Aim: For a weighted graph G, find the minimum spanning tree using Prims algorithm

Objective: To Create a undirected graph.

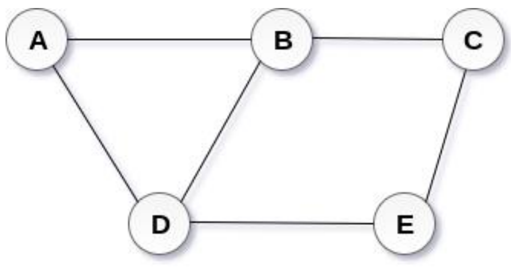
Theory:

Graph

A graph can be defined as group of vertices and edges that are used to connect these vertices. A graph can be seen as a cyclic tree, where the vertices (Nodes) maintain any complex relationship among them instead of having parent child relationship.

Definition

A graph G can be defined as an ordered set G(V, E) where V(G) represents the set of vertices and E(G) represents the set of edges which are used to connect these vertices.



Undirected Graph

Spanning Tree

Spanning tree can be defined as a sub-graph of connected, undirected graph G that is a tree produced by removing the desired number of edges from a graph. In other words, Spanning tree is a non-cyclic sub-graph of a connected and undirected graph G that connects all the vertices together. A graph G can have multiple spanning trees.

Minimum Spanning Tree

There can be weights assigned to every edge in a weighted graph. However, A minimum spanning tree is a spanning tree which has minimal total weight. In other words, minimum spanning tree is the one which contains the least weight among all other spanning tree of some particular graph.

Prim's Algorithm

Prim's Algorithm is used to find the minimum spanning tree from a graph. Prim's algorithm finds the subset of edges that includes every vertex of the graph such that the sum of the weights of the edges can be minimized.

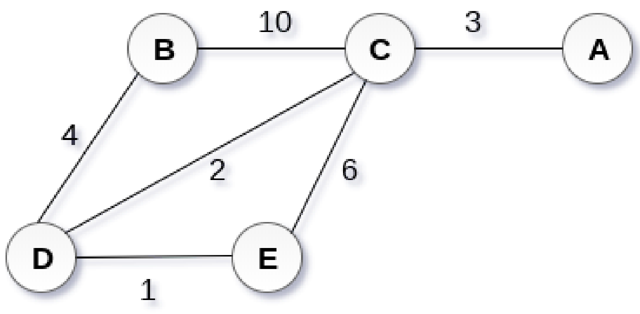
Prim's algorithm starts with the single node and explore all the adjacent nodes with all the connecting edges at every step. The edges with the minimal weights causing no cycles in the graph got selected.

Algorithm:

- **Step 1:** Select a starting vertex
- **Step 2:** Repeat Steps 3 and 4 until there are fringe vertices
- **Step 3:** Select an edge e connecting the tree vertex and fringe vertex that has minimum weight
- **Step 4:** Add the selected edge and the vertex to the minimum spanning tree T
  [END OF LOOP]
- **Step 5:** EXIT

Example :

Construct a minimum spanning tree of the graph given in the following figure by using prim's algorithm.



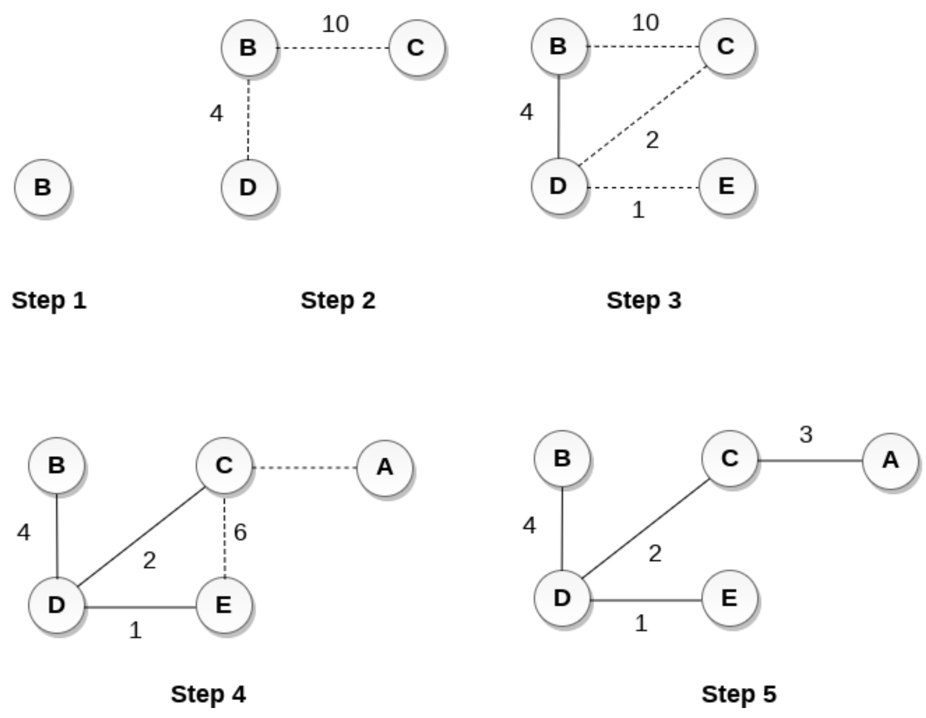
Solution

- **Step 1 :** Choose a starting vertex B.
- **Step 2:** Add the vertices that are adjacent to A. the edges that connecting the vertices are shown by dotted lines.
- **Step 3:** Choose the edge with the minimum weight among all. i.e. BD and add it to MST. Add the adjacent vertices of D i.e. C and E.
- **Step 3:** Choose the edge with the minimum weight among all. In this case, the edges DE and CD are such edges. Add them to MST and explore the adjacent of C i.e. E and A.
- **Step 4:** Choose the edge with the minimum weight i.e. CA. We can't choose CE as it would cause cycle in the graph.

The graph produces in the step 4 is the minimum spanning tree of the graph shown in the above figure.

The cost of MST will be calculated as;

cost(MST) = 4 + 2 + 1 + 3 = 10 units.



Program

```
#include <iostream>
using namespace std;


class Graph
{
        private:
                int nVertices,choiceOfEdge,startVertex,endVertex;
                int **graph;
                int *visited;
```

```
public :
        Graph(int nVertices)
        {
                this ->nVertices = nVertices;
                graph = new int*[nVertices];
                for(int i=0;i<nVertices;i++)
                {
                        graph[i] = new int [nVertices];
                }

                //initialize graph matrix with 0
                for(int i=0;i<nVertices;i++)
                {
                        for(int j=0;j<nVertices;j++)
                        {
                                *(*(graph+i)+j)=0;
                        }
                }

                visited = new int[nVertices];
```

```
		}
		int** createGraph(int nVertices)
		{
			bool hasNextEdge = true;
			//input edge weights
			while(hasNextEdge)
			{
				cout << "\t1.Enter a Edge in graph" <<
endl;
				cout << "\t2.Exit" << endl;
				cin >> choiceOfEdge;
				switch(choiceOfEdge)
				{
					case 1:
						cout << "\tEnter first vertex" <<
endl;
						cin >> startVertex;
						cout << "\tEnter End vertex" <<
endl;
						cin >> endVertex;
						cout << "\tEnter edge Weight" <<
endl;
						cin >>
*(*(graph+startVertex)+endVertex);
```

```
        *(*(graph+endVertex)+startVertex) =
*(*(graph+startVertex)+endVertex);
                                        hasNextEdge = true;
                                        break;

                                case 2:
                                        hasNextEdge = false;
                        }
                }

                //assign 999 to not present edges
                for(int i=0;i<nVertices;i++)
                {
                        for(int j=0;j<nVertices;j++)
                        {
                                if(*(*(graph+i)+j) == 0)
                                        *(*(graph+i)+j) = 999;
                        }
                }
                return graph;
        }
```

```
void display()
{
        for(int i=0;i<nVertices;i++)
        {
                for(int j=0;j<nVertices;j++)
                {
                        cout << *(*(graph+i)+j) << " " ;
                }
                cout << endl;
        }
}

int minKey(int bucket[],int *visited)
{
        int min=999,minIndex;
        for(int i=0;i<nVertices;i++)
        {
                if(*(visited+i)==0 && bucket[i]<min)
                {
```

```
                min = bucket[i];
                minIndex = i;
            }
        }
        return minIndex;
    }

    void primAlgo()
    {
        int mst[nVertices];
        int bucket[nVertices];

        for(int i=0;i<nVertices;i++)
        {
            bucket[i]=999;
            *(visited+i)=0;
        }

        bucket[0]=0;
        mst[0]=-1;
```

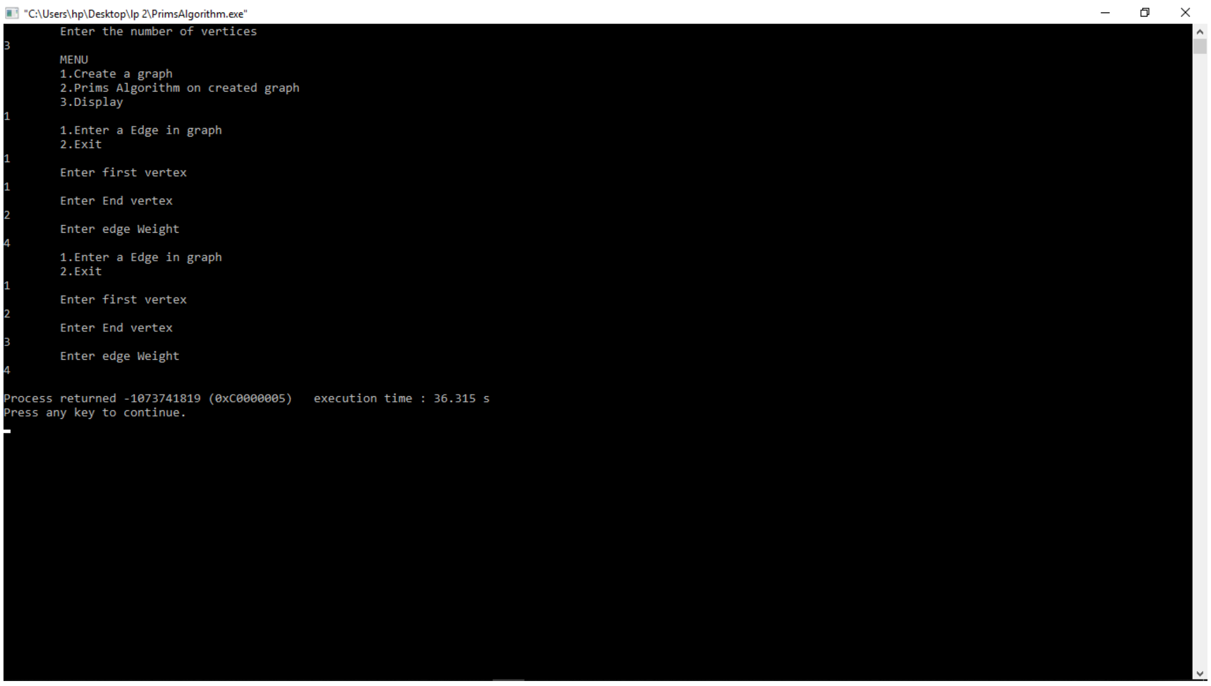
```
                for(int j=0;j<nVertices-1;j++)

                {

                        int min = minKey(bucket,visited);

                        visited[min]=1;

                        for(int k=0;k<nVertices;k++)

                        {

                                if(*(visited+k)==0 &&
*(*(graph+min)+k) < bucket[k])

                                {

                                        mst[k]=min;

                                        bucket[k]=*(*(graph+min)+k);

                                }

                        }

                }


                int cost=0;

                cout << endl;

                cout << "\t EDGE" << "\tWEIGHT" << endl;

                for(int i=1;i<nVertices;i++)

                {

                        cout << "\t" <<mst[i] << " - " << i << "\t  " <<
*(*(graph+i)+mst[i]) << endl;
```

```
                        cost += *(*(graph+i)+mst[i]);
                }
                cout << endl;
                cout << "\tThe cost of minimun spanning tree is
" << cost << endl;


        }
};




int main()
{
    int nVertices,choice;
    char ch='y';
    cout << "\tEnter the number of vertices" << endl;
    cin >> nVertices;
    Graph graph(nVertices);
    while(ch == 'y')
    {
        cout << "\tMENU"<< endl;
        cout << "\t1.Create a graph" << endl;
```

```
            cout << "\t2.Prims Algorithm on created graph" <<
endl;
            cout << "\t3.Display" << endl;
            cin >> choice;
            switch(choice)
            {
                case 1:
                    graph.createGraph(nVertices);
                    break;
                case 2:
                    graph.primAlgo();
                    break;
                case 3:
                    graph.display();
                    break;
                default:
                    cout << "\tINVALID CHOICE" << endl;
            }
            cout << "\tDo you wish to continue" << endl;
            cout << "\tenter y if yes" << endl;
            cin >> ch;
        }
```

}

Output:



```
"C:\Users\hp\Desktop\lp 2\PrimsAlgorithm.exe"
        Enter the number of vertices
3
        MENU
        1.Create a graph
        2.Prims Algorithm on created graph
        3.Display
1
        1.Enter a Edge in graph
        2.Exit
1
        Enter first vertex
1
        Enter End vertex
2
        Enter edge Weight
4
        1.Enter a Edge in graph
        2.Exit
1
        Enter first vertex
2
        Enter End vertex
3
        Enter edge Weight
4

Process returned -1073741819 (0xC0000005)   execution time : 36.315 s
Press any key to continue.
```

Conclusion:

Graphs are nonlinear data structures that make operations on large data easier.

Minimum spanning trees are useful for many real time applications where a minimum of many possible outcomes is required.