Homework #4: Priority Queues, Disjoint Sets
Due Date: Tuesday, 25 March 2003

Guidelines

Please make sure you adhere to the policies on collaboration and academic honesty as outlined in Handout #1.

Reading

Read Ch. 19, 20, 21 of CLRS.

Problems

Problem A (25 points) "You may discuss ideas with other students."

We have already seen several algorithms for finding a minimum spanning tree of an undirected graph. The following pseudocode outlines yet another such algorithm. It maintains a partition $\{V_i\}$ of the vertices of V and, with each set V_i, a set

$$E_i \subseteq \{(u,v) : u \in V_i \text{ or } v \in V_i\}$$

of edges incident on vertices in V_i.

```
MST(G)
 1  T ← ∅
 2  for each vertex v_i ∈ V[G]
 3       do V_i ← {v_i}
 4          E_i ← {(v_i, v) ∈ E[G]}
 5  while there is more than one set V_i
 6       do choose any set V_i
 7          extract the minimum-weight edge (u, v) from E_i
 8          assume without loss of generality that u ∈ V_i and v ∈ V_j
 9          if i ≠ j
10          then T ← T ∪ {(u, v)}
11               V_i ← V_i ∪ V_j, destroying V_j
12               E_i ← E_i ∪ E_j
```

The correctness of the algorithm can be proven using techniques from Section 23.1. (*you do NOT have to prove the correctness yourself*).

The task for you is to show how the mergable-heap and disjoint-set data structures can be used to implement this algorithm. The first pages of Chapter 19 and Chapter 21 define, respectively, a set of mergable-heap and disjoint-set operations. Those data structures are implemented in the remainder of those chapters and you can assume their correctness and stated efficiency.

i. Describe how the mergable-heap and disjoint-set data structures can be used to maintain the various sets in this MST algorithm. You may use additional elementary data structures as needed.

   Be very specific about which lines in the above pseudocode correspond to which specific mergable heap or disjoint-set operations, as well as the parameters used for each such operation.

ii. Give the overall running time of the above MST algorithm, assuming that the mergable heaps are implemented as binomial heaps, and the disjoint sets use path-compression and union-by-rank.

Problem B (10 points) "Work entirely on your own."

Do Exercise 19.2-2 on page 472 of the text.

Note: During the execution of `Binomial-Heap-Union`*, it may be that the working list contains three consecutive trees of equal degree. The final outcome depends on which two of the three trees you link in this situation. We will gladly accept any such outcome; you need not worry about making sure that you choose the identical two trees as the specific code in the text.*

Problem C (15 points) "Work entirely on your own."

Do Exercise 19.2-3 on page 472 of the text.

Note: Similar comments as with Problem B

Problem D (10 points) "Work entirely on your own."

Do Exercise 20.2-1 on page 488 of the text.

Note: There is again some non-determinism throughout the operations, which depends on the exact order of the circular lists. We will accept any valid outcome, so long as it could result from the correct execution of the operation in question.

Problem E (15 points) "You may discuss ideas with other students."

Show that for any positive integer n, there exists a sequence of Fibonacci-heap operations that creates a Fibonacci heap consisting of just one tree that is a linear chain of n nodes.

Note: Make sure that you prove this fact (inductively) for all values of n.

Problem F (25 points) "You may discuss ideas with other students."

Recall that Prim's MST algorithm, when run using Fibonacci heaps as the underlying priority queue implementation, runs in worst-case time of $O(E + V \lg V)$.

Suppose edge weights in a graph are integers in the range from 1 to W. Develop a new priority queue implementation for this situation which results in a worst-case bound of $O(E + WV)$ time if used as the underlying priority queue for Prim's algorithm.

(*Hint: To get going, think about what you could do if $W = 1$. What if $W = 2$? What if $W = 3$?*)

Problem G **(EXTRA CREDIT – 10 points)**

"You may discuss ideas with other students."

Show that any sequence of m `Make-Set`, `Find-Set`, and `Union` operation, where all the `Union` operations appear before any of the `Find-Set` operations, takes only $O(m)$ time if both path compression and union by rank are used. What happens in the same situation if only the path-compression heuristic is used?