CSN-261:Data Structures Laboratory

Assignment-6

Name-Kavya Barnwal

Class- B.Tech CSE

Sub Batch- O2

Enrollment No- 18114039

Email Id- kbarnwal@cs.iitr.ac.in

Problem1.

Write a menu driven C++ program to implement a graph using adjacency list (linked list) without using STL. Perform following operations on the graph.

1. Insert edge
2. BFS traversal
3. DFS traversal
4. Cycle finding in the graph
5. Calculate diameter of the graph .

Algorithms and Data Structures used in the implementation

Data structure- Adjacency List(Linked list)

Algorithm-

BFS-

First move horizontally and visit all the nodes of the current layer

Move to the next layer

Consider the following diagram.

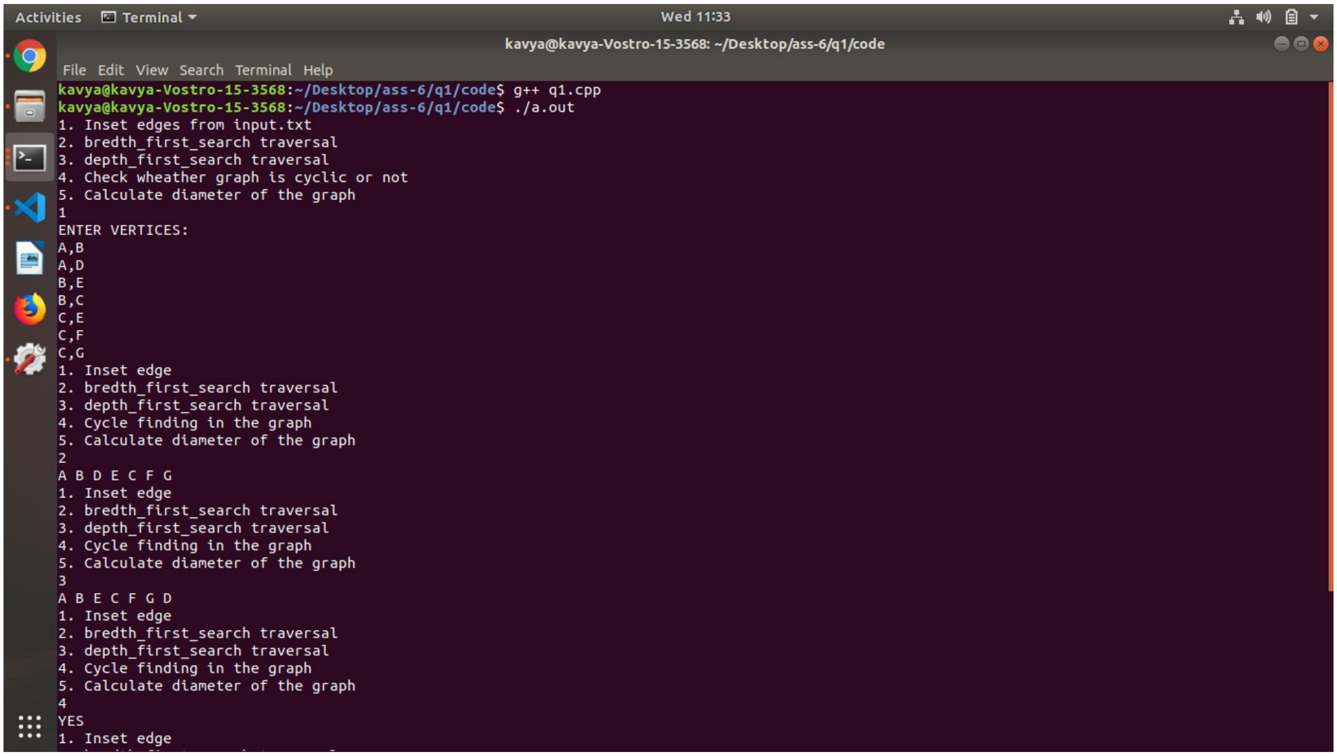
The distance between the nodes in layer 1 is comparitively lesser than the distance between the nodes in layer 2. Therefore, in BFS, you must traverse all the nodes in layer 1 before you move to the nodes in layer 2.

DFS-
Pick a starting node and push all its adjacent nodes into a stack.
Pop a node from stack to select the next node to visit and push all its adjacent nodes into a stack.
Repeat this process until the stack is empty. However, ensure that the nodes that are visited are marked. This will prevent you from visiting the same node more than once. If you do not mark the nodes that are visited and you visit the same node more than once, you may end up in an infinite loop.

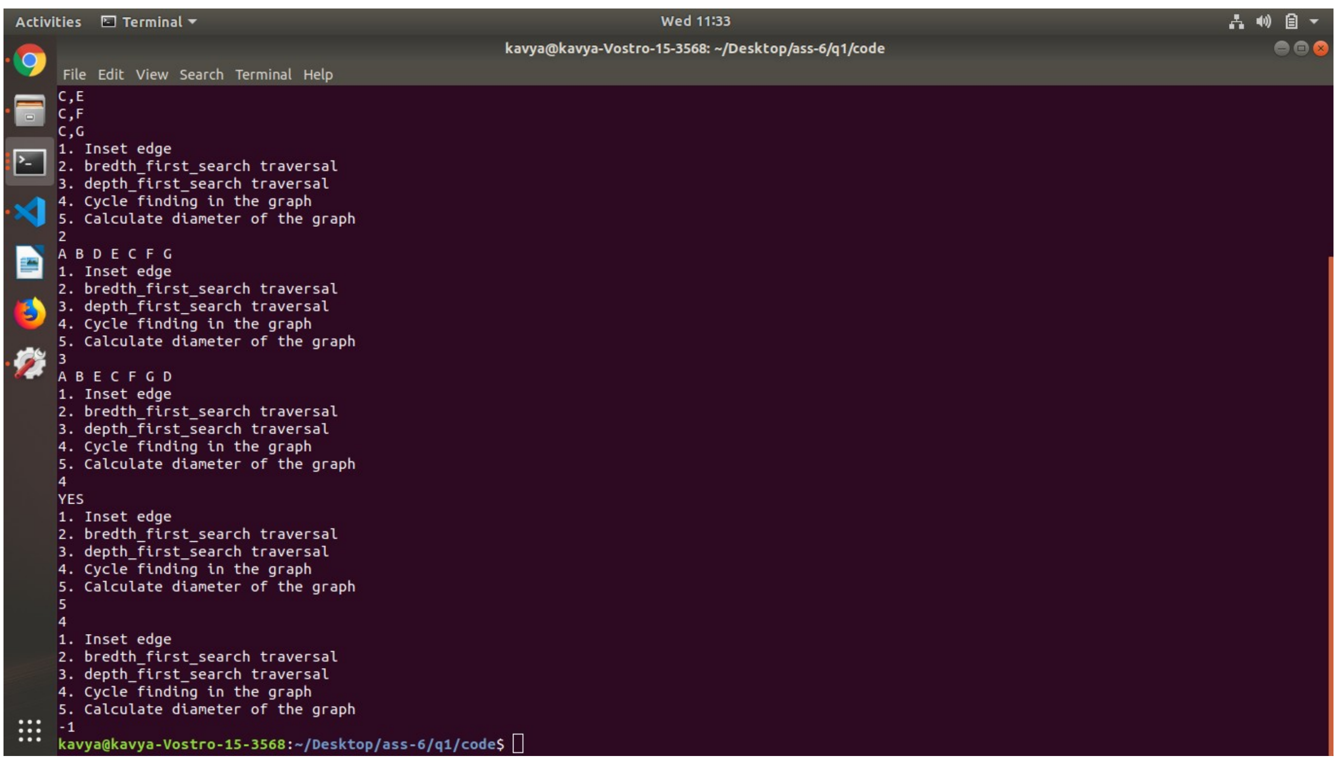
Cycle Detection-

To detect a back edge, we can keep track of vertices currently in recursion stack of function for DFS traversal. If we reach a vertex that is already in the recursion stack, then there is a cycle in the tree. The edge that connects current vertex to the vertex in the recursion stack is a back edge. We have used recStack[] array to keep track of vertices in the recursion stack.

1. Choose a vertex .
2. Among all the vertices that are as far from as possible, let be one with minimal degree.
3. If then set and repeat with step 2, else is a pseudo-peripheral vertex.




```
kavya@kavya-Vostro-15-3568:~/Desktop/ass-6/q1/code$ g++ q1.cpp
kavya@kavya-Vostro-15-3568:~/Desktop/ass-6/q1/code$ ./a.out
1. Inset edges from input.txt
2. bredth_first_search traversal
3. depth_first_search traversal
4. Check wheather graph is cyclic or not
5. Calculate diameter of the graph
1
ENTER VERTICES:
A,B
A,D
B,E
B,C
C,E
C,F
C,G
1. Inset edge
2. bredth_first_search traversal
3. depth_first_search traversal
4. Cycle finding in the graph
5. Calculate diameter of the graph
2
A B D E C F G
1. Inset edge
2. bredth_first_search traversal
3. depth_first_search traversal
4. Cycle finding in the graph
5. Calculate diameter of the graph
3
A B E C F G D
1. Inset edge
2. bredth_first_search traversal
3. depth_first_search traversal
4. Cycle finding in the graph
5. Calculate diameter of the graph
4
YES
1. Inset edge
```




```
C,E
C,F
C,G
1. Inset edge
2. bredth_first_search traversal
3. depth_first_search traversal
4. Cycle finding in the graph
5. Calculate diameter of the graph
2
A B D E C F G
1. Inset edge
2. bredth_first_search traversal
3. depth_first_search traversal
4. Cycle finding in the graph
5. Calculate diameter of the graph
3
A B E C F G D
1. Inset edge
2. bredth_first_search traversal
3. depth_first_search traversal
4. Cycle finding in the graph
5. Calculate diameter of the graph
4
YES
1. Inset edge
2. bredth_first_search traversal
3. depth_first_search traversal
4. Cycle finding in the graph
5. Calculate diameter of the graph
5
4
1. Inset edge
2. bredth_first_search traversal
3. depth_first_search traversal
4. Cycle finding in the graph
5. Calculate diameter of the graph
-1
kavya@kavya-Vostro-15-3568:~/Desktop/ass-6/q1/code$
```

Problem2 :

A binomial heap is implemented as a set of binomial trees, which are defined recursively as follows: x A binomial tree of order 0 is a single node x A binomial tree of order k has a root node whose children are roots of binomial trees of orders k−1, k−2, ..., 2, 1, 0 (in this order). x A binomial tree of order k has 2k nodes, height k. Write a C++ program to implement a binomial heap using heap data structures (without using STL). Print the order of each binomial heap and use Graphviz to show the forest of binomial heap.
Input 7 10, 20, 30, 40, 50, 60, 70
Output:
Order : Heap elements
0 : 70
1 : 50 60
2 : 10 30 40 20 B C E G F
3 : binomial heap will be formed of order 0 1 and 2

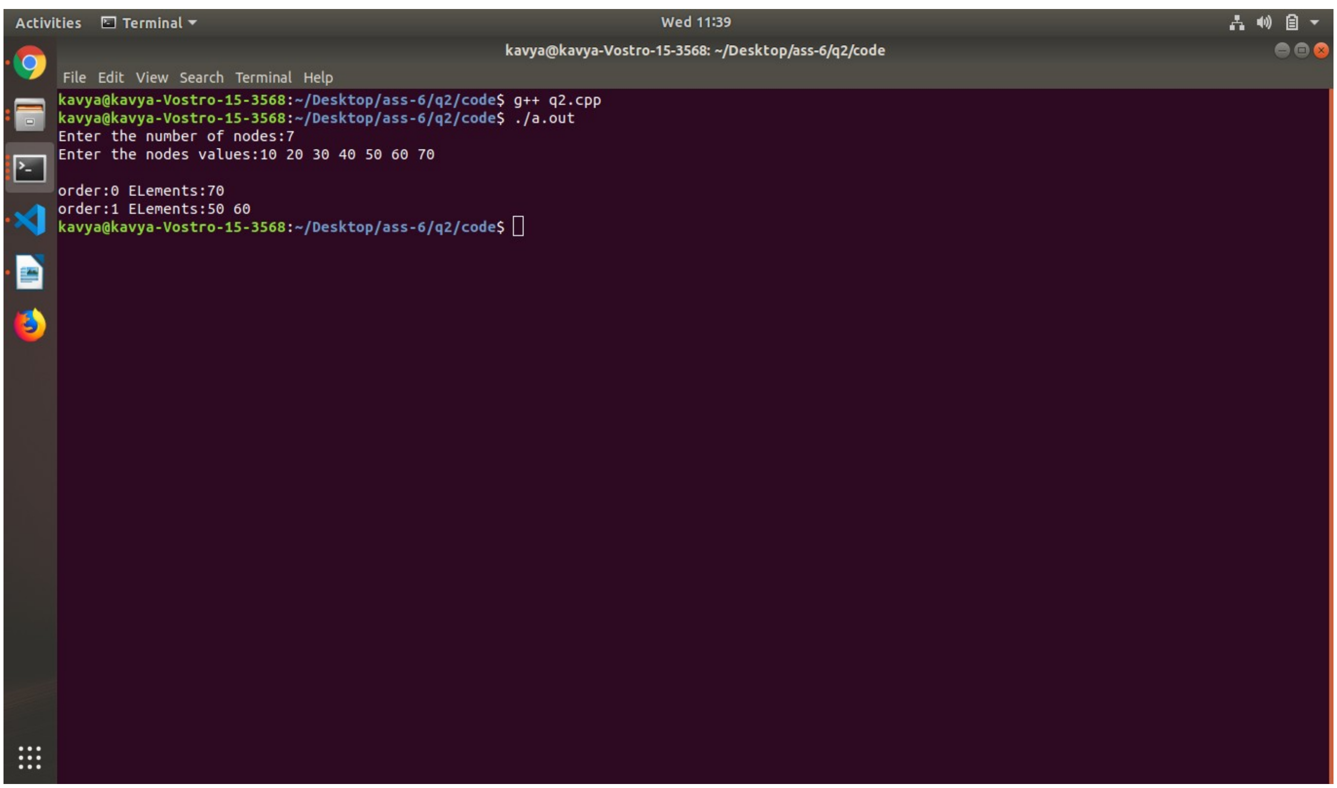
Algorithm and Data Structures used:

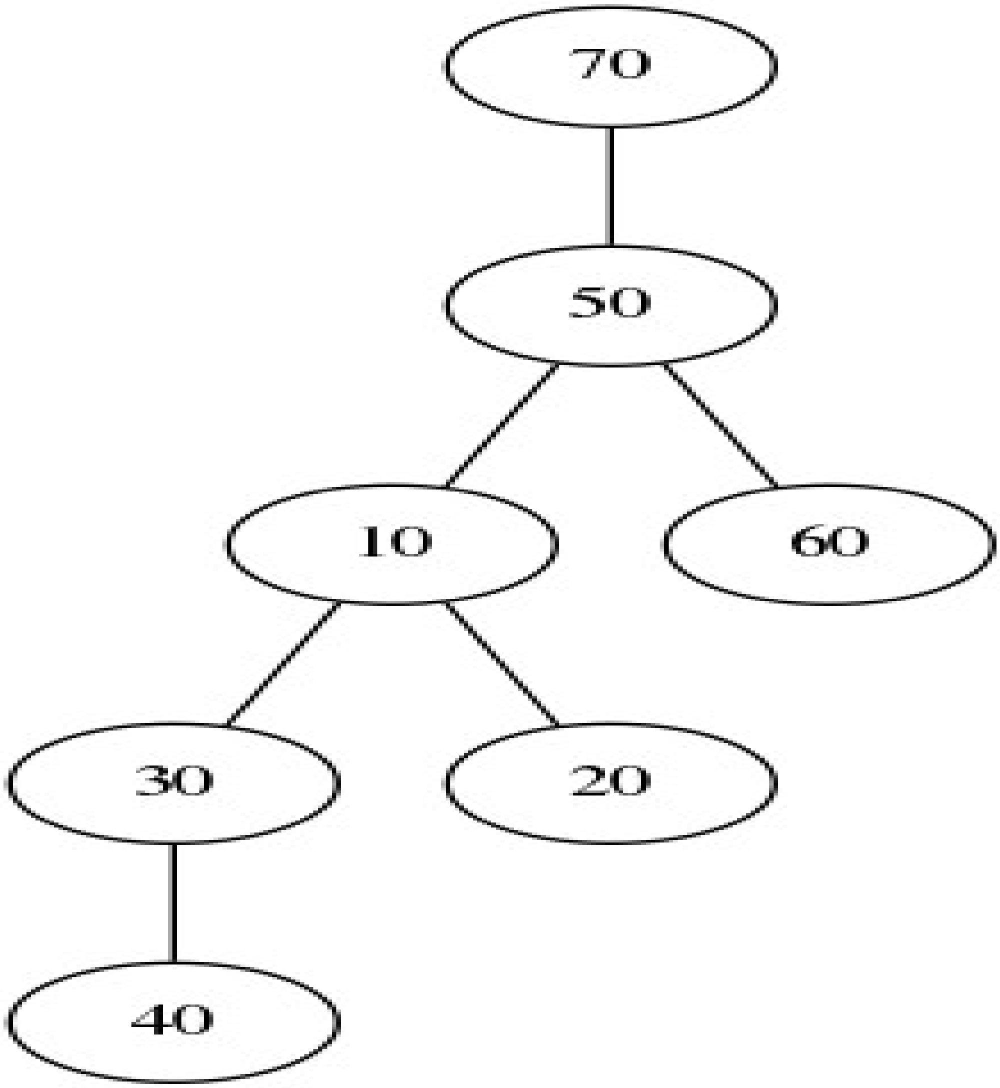
Data structure : Binomial Trees

Algorithm:

The following procedure inserts node *x* into binomial heap *H*, assuming of course that node *x* has already been allocated and *key*[*x*] has already been filled in.

```
BINOMIAL-HEAP-INSERT(H,x)
H' <= MAKE-BINOMIAL-HEAP()
p[x] <= NIL
child[x] <= NIL
sibling[x] <= NIL
degree[x] <= 0
head[H'] <= x
H <= BINOMIAL-HEAP-UNION(H,H')
```




70
50
10
60
30
20
40

Problem Statement 3:

Write a C++ program to implement Bentley-Ottmann Algorithm to find and print all the intersection points of n given lines. Use of STL is allowed. The specific type of data structure that must be used include Priority Queue and BST. Using least square method find the linear fit of the M found intersection points and print the line in the form ax+b. The student should demonstrate this on a GUI using QT library. The input should be given in following format: 1. Input number of line segments, N 2. N lines where 2N points are provided, i.e., 2 points in each line
Sample Input:
N = 6 P1X P1Y P2X P2Y
104 212 513 727
229 424 538 278
249 324 654 657
508 440 531 623
453 295 517 398
639 290 601 116
Sample Output:
No. of intersection points: 4 (260.53, 409.10) (318.94, 381.50) (464.13, 312.91) (521.59, 548.13) Linear fit: 0.2937x + 297.9693

Algorithm and Data Structures used:

Data Structure Used: BST & Priority Queue

Algorithm:

Initialize a priority queue *Q* of potential future events, each associated with a point in the plane and prioritized by the *x*-coordinate of the point. So, initially, *Q* contains an event for each of the endpoints of the input segments.

1. Initialize as self-balancing binary search tree *T* of the line segments that cross the sweep line *L*, ordered by the *y*-coordinates of the

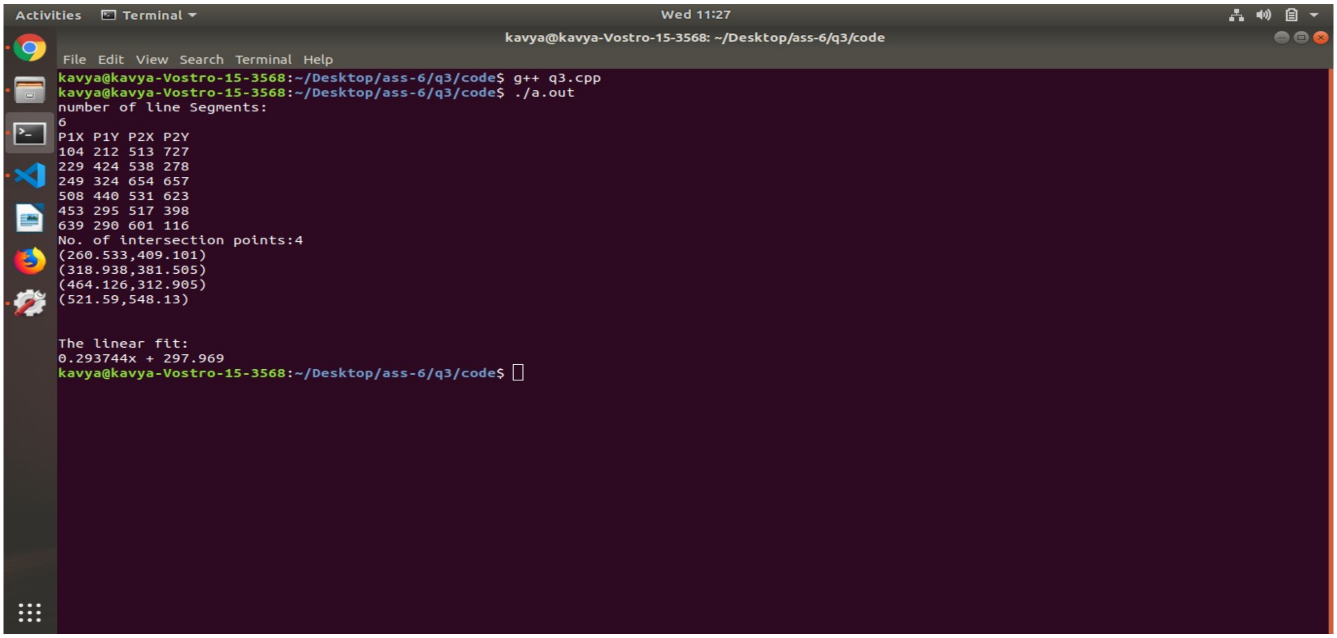
crossing points. Initially, T is empty. (Even though the line sweep L is not explicitly represented, it may be helpful to imagine it as a vertical line which, initially, is at the left of all input segments.)

2. While Q is nonempty, find and remove the event from Q associated with a point p with minimum x-coordinate. Determine what type of event this is and process it according to the following case analysis:

- If p is the left endpoint of a line segment s, insert s into T. Find the line-segments r and t that are respectively immediately above and below s in T (if they exist); if the crossing of r and t (the neighbours of s in the status data structure) forms a potential future event in the event queue, remove this possible future event from the event queue. If s crosses r or t, add those crossing points as potential future events in the event queue.

- If p is the right endpoint of a line segment s, remove s from T. Find the segments r and t that (prior to the removal of s) were respectively immediately above and below it in T (if they exist). If r and t cross, add that crossing point as a potential future event in the event queue.

- If p is the crossing point of two segments s and t (with s below t to the left of the crossing), swap the positions of s and t in T. After the swap, find the segments r and u (if they exist) that are immediately below and above t and s, respectively. Remove any crossing points rs (i.e. a crossing point between r and s) and tu(i.e. a crossing point between t and u) from the event queue, and, if r and t cross or s and u cross, add those crossing points to the event queue.


Activities
Terminal
Wed 11:27
kavya@kavya-Vostro-15-3568: ~/Desktop/ass-6/q3/code
File Edit View Search Terminal Help
kavya@kavya-Vostro-15-3568:~/Desktop/ass-6/q3/code$ g++ q3.cpp
kavya@kavya-Vostro-15-3568:~/Desktop/ass-6/q3/code$ ./a.out
number of line Segments:
6
P1X P1Y P2X P2Y
104 212 513 727
229 424 538 278
249 324 654 657
508 440 531 623
453 295 517 398
639 290 601 116
No. of intersection points:4
(260.533,409.101)
(318.938,381.505)
(464.126,312.905)
(521.59,548.13)
The linear fit:
0.293744x + 297.969
kavya@kavya-Vostro-15-3568:~/Desktop/ass-6/q3/code$