CMPS 2200 Introduction to Algorithms – Fall 12

10/29/12

7. Homework

Due **11/6/12** in the lab

1. **Faster MST (8 points)**
   Let $G = (V, E)$ be a connected undirected graph with edge weights $w : E \to \mathbb{R}$.

   (a) (4 points) If all of the edge weights are integers between 1 and $|V|$, show how Kruskal's algorithm can be adapted to run faster. What is the resulting runtime? (*Hint: What is the runtime-bottleneck of Kruskal's algorithm?*)

   (b) (4 points) Assume all edge weights are integers between 1 and 10. Show that Prim's algorithm can be implemented to work in $O(|V| + |E|)$ time in this case. *(Hint: Suggest a data structure based on bucketing vertices with the same weight, that replaces the priority queue and analyze the runtime.)*

2. **Queue from Stacks (8 points)**

   Assume we are given an implementation of a stack, in which PUSH and POP operations take constant time each. We now implement a FIFO queue using two stacks A and B as follows:

   ENQUEUE(x):
   • Push x onto stack A

   DEQUEUE():
   • If stack B is nonempty, return B.POP()
   • Otherwise, pop all elements from A and immediately push them onto B. Return B.POP()

   a) (2 points) Show how the following sequence of operations operates on the two stacks. Suppose the stacks are initially empty.
   Enqueue(1), Enqueue(2), Enqueue(3), Enqueue(4), Dequeue(), Enqueue(5), Enqueue(6), Dequeue()

   b) (2 points) Why do these implementations of ENQUEUE and DEQUEUE correctly implement FIFO queue behavior? *(Hint: It might help to argue which invariants hold for A and B.)*

   c) (1 point) What is the worst-case runtime of a single ENQUEUE operation? What is the worst-case runtime of a single DEQUEUE operation?

   d) (3 points) Show that the amortized runtime of ENQUEUE and DEQUEUE each is $O(1)$. *(Hint: Use the accounting method to prepay for future operations.)*

FLIP OVER TO BACK PAGE $\Longrightarrow$

3. **Union-Find (6 points)**

```
for(i=1; i<=16; i++) x[i]=MAKE-SET(i);
for(i=1; i<=15; i+=2) UNION(x[i],x[i+1]);
for(i=1; i<=13; i+=4) UNION(x[i],x[i+2]);
UNION(x[12],x[13]); UNION(x[1],x[8]);
FIND-SET(x[16]);
```

Assume an implementation of the Union-Find data structure with a disjoint-set forest with union-by-weight and path compression.

Show the data structure after every line of code. What is the answer to the `FIND-SET` operation?