CSN-261
ASSIGNMENT-5

- Name-Kavya Barnwal
- Enrollment No.-18114039
- Course-B.Tech-CSE
- Email: kbarnwal@cs.iitr.ac.in
- Contact:6205125517

==

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
```

=====================================

Data Structure And Algorithm Used :

Data Structures Used :

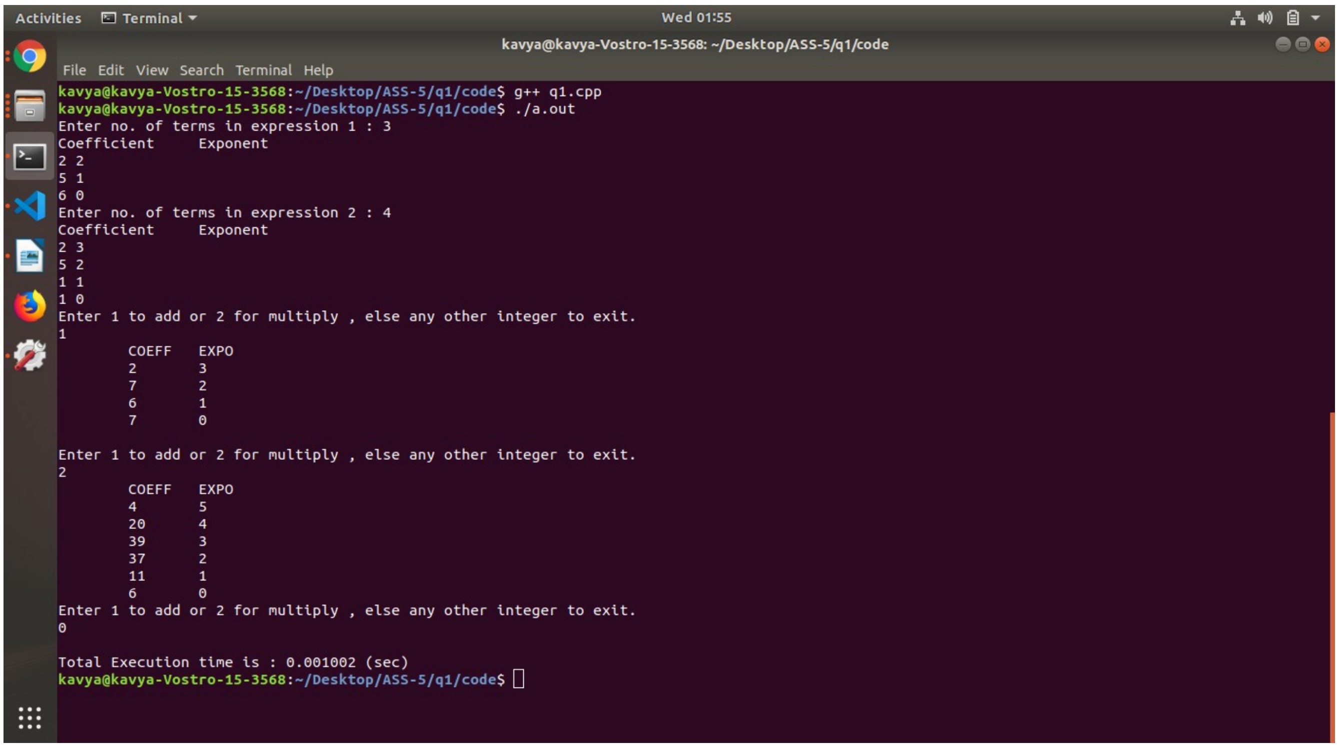
Linked List,arrays

Algorithms Used :

- Made two linked list of two polynomial expressions where each node containing the coefficient and exponent of each term.
- Added the terms with equal exponent by traversing each linked list and comparing exponent at each step and applying conditions according if first linked list head node exponent is greater than traversing the second linked list for every node

of first linked list until the exponent becomes equal or pointer becomes null and vice versa.

- Multiplied the linked list by traversing the linked list with smaller exponent of head node for every node of the other linked list and storing the values in new linked list and every time the linked list is fully traversed and starting of with the next node after multiplication cokparing the resultant exponent with the new linked list and applying insert after method so as to make the new linked list in decreasing order of exponents.



Activities
Terminal
Wed 01:55
kavya@kavya-Vostro-15-3568: ~/Desktop/ASS-5/q1/code
File Edit View Search Terminal Help

```
kavya@kavya-Vostro-15-3568:~/Desktop/ASS-5/q1/code$ g++ q1.cpp
kavya@kavya-Vostro-15-3568:~/Desktop/ASS-5/q1/code$ ./a.out
Enter no. of terms in expression 1 : 3
Coefficient     Exponent
2 2
5 1
6 0
Enter no. of terms in expression 2 : 4
Coefficient     Exponent
2 3
5 2
1 1
1 0
Enter 1 to add or 2 for multiply , else any other integer to exit.
1
        COEFF   EXPO
        2       3
        7       2
        6       1
        7       0

Enter 1 to add or 2 for multiply , else any other integer to exit.
2
        COEFF   EXPO
        4       5
        20      4
        39      3
        37      2
        11      1
        6       0
Enter 1 to add or 2 for multiply , else any other integer to exit.
0

Total Execution time is : 0.001002 (sec)
kavya@kavya-Vostro-15-3568:~/Desktop/ASS-5/q1/code$
```

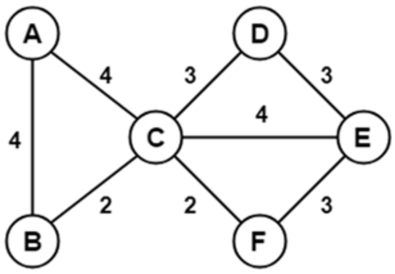
====================================

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

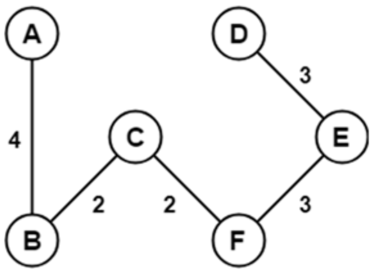
B C 2

C F 2

F E 3

D E 3

A B 4



*=======================================
==*

Data Structure And Algorithm Used :

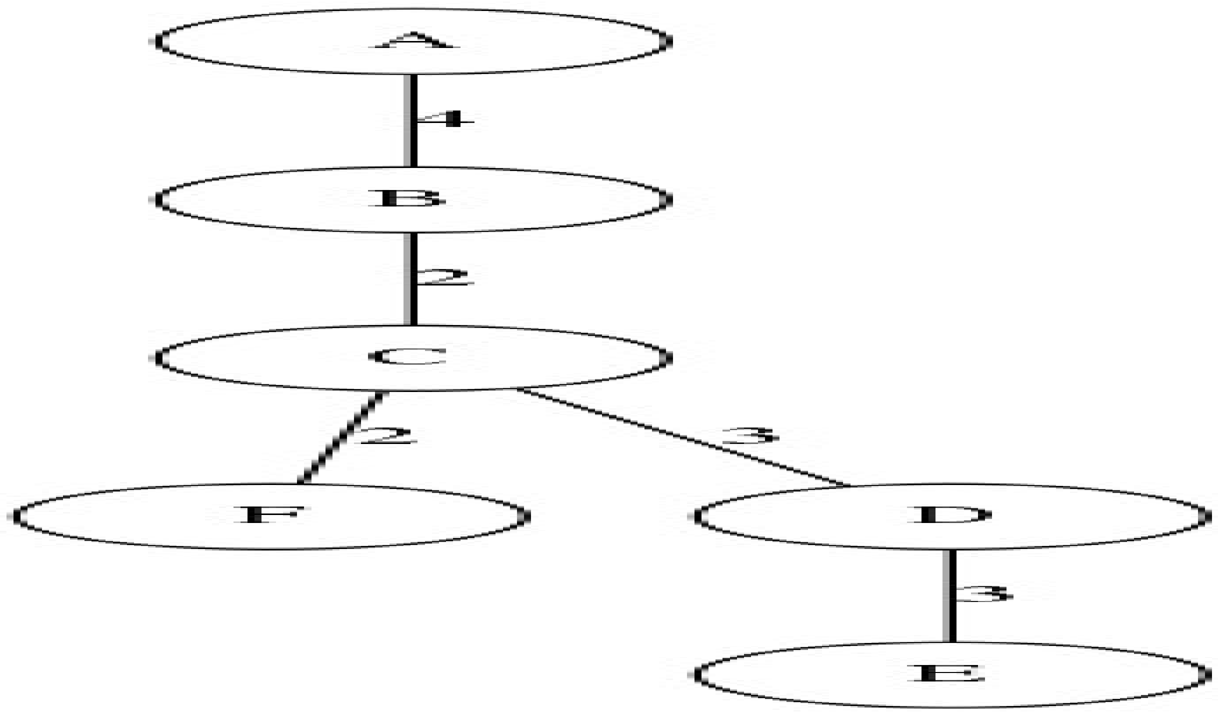
Data Structures Used :

Graph,Union-Find,arrays

Algorithms Used :

- Reading the file lin by line using getline and storing it into string then storing the string characters in character array except commas and then storing the characters as integers in another array.Making graph using the array which gives us the source ,destination and weight.
- Krushkal main function is called which uses find and union data structure implemented as methods.first it uses the qsort to sort all the edges in non-decreasing order of their weight.Creates v subsets each containing one vertex.And setting the representative of each set as the vertex itself and height 0.Calling the find and union on every edge gives the mst.
- Find-finds the representative of the set that some t is an element of if it is the parent of itself then it is itself the representative of the set else it is not and so recursively find the representative and the result is cached by moving t’s node directly under the representative of this set and then returns the result.
- Union-unites the set that includes s and the set that incudes t (the arguments).Finds the representative or the

root nodes for the set that includes s and t.If the element are in the same set no need to unite them and return else get the height of the two trees and unite the less with the larger one with root of resultant tree as root of larger and increase the height of larger by one.



===

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

==

<u>Data Structure And Algorithm Used :</u>

<u>Data Structures Used :</u>

Fibonacci Heap,arrays

<u>Algorithms Used :</u>

<u>To insert a node in a Fibonacci heap H,</u>

- Create a new node ‘x’.
- Check whether heap H is empty or not.
- If H is empty then:
- Make x as the only node in the root list.
- Set H(min) pointer to x.
- Else:
    - Insert x into root list and update H(min).

Union of two Fibonacci heaps H1 and H2 ,

- Join root lists of Fibonacci heaps H1 and H2 and make a single Fibonacci heap H.
- If H1(min) < H2(min) then:
- H(min) = H1(min).
- Else:
    - H(min) = H2(min).

In extract min we create a function for deleting the minimum node and setting the min pointer to the minimum value in the remaining heap,

- Delete the min node.

- Set head to the next min node and add all the tree of the deleted node in root list.
- Create an array of degree pointers of the size of the deleted node.
- Set degree pointer to current node.
- Move to the next node.
    - If degrees are different then set degree pointer to next node.
    - If degrees are same then join the Fibonacci trees by union operation.
- Repeat steps 4 and 5 until the heap is completed.

To decrease the value of any element in the heap, we follow the following algorithm:

- Decrease the value of the node ‘x’ to the new chosen value.
- case 1:If min heap property is not violated,
- Update min pointer if necessary.
- case 2: If min heap property is violated and parent of ‘x’ is unmarked,
- Cut off the link between ‘x’ and its parent.
- Mark the parent of ‘x’.
- Add tree rooted at ‘x’ to the root list and update min pointer if necessary.
- case 3:If min heap property is violated and parent of ‘x’ is marked,
- Cut off the link between ‘x’ and its parent p[x].
- Add ‘x’ to the root list, updating min pointer if necessary.
- Cut off link between p[x] and p[p[x]].
- Add p[x] to the root list, updating min pointer if necessary.
- If p[p[x]] is unmarked, mark it.

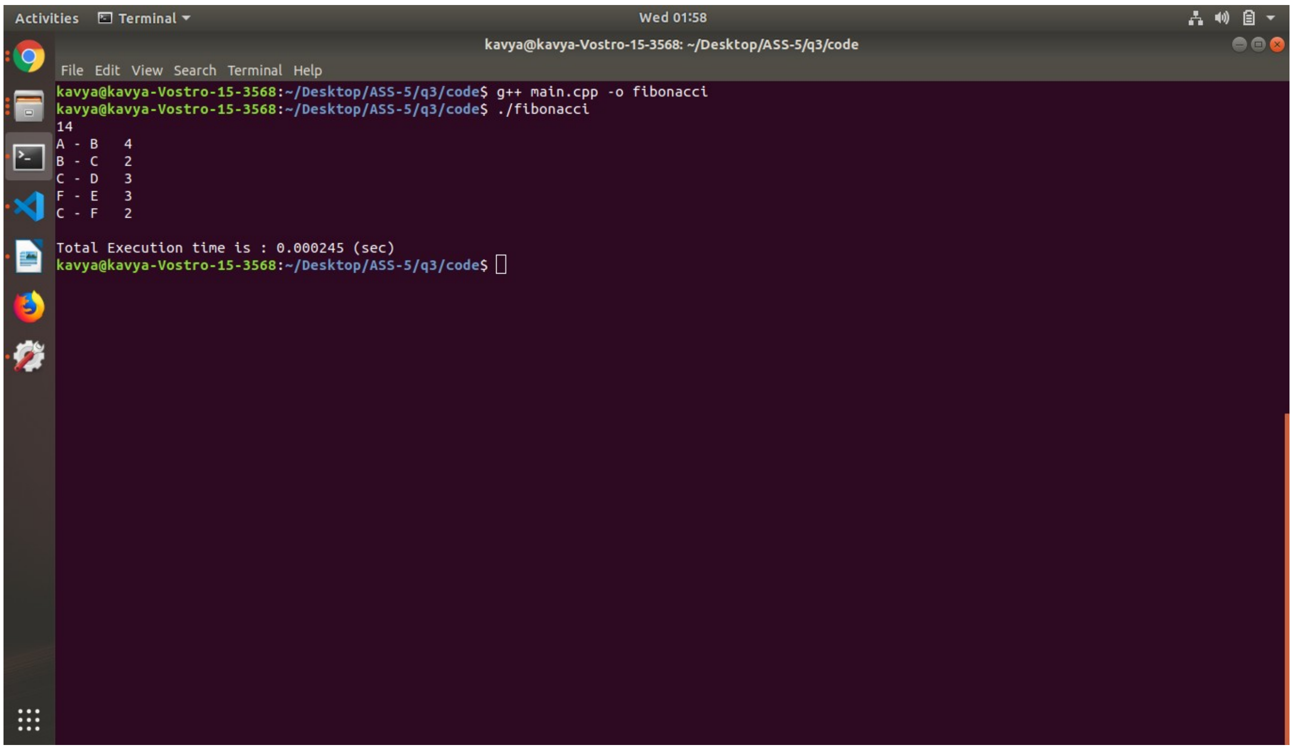
- Else, cut off p[p[x]] and repeat steps 4.2 to 4.5, taking p[p[x]] as ‘x’.

To delete any element in a Fibonacci heap,

- Decrease the value of the node to be deleted ‘x’ to minimum by Decrease_key() function.
- By using min heap property, heapify the heap containing ‘x’, bringing ‘x’ to the root list.
- Apply Extract_min() algorithm to the Fibonacci heap.

Prim’s algorithm a spanning tree means all vertices must be connected. So the two disjoint subsets 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,

- Create a set that keeps track of vertices already included in MST.
- 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.
- While the set doesn’t include all vertices
- Pick a vertex *u* which is not there in the set and has minimum key value.
- Include *u* to set.
- 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.


Activities
Terminal
Wed 01:58
kavya@kavya-Vostro-15-3568: ~/Desktop/ASS-5/q3/code
File Edit View Search Terminal Help

```
kavya@kavya-Vostro-15-3568:~/Desktop/ASS-5/q3/code$ g++ main.cpp -o fibonacci
kavya@kavya-Vostro-15-3568:~/Desktop/ASS-5/q3/code$ ./fibonacci
14
A - B   4
B - C   2
C - D   3
F - E   3
C - F   2

Total Execution time is : 0.000245 (sec)
kavya@kavya-Vostro-15-3568:~/Desktop/ASS-5/q3/code$ 
```

===