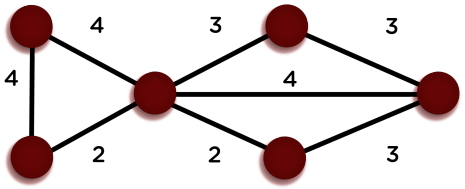
Prim's Algorithm

This algorithm is used to find MST for a given undirected-weighted graph (which can also be achieved using Kruskal's Algorithm).

In this algorithm, the MST is built by adding one edge at a time. In the beginning, the spanning tree consists of only one vertex, which is chosen arbitrarily from the set of all vertices of the graph. Then the minimum weighted edge, outgoing from this vertex, is selected and simultaneously inserted into the MST. Now, the tree contains two edges. Further, we will be selecting the edge with the minimum weight such that one end is already present there in the MST and the other one from the unselected set of vertices. This process is repeated until we have inserted a total of (n-1) edges in the MST.

Consider the following example for a better understanding.

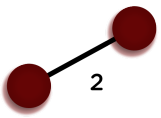


Step: 1

Start with a weighted graph

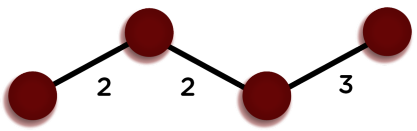
Step: 2

Choose a vertex



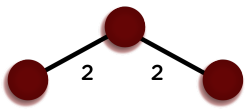
Step: 3

Choose the shortest edge from this vertex add it

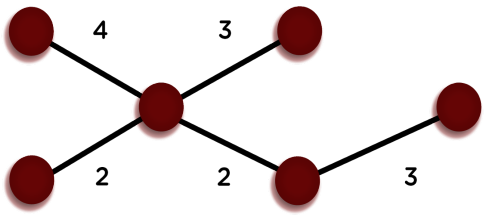


Step: 5

Choose the nearest edge not yet in the solution,
if there are multiple choices, choose one at random



Step: 4
Choose the nearest vertex not yet in the solution



Step: 6

Repeat until you have a spanning tree

Implementation:

- We are considering the starting vertex to be 0 with a parent equal to -1, and weight is equal to 0 (The weight of the edge from vertex 0 to vertex 0 itself).
- The parent of all other vertices is assumed to be NIL, and the weight will be equal to infinity, which means that the vertex has not been visited yet.
- We will mark the vertex 0 as visited and the rest as unvisited. If we add any vertex to the MST, then that vertex will be shifted from the unvisited section to the visited section.
- Now, we will update the weights of direct neighbors of vertex 0 with the edge weights as these are smaller than infinity. We will also update the parent of these vertices and assign them 0 as we reached these vertices from vertex 0.
- This way, we will keep updating the weights and parents, according to the edge, which has the minimum weight connected to the respective vertex.

Time Complexity of Prim's Algorithm:
Here, n is the number of vertices, and E is the number of edges.

- The time complexity for finding the minimum weighted vertex is O(n) for each iteration. So for (n-1) edges, it becomes $O(n^2)$.
- Similarly, for exploring the neighbor vertices, the time taken is $O(n^2)$.

It means the time complexity of Prim's algorithm is $O(n^2)$. We can improve this in the following ways:

- For exploring neighbors, we are required to visit every vertex because of the adjacency matrix. We can improve this by using an adjacency list instead of a matrix.

- Now, the second important thing is the time taken to find the minimum weight vertex, which is also taking a time of $O(n^2)$. Here, out of the available list, we are trying to figure out the one with minimum weight. This can be optimally achieved using a **priority queue** where the priority will be taken as weights of the vertices. This will take O(log(n)) time complexity to remove a vertex from the priority queue.

These optimizations can lead us to the time complexity of **O((n+E)log(n))**, which is much better than the earlier one. Try to write the optimized code by yourself.