TEL AVIV UNIVERSITY
Department of Computer Science
0368.4281 – Advanced topics in data structures
Spring Semester, 2000/2001

Final project, june 6 2001

Due on August 15 2001.

Choose **one** of the following programming projects. You can implement the algorithm in a language of your choice as long as we can easily compile and run it. Be sure that the code is standard and can be compiled easily by any compiler. Hand in the following items.
1) The source file/files of your code documented properly. Don't compromise the readability of your code.
2) Examples of inputs on which you tested your implementation and the corresponding outputs.
2) A written document that describes in detail the followings:
a) how to compile and run your program.
b) The details of your implementation and all design choices you have made. For example you should say which version of data structure have you implemented ? How do you represent it ? etc.
c) The high level structure of the code. I.e. which module contains which part of the implementation.

1. Implement Fibonacci heaps. Use your implementation of Fibonnacci heaps to implement Prim's algorithm that finds a minimum spanning tree in $O(m + n \log n)$ time. The input to your program should consists of a line containing the number of vertices in the input graph, say n, followed by a list of edges. We assume that each vertex is identified by a unique nonnegative integer $i < n$. Each edge is described by a triple i, j, w of numbers in a single line. The first two numbers i and j are integers that identify the vertices which are the endpoints of the edge. The last number is the weight of the edge. For example

5
0 1 5
0 2 6
0 3 17
0 4 21
0 5 1

is a legitimate input that describes a star graph on five vertices. Your output should consists of the list of edges in the minimum spanning tree. You may assume that the graph is connected and simple (no loops or parallel edges).

2. Implement dynamic trees, supporting the set of operations as required by the algorithm for computing a blocking flow that we described in class. Use your implementation to implement an algorithm that finds a blocking flow from s to t in a directed graph and runs in $O(m \log n)$ time.

The input to your program is as follows. The first line contains the number of nodes in the graph, s, and t. We assume that if n is the number of vertices in the graph then each vertex is identified by an integer $i < n$. Each of the following lines describes a directed edge and its capacity. For example the input

5 0 4
0 1 1
1 2 2
2 3 6
3 4 7

describes a graph with 5 vertices in which we want to find a blocking flow from vertex 0 to vertex 4. The graph is a directed path $0- > 1- > 2- > 3- > 4$ where the capacity of the arc $0- > 1$ is 1, the capacity of the arc $1- > 2$ is 2 etc. Your output should consists of the list of the arcs in the graph together with the blocking flow on each arc.

3. Implement homogenous 2-4 trees and use your implementation to implement the algorithm described in class to identify and sort jordan sequences. Your input should consist of a sequence of numbers. If there is no jordan curve (nowhere tanget to the x axis) such that the given sequence is the sequence of its intersection points with the x-axis your output should provide a proof that this is indeed the case (show the point of conflict). If there is such a curve you should print the sorted sequence.

4. Implement partially persistent balanced search trees that support insert and delete in $O(\log n)$ time and $O(1)$ space (amortized). Use your implementation to construct a data structure for the following version of the planar point location problem.

In our version of the planar point location problem the input is a sequence of line seqments that may intersect only at their endpoints. You should preprocess the sequence and construct a data structure such that given a point in the plane you can easily report the first segment that a vertical ray from the point down would meet (or an indication that there is no such segment). After processing the input your program should turn into an interactive mode and be ready to answer queries. The line segments are described by pairs of points on the plane such that each pair spans one line. For example the line

1 4 2 6

describes the segment from (1,4) to (2,6). Be sure that your data structure takes $O(n)$ space, your preprocessing time is $O(n \log n)$ and that you answer queries in $O(\log n)$ time, where n is the number of segments in the input.