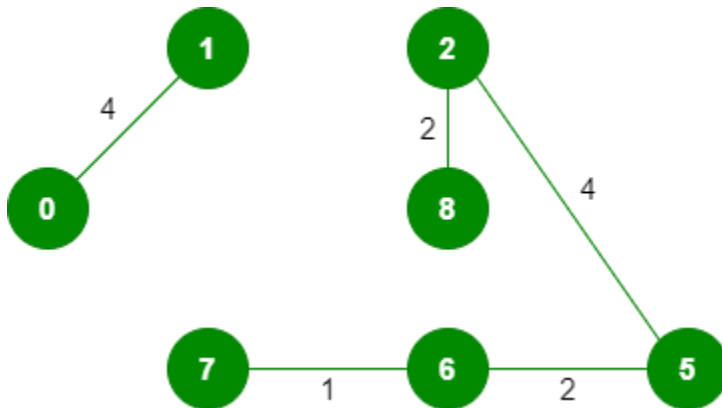
3. Pick edge 6-5: No cycle is formed, include it.



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

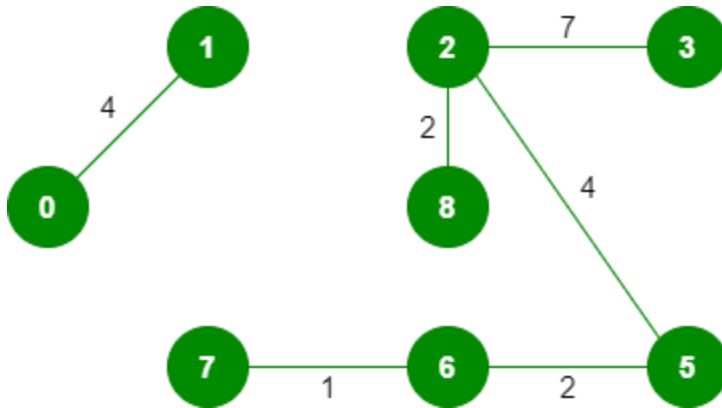


5. Pick edge 2-5: No cycle is formed, include it.



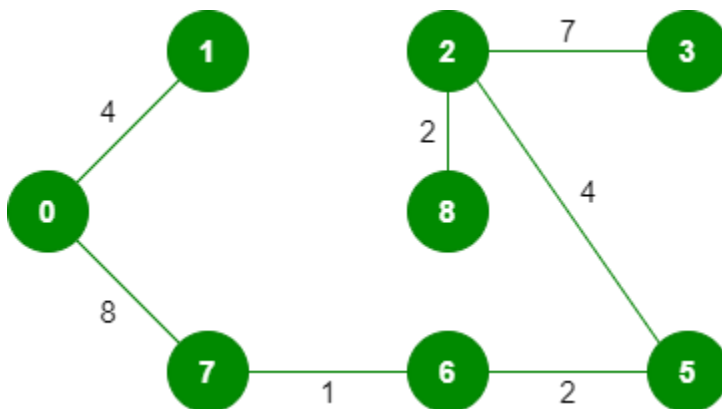
6. Pick edge 8-6: Since including this edge results in the cycle, discard it.

7. Pick edge 2-3: No cycle is formed, include it.



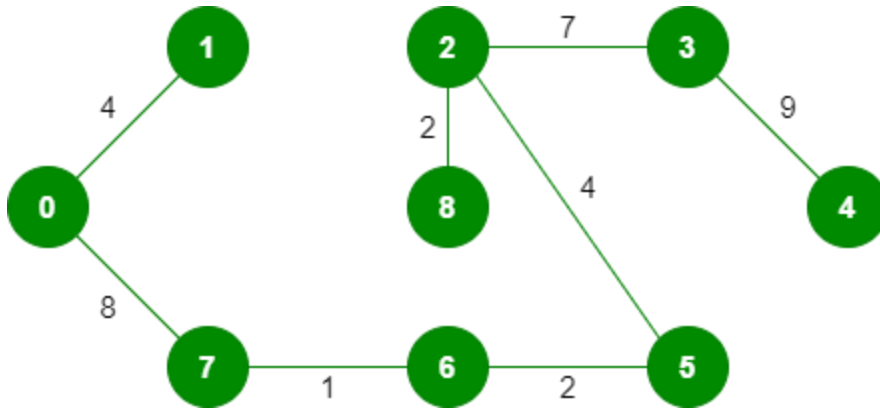
8. Pick edge 7-8: Since including this edge results in the cycle, discard it.

9. Pick edge 0-7: No cycle is formed, include it.



10. Pick edge 1-2: Since including this edge results in the cycle, discard it.

11. Pick edge 3-4: No cycle is formed, include it.



Since the number of edges included equals $(V - 1)$, the algorithm stops here.

Implementation

<https://github.com/lavishabhambri/Weekly-Algo-Newsletter/blob/main/Graphs/Codes/kruskalsAlgorithm.cpp>

3. Prim's algorithm

Like Kruskal's algorithm, Prim's algorithm is also a Greedy algorithm. It starts with an empty spanning tree. The idea is to maintain two sets of vertices. The first set contains the vertices already included in the MST, the other set contains the vertices not yet included. At every step, it considers all the edges that connect the two sets, and picks the minimum weight edge from these edges. After picking the edge, it moves the other endpoint of the edge to the set containing MST.

A group of edges that connects two set of vertices in a graph is called cut in graph theory. So, at every step of Prim's algorithm, we find a cut (of two sets, one contains the vertices already included in MST and other contains rest of the vertices), pick the minimum weight edge from the cut and include this vertex to MST Set (the set that contains already included vertices).

How does Prim's Algorithm Work?

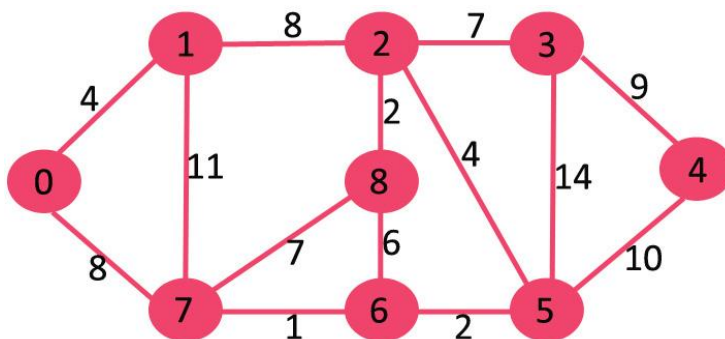
The idea behind Prim's algorithm is simple, a spanning tree means all vertices must be connected. So the two disjoint subsets (discussed above) of vertices must be connected to make a Spanning Tree. And they must be connected with the minimum weight edge to make it a Minimum Spanning Tree.

Algorithm

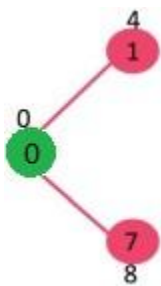
- 1) Create a set *mstSet* that keeps track of vertices already included in MST.
- 2) Assign a key value to all vertices in the input graph. Initialize all key values as INFINITE. Assign key value as 0 for the first vertex so that it is picked first.
- 3) While *mstSet* doesn't include all vertices
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

The idea of using key values is to pick the minimum weight edge from cut. The key values are used only for vertices which are not yet included in MST, the key value for these vertices indicate the minimum weight edges connecting them to the set of vertices included in MST.

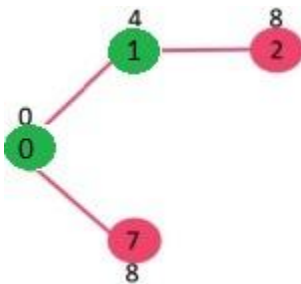
Let us understand with the following example:



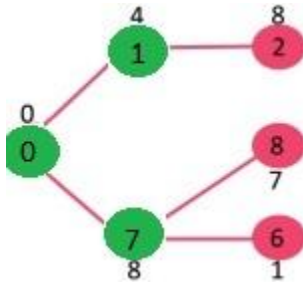
The set *mstSet* is initially empty and keys assigned to vertices are {0, INF, INF, INF, INF, INF, INF, INF} where INF indicates infinite. Now pick the vertex with the minimum key value. The vertex 0 is picked, include it in *mstSet*. So *mstSet* becomes {0}. After including to *mstSet*, update key values of adjacent vertices. Adjacent vertices of 0 are 1 and 7. The key values of 1 and 7 are updated as 4 and 8. Following subgraph shows vertices and their key values, only the vertices with finite key values are shown. The vertices included in MST are shown in green color.



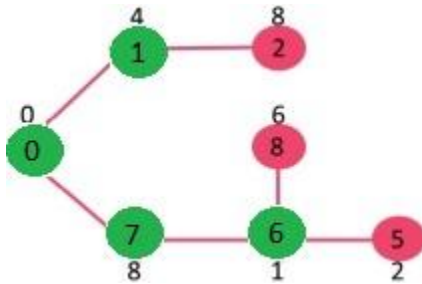
Pick the vertex with minimum key value and not already included in MST (not in *mstSet*). The vertex 1 is picked and added to *mstSet*. So *mstSet* now becomes {0, 1}. Update the key values of adjacent vertices of 1. The key value of vertex 2 becomes 8.



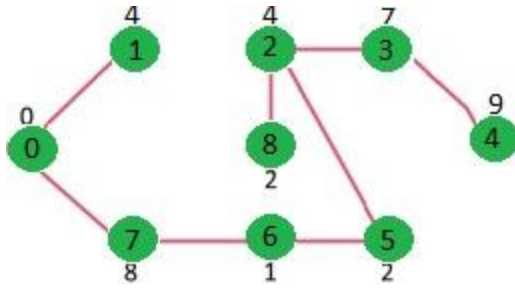
Pick the vertex with minimum key value and not already included in MST (not in *mstSet*). We can either pick vertex 7 or vertex 2, let vertex 7 is picked. So *mstSet* now becomes {0, 1, 7}. Update the key values of adjacent vertices of 7. The key value of vertex 6 and 8 becomes finite (1 and 7 respectively).



Pick the vertex with minimum key value and not already included in MST (not in *mstSet*). Vertex 6 is picked. So *mstSet* now becomes {0, 1, 7, 6}. Update the key values of adjacent vertices of 6. The key value of vertex 5 and 8 are updated.



We repeat the above steps until *mstSet* includes all vertices of given graph. Finally, we get the following graph.



How to implement the above algorithm?

We use a boolean array `mstSet[]` to represent the set of vertices included in MST. If a value `mstSet[v]` is true, then vertex `v` is included in MST, otherwise not. Array `key[]` is used to store key values of all vertices. Another array `parent[]` to store indexes of parent nodes in MST. The parent array is the output array which is used to show the constructed MST.

Implementation

<https://github.com/lavishabhambri/Weekly-Algo-Newsletter/blob/main/Graphs/Codes/primsAlgorithm.cpp>