CSN-261
ASN-1
REPORT

NAME : RITIK JAIN
ENR. NO. : 18114068
BATCH : O3 (CSE, BTECH. IInd Yr.)

CONTENTS

- Problem Statements
- Algorithms and Data-structures
- Snapshots of Running Code

Problem Statements

Problem Statement 1 :

Write a C++ program to perform addition and multiplication of two polynomial expressions using any data structure chosen from STL. The polynomial expressions are of the form $ax^2 + bx + c$, *where* a, b and c are real constants. The inputs for $2x^2 + 5x + 6$ and $2x^3 + 5x^2 + 1x + 1$ are shown below (real constants followed by their power of x).

Input:
No. of terms in the expression: 3
Coefficient Power
2 2
5 1
6 0

No. of terms in the expression: 4
Coefficient Power
2 3
5 2
1 1
1 0

Enter 1 to add or 2 for multiply
1

Output:
2 3
7 2
6 1
7 0

Enter 1 to add or 2 for multiply
2

Output:
4 5
20 4
39 3
37 2
11 1
6 0

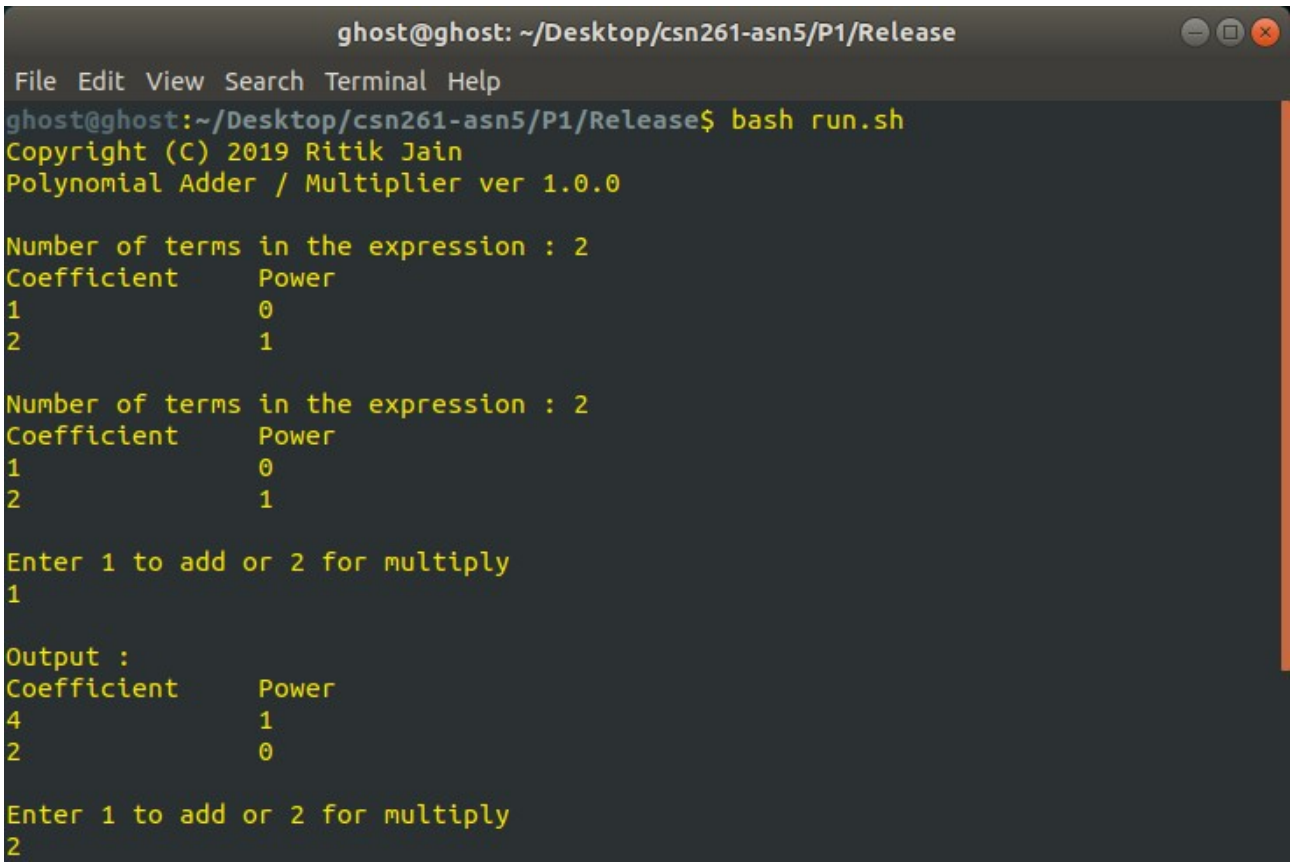
Algorithm/ Datastructures :

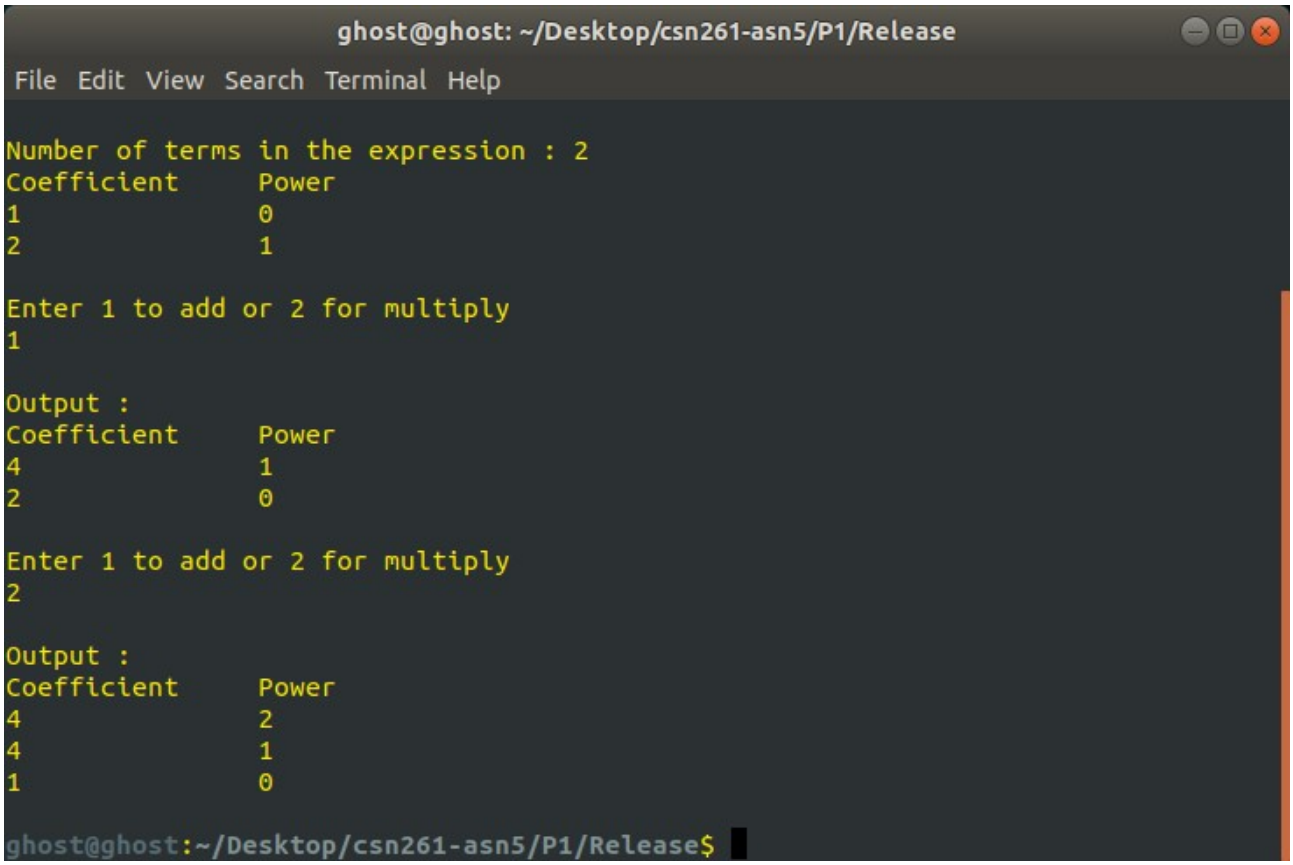
The class Polynomial in namespace asn5 is responsible for the creation and storage of polynomials. The class also provides operations on those polynomials which include the modification of coefficients, the addition of new coefficients, the addition of two polynomials and their multiplication as well.

The coefficients of a polynomial are stored in an Unordered Hash Set. Only the non-zero coefficients of the polynomial are stored inorder to decrease the space used by it. Furthermore, given the function std::hash<int>() is good enough, the search/ and or modification of coefficients has a time complexity of O(1).

The main interface provides a user-interface for the construction, addition and multiplication of polynomials.

Screenshots :





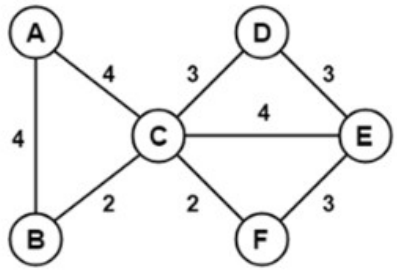
Problem Statement 2 :

Given a set of nodes connected to each other in the form of a weighted undirected graph G, find the minimum spanning tree (*MST*). A spanning tree *T* of an undirected graph *G* is a subgraph that is a tree which includes all of the vertices of *G*, with minimum possible number of edges. *G* may have more than one spanning trees. The weight of a spanning tree is the sum of weights given to each edge of the spanning tree. A *minimum spanning tree (MST)* is a spanning tree whose weight is less than or equal to that of every other spanning tree.

For given input graph (given as a CSV file having the format as shown in the example below), implement Kruskal's algorithm in C++ program using **UNION FIND** data structures (**without using STL**) and show all the edges of the *MST* as output in both the command line and in the "dot file", where DOT is a graph description language. Also, print the total edge weight of the *MST*. For

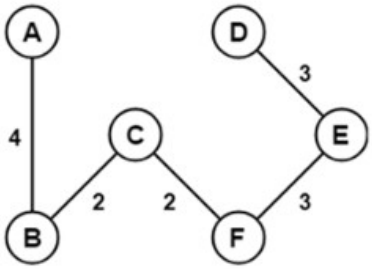
more details follow this link https://www.graphviz.org/doc/info/lang.html. Further use the “dot file” file to visualize the output graph in .pdf or .png file using **Graphviz.**

Input:



Node1 Node2 Weight
A B 4
A C 4
B C 2
C D 3
C F 2
C E 4
D E 3
F E 3

OUTPUT:
Node1 Node2 Weight
B C 2
C F 2
F E 3
D E 3
A B 4

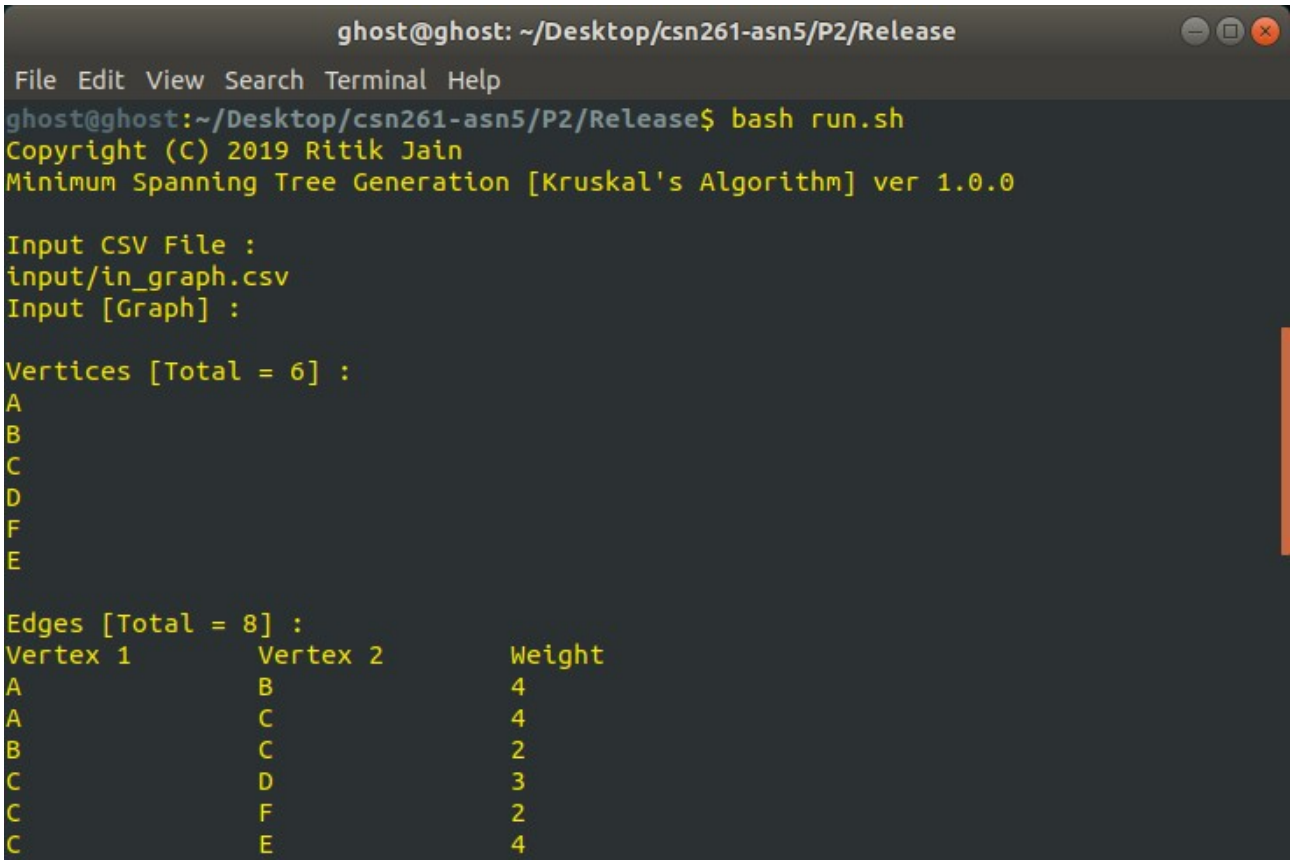


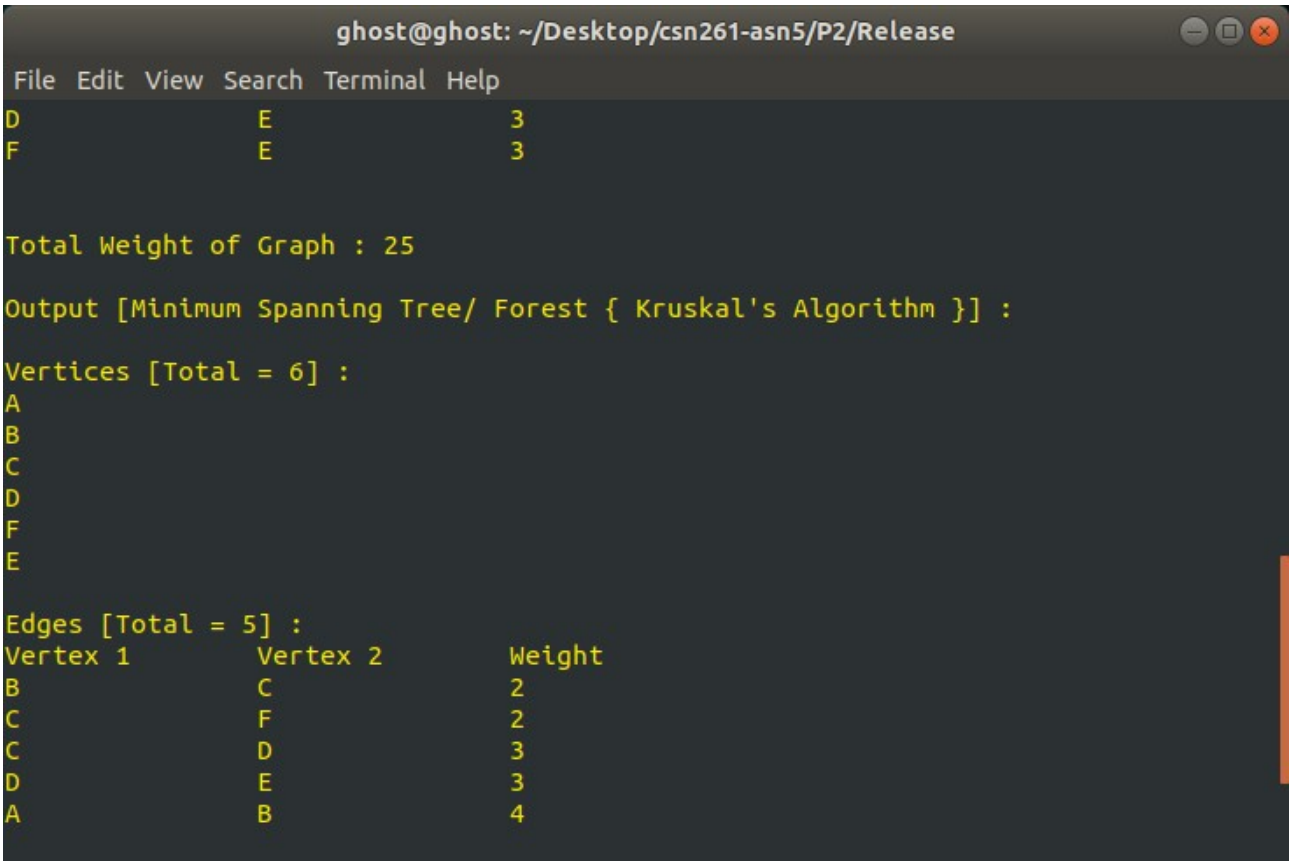
Algorithm/ Datastructures :

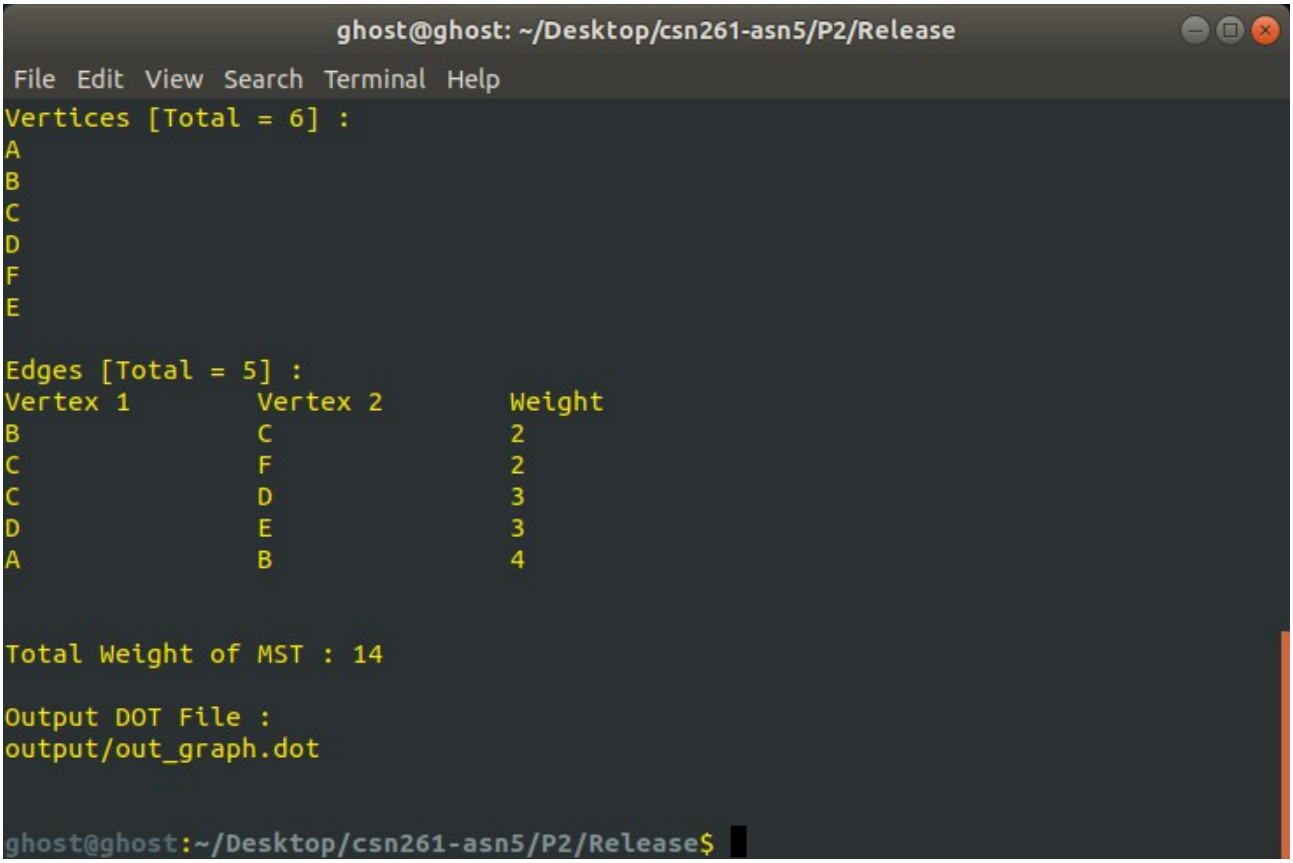
The class MSTGen in MinimumSpanningTreeKruskalUnionFindGenerator.h provides operations for loading a graph, generating its Minimum Spanning Tree using Kruskal's Algorithm with Union-Find datastructure, displaying the results and saving the MST as a 'dot' file which can then be compiled with graphwiz [dot] to convert it into required image format for visualization.

The Kruskal's algorithm is a greedy algorithm. We first sort the edges in order to their increasing weights. The minimum weight edge is selected at each iteration through the weight-ordered edeges. We initiallty generate a forest with only the vertices and no edges. Each time we iterate, the selected edge is placed into the forest. We then try to find a Cycle in the forest using the Union-Find method. The union-find method consists of two phases. The find phase : the sets which both the vertices belong to are searched. The union phase: If both the vertices do not belong to the same set, then the sets which they belong to are unioned. However, if it is found that both the vertices belong to the same, set it means we have found a cycle. Given that we don't find a cycle using union-find we include the edge (otherwise we discard the edge). The process is repeated until all the edges are processed. The final resulting tree is the Minimum Spanning Tree if the graph was connected or the Minimum Spanning Forest for a disconnected graph.

Screenshots :







Output DOT File :

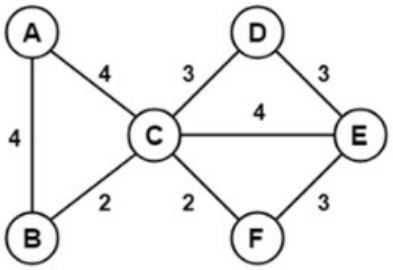
```
/**
* Copyright (C) 2019 Ritik Jain
* Graph File generated by Minimum Spanning Tree Generator ver 1.0.0
* Licensed under GNU Public License
*/

graph output
{
//Vertices
A;
B;
C;
D;
F;
E;

//Edges
B--C [label="2"];
C--F [label="2"];
C--D [label="3"];
D--E [label="3"];
A--B [label="4"];
}
//End of graph
```

Problem Statement 3 :

Write a C++ program to implement Prim's algorithm for a given input graph (given as a CSV file having the format as shown in the example below) using **Fibonacci heap** data structure to find the minimum spanning tree (*MST)*. **You can use STL** for the data structure used in this C++ program.
It is a greedy algorithm that finds a minimum spanning tree for a weighted undirected graph. This means it finds a subset of the edges that forms a tree that includes every vertex, where the total weight of all the edges in the tree is minimized. The algorithm generates the *MST* by adding one

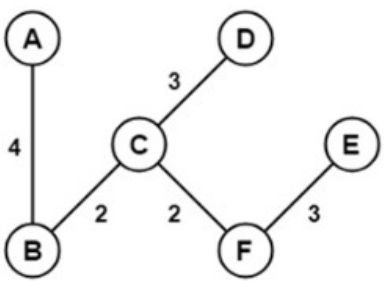
vertex at a time, starting from an arbitrary vertex. At each step the cheapest possible edge weight is chosen from the already selected vertex. These algorithms find the minimum spanning forest in a possibly disconnected graph; in contrast, the most basic form of Prim's algorithm only finds minimum spanning tree in connected graphs.
Show all the edges of the *MST* as the output in command line. Also, print the total edge weight of the MST. Use *Newick* file format (https://en.wikipedia.org/wiki/Newick_format) for visualization of the *MST* in ETE Toolkit (http://etetoolkit.org/).

Input:



Node1 Node2 Weight
A B 4
A C 4
B C 2
C D 3
C F 2
C E 4
D E 3
F E 3

OUTPUT:
A B 4
B C 2
C F 2
F E 3
C D 3

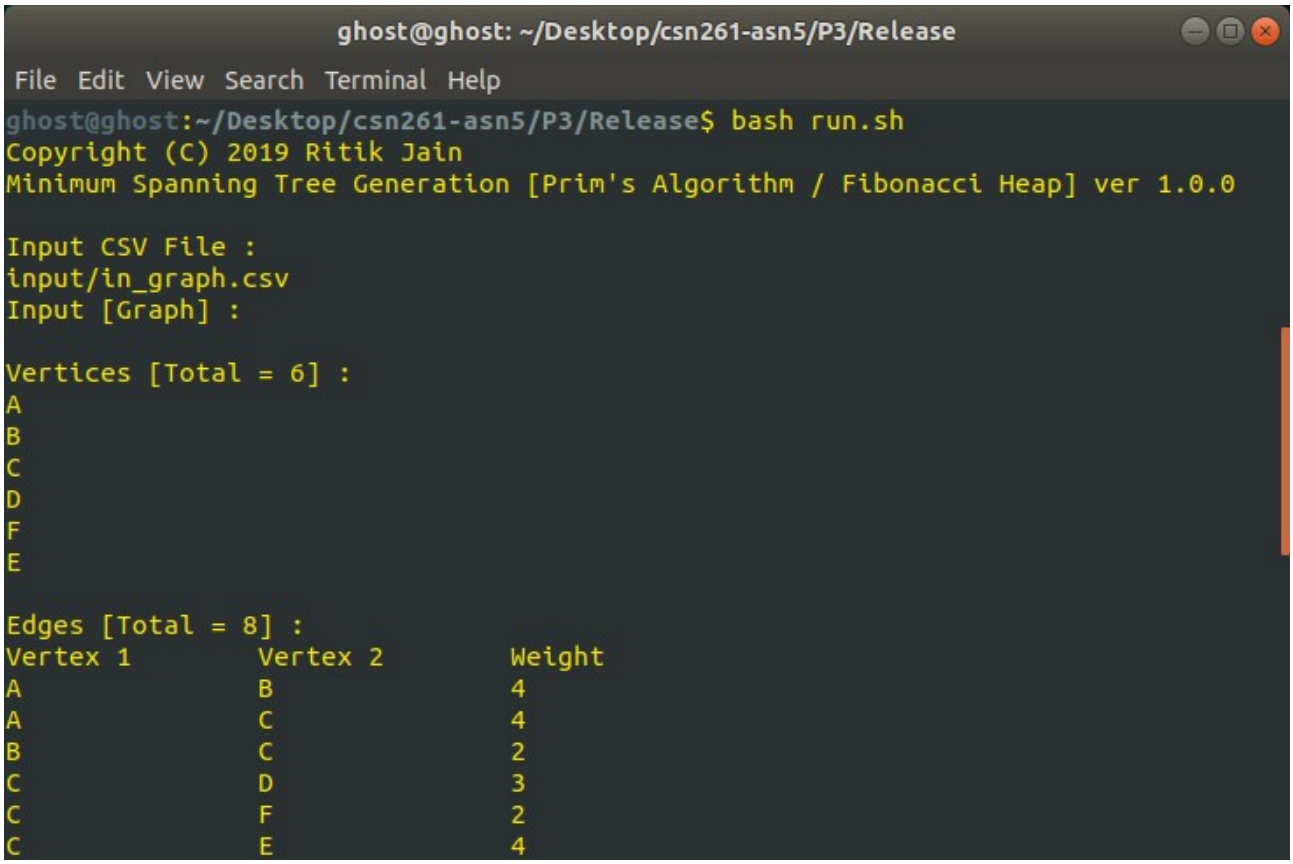


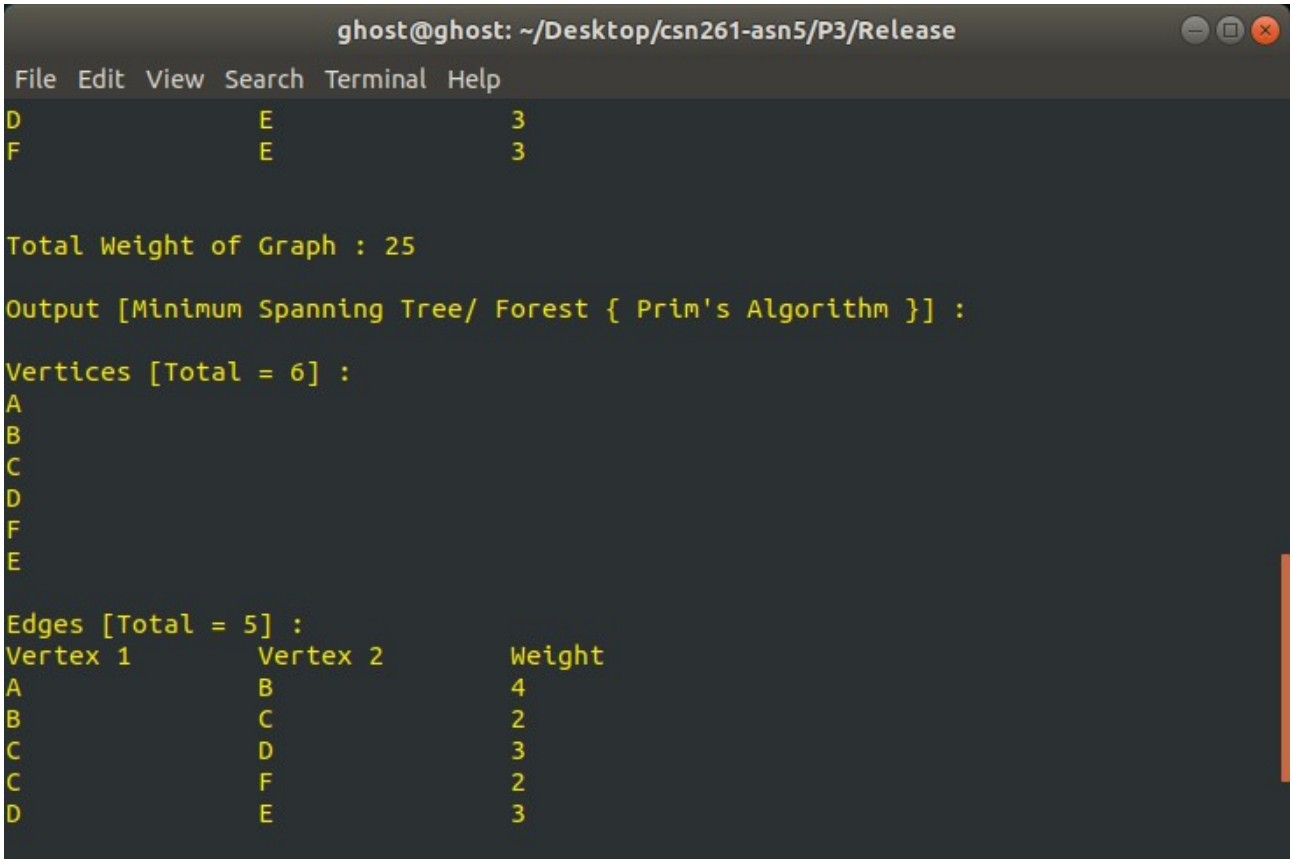
Algorithms/ Datastructures :

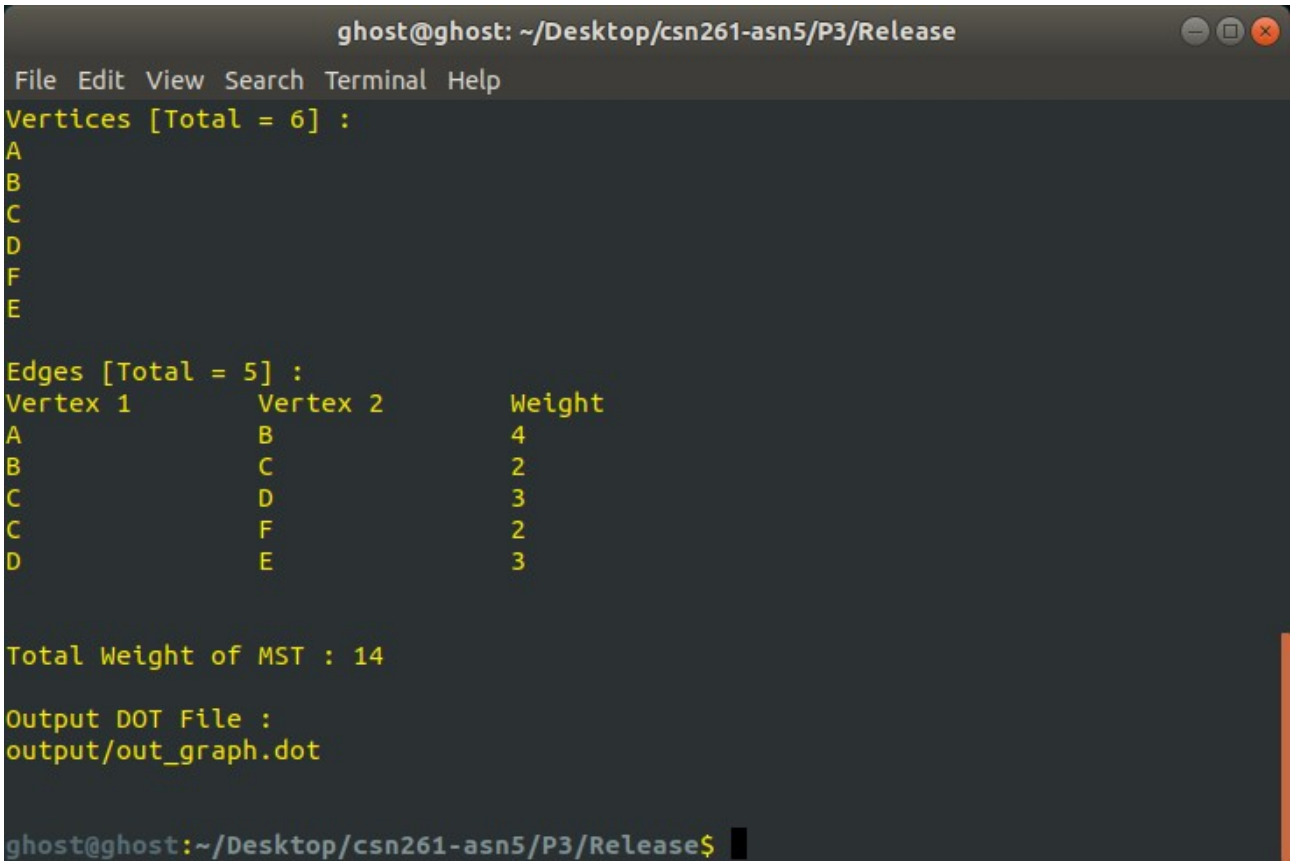
The class MinimumSpanningTreePrimFibonacci.h provides operations for loading a graph, generating its Minimum Spanning Tree using Prim's Algorithm and Fibonacci Heap Datastructure. Prim's algorithm processes each vertex and assigns its cut the keys which are nothing but the weight of the edge connecting them to their parent vertex if the new key is less than the previous key. Initially only the vertex zero is included which has no parent and all the other vertices are assigned infinite keys. Gradually their keys are updated (using cut(Vertex Zero) and other cuts). The process is repeated for each vertex and we finally find the Minimum Spanning Tree of the graph.

The fibonacci heap is an optimized heap which contains operations : insert , decreaseKey, extractMinimum , union , etc.
It facilitates efficient storage and removal of keys in Prim's algorithm.

Screenshots :







Output DOT File :

```
/**
* Copyright (C) 2019 Ritik Jain
* Graph File generated by Minimum Spanning Tree Generator ver 1.0.0
* Licensed under GNU Public License
*/

graph output
{
//Vertices
A;
B;
C;
D;
F;
E;

//Edges
A--B [label="4"];
B--C [label="2"];
C--D [label="3"];
C--F [label="2"];
D--E [label="3"];
}
//End of graph
```