INDIAN INSTITUTE OF TECHNOLOGY ROORKEE

CSN-261 : Data Structures Laboratory

Assignment-6

Name : **Abhishek Patil**

Enr. No. : **18114003**

E-mail : **apatil@cs.iitr.ac.in**

Problem 1 :

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

Algorithms Used :

BFS is a traversing algorithm where you should start traversing from a selected node (source or starting node) and traverse the graph layerwise thus exploring the neighbour nodes (nodes which are directly connected to source node). You must then move towards the next-level neighbour nodes.

As the name BFS suggests, you are required to traverse the graph breadthwise as follows:

1.First move horizontally and visit all the nodes of the current layer
2.Move to the next layer

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.

Depth First Traversal can be used to detect a cycle in a Graph. DFS for a connected graph produces a tree. There is a cycle in a graph only if there is a back edge present in the graph. A back edge is an edge that is from a node to itself (self-loop) or one of its ancestor in the tree produced by DFS.For a disconnected graph, we get the DFS forest as output. To detect cycle, we can check for a cycle in individual trees by checking back edges.

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.

Data Structure Used :

1. linked list
2. graph

3. stack

4. queue

Snapshots :

```
Question1$ g++ question1.cpp
Question1$ ./a.out
1. insert edge
2. BFS transversal
3. DFS transversal
4. cycle finding in graph
5. Calculate diameter of graph
2
starting vertex : A
BFS is : A B C I F D U
```

```
1. insert edge
2. BFS transversal
3. DFS transversal
4. cycle finding in graph
5. Calculate diameter of graph
3
starting vertex : A
```

```
DFS is : A B I U D F C
1. insert edge
2. BFS transversal
3. DFS transversal
4. cycle finding in graph
5. Calculate diameter of graph
4
Yes
```

```
1. insert edge
2. BFS transversal
3. DFS transversal
4. cycle finding in graph
5. Calculate diameter of graph
5

diameter is : 3
```

Problem 2 :

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

Algorithms Used :

Binomial Heap is an extension of Binary Heap that provides faster union or merge operation together with other operations provided by Binary Heap.

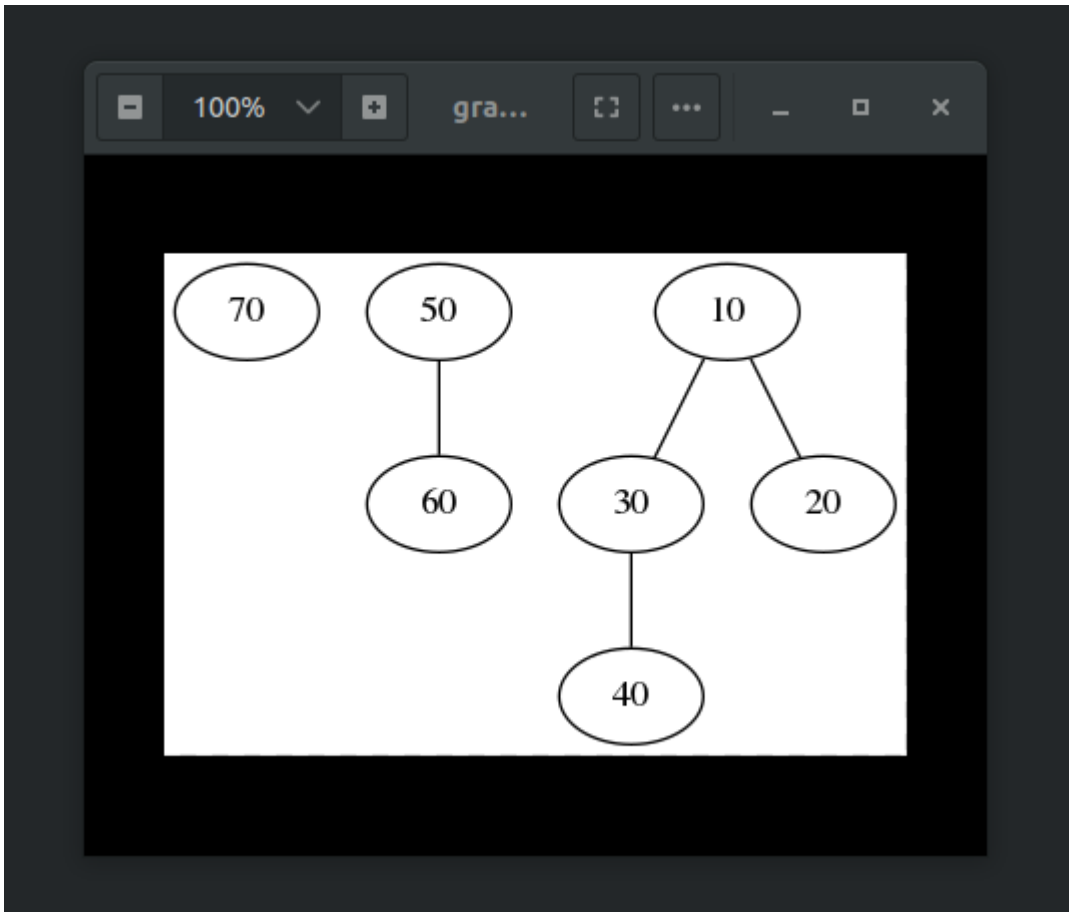
A Binomial Tree of order 0 has 1 node. A Binomial Tree of order k can be constructed by taking two binomial trees of order k-1 and making one as leftmost child or other.

Data Structure Used :

1. min Heap
2. binomial heap

Snapshots :

```
Question2$ ./a.out
Order : Heap elements
    0 : 70
    1 : 50 60
    2 : 10 30 40 20
```



Problem 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

Algorithms Used :

The Bentley–Ottmann algorithm performs the following steps.

1.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.

2.Initialize a 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 T 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.)

3.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.

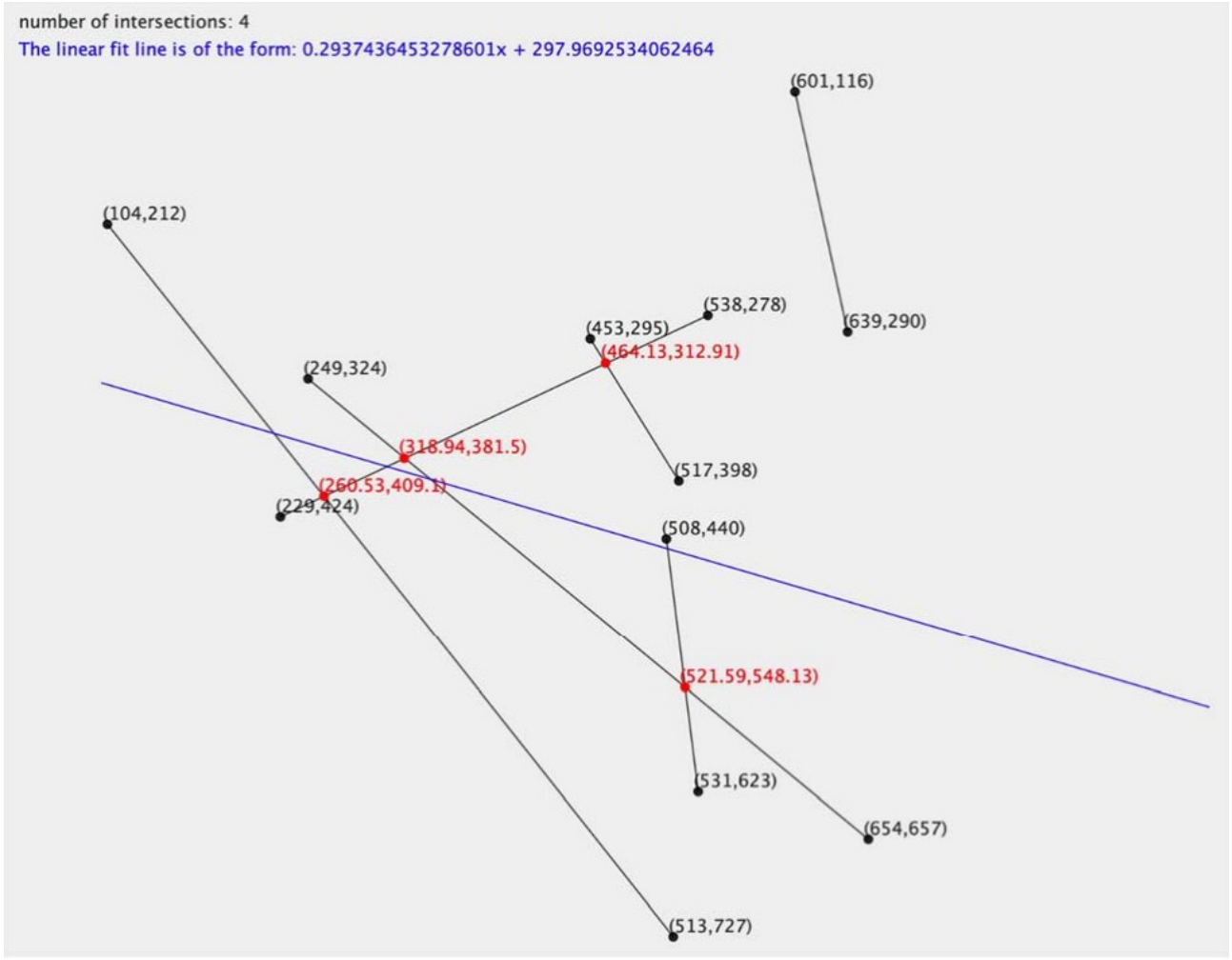
•If *p* is the right endpoint of a line segment *s*, remove *s* from *T*. Find the segments *r* and *t* that (prior to the removal of *s*) were respectively immediately above and below it in *T* (if they exist). If *r* and *t* cross, add that crossing point as a potential future event in the event queue.

•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.

Data Structure Used :

1. priority queue
2. BST

Snapshots :


number of intersections: 4
The linear fit line is of the form: 0.2937436453278601x + 297.9692534062464
(601,116)
(104,212)
(538,278)
(639,290)
(453,295)
(464.13,312.91)
(249,324)
(318.94,381.5)
(517,398)
(260.53,409.1)
(229,424)
(508,440)
(521.59,548.13)
(531,623)
(654,657)
(513,727)