Project 3 : Prim Algorithm (100 points)

This project requires one cpp file submission. Do not upload a code snippet, the only files you submit for the code should be the source files. Do not submit any zip or compressed files, binary files, or any other generated files. Do not put the code inside the PDF file. Submission of unnecessary files or binary files in addition to source files will make you lose the points of the assignment.

The code must have the following:
a. Use consistent indentation through your code.
b. Use consistent braces style through your code.
c. File-level comments, these comments will include the file name, your name, course number, and date of last modification.
d. Function level comments include the function's description, the function's parameters, and the function's return value. These comments will appear before each of the functions, not the prototypes.
e. In function comments, these comments will include a description of the atomic functionalities within the bodies of the functions.
f. We evaluate your code using the following C++ online compiler. You will not get points if your assignment does not compile with this.

https://www.onlinegdb.com/online_c++_compiler

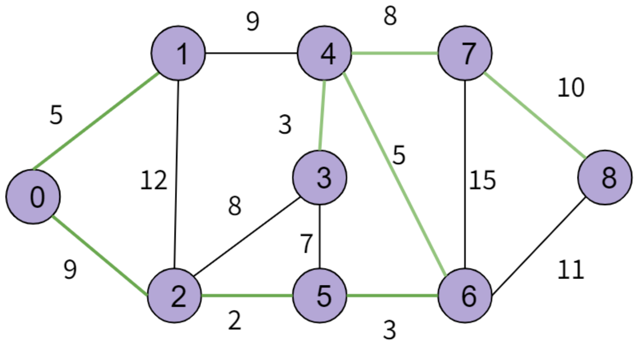
For this project, you need to study Prim algorithm very well before starting working on it.

Implement Prim's algorithm to find a Minimum Spanning Tree for an undirected, connected, weighted graph using the following class Graph.

```
class Graph
{
      // Number of vertices
      int V;

      // Adjacency list representation
      // Pair of a neighbor vertex  and a weight for every edge
      list< pair<int, int> > *adj;

public:
      Graph(int V);

      // Function to add an edge to graph
      void addEdge(int u, int v, int w);

      // Print MST using Prim's algorithm
      void primMST();
};

// Allocates memory for adjacency list
Graph::Graph(int V)
{
      this->V = V;
      adj = new list< pair<int, int>> [V];
}
```

Example:
For the graph below you need to print the list of edges in MST in the console with the following format:

```
0 - 1
0 - 2
2 - 5
5 - 6
6 - 4
4 - 7
7 - 8
```



You need to have a main function to test your program similar to the following one:

```
int main()
{
    int V = 8;
    Graph g(V);

    g.addEdge(0, 1, 5);
    g.addEdge(0, 4, 8);
    g.addEdge(1, 2, 8);
    g.addEdge(1, 7, 11);
    g.addEdge(2, 3, 7);
    g.addEdge(2, 7, 2);
    g.addEdge(3, 6, 9);
    g.addEdge(3, 5, 14);
    g.addEdge(4, 5, 10);
    g.addEdge(5, 6, 2);
    g.addEdge(6, 7, 3);

    g.primMST();

    return 0;
}
```

Hints:

- You can use `priority_queue` in STL
- You need to create a priority queue (min heap) to store vertices that are being in MST
  `priority_queue< pair<int, int>, vector <pair<int, int>> , greater<pair<int, int>> > pq;`
- Some useful functions for priority_queues are: `top(), push(), pop(), and empty().` You may use these functions for the Prim implementation
- Create a vector for values and initialize all values as infinite (INF) :
  `vector<int> value(V, INF);`
- Assume that weights are less than 100. You can define INF as 1000(or any larger value).
- For inserting edges into the priority queue you need to insert them as pair of (value, vertex). You can use `make_pair(int value, int vertex)`to make a pair in STL
    - Pair is used to combine together two values that may be different in type (here we have two integers, one corresponds to the vertex index and another corresponds to the value of that vertex). Pair provides a way to store two heterogeneous objects as a single unit. It is basically used if we want to store tuples. The pair container is a simple container defined in `<utility>` header consisting of two data elements or objects.

    - The first element is referenced as ‘first’ and the second element as ‘second’ and the order is fixed (first, second).
    - Pair can be assigned, copied, and compared. The array of objects allocated in a map or hash_map is of type ‘pair’ by default in which all the ‘first’ elements are unique keys associated with their ‘second’ value objects.
    - To access the elements, you can use variable name followed by dot operator followed by the keyword first or second.

- Always store the value(value of each vertex in Prim Algorithm) corresponding to each vertex in the first element of the pair. Since STL inserts these pairs into a priority queue with respect to the first element.
- You can get the first element and the second element of a pair using first and second keywords respectively.
    - EX: `int i = pq.top().firs;` //Returns the first element of the top value in the priority queue named pq

- You need to store parent vertices. You can initialized a vector of size V with -1 to store parent array `vector<int> parent(V, -1);`
- You can keep track of vertices included in MST by a boolean vector of size V. First, initialize everything with false as follows:
  `vector<bool> inMST(V, false);`
- You need to first insert source vertex into pq and make its value as 0 then implement the rest of the algorithm.
- In the end, you need to print MST edges using parent array.

Tentative Rubric:
Implementing addEdge(int u, int v, int w) correctly: 20%
Implementing primMST() correctly: 50%
Printing MST with the correct format(see the example): 20%
Main function and other components: 10%