CS 5633 Analysis of Algorithms – Spring 08

3/13/08

8. Homework

Due **Thursday 3/27/08** before class

1. **Knapsack DP (6 points)**
   Design a dynamic programming algorithm for the 0-1-Knapsack problem.

   - **(2 points)** Let $D[i, w]$ be the value of a solution considering items $1..i$ only and with maximum weight w. Come up with a recurrence relation for $D[i, w]$. *(Hint: it has two non-trivial cases, and it should use the maximum at some point.)*
   - **(2 points)** Use dynamic programming to compute $D[n, W]$. What is the runtime in terms of n and W?
   - **(2 points)** Extract the optimum set of items from the dynamic programming table. What is the runtime in terms of n and W?

2. **Greedy scheduling (6 points)**

   Suppose you are the manager of a train station. You are given a sorted set $A = a_1, \ldots, a_n$ of n points in time which are the times when trains will arrive at your station. Only when a train arrives you need to have employees at the station. Due to union rules, each employee can work at most one hour at the station. The problem is to find a scheduling of employees (i.e., a set of one-hour intervals) that covers all the times in A and uses the fewest number of employees.

   Prove or disprove that the two greedy approaches below correctly solve this problem. (Only one is correct, and the correctness can be proved using a copy-paste argument. In order to disprove give a counterexample.)

   (a) Let I be an interval that covers the most number of points in A. Add I to the solution and recursively continue on the points in A not covered by I.

   (b) Let a_j be the smallest point in A. Add the interval $I = (a_j, a_j + 1)$ to the solution, and recursively continue on the points in A not covered by I.

3. **Binary Counter (3 points)**
   Use aggregate analysis to show that, over a sequence of n increment operations on a binary counter, the amortized runtime of one such increment operation is $O(1)$. *(Hint: Study the flipping behavior of every single bit $A[i]$.)*

FLIP OVER TO BACK PAGE $\Longrightarrow$

4. **Queue from Two Stacks – (7 points)**

Assume we are given an implementation of a stack, in which PUSH and POP operations take constant time each. We now implement a 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 while doing so 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(9), Enqueue(8), Enqueue(7), Enqueue(6), Dequeue(), Dequeue(), Enqueue(5), Enqueue(6), Dequeue()

b) (2 points) Why is the algorithm correct? Argue which invariants hold for A and B.

c) (3 points) Prove using the accounting method that the amortized runtime of ENQUEUE and DEQUEUE each is $O(1)$. Argue why your account balance is always non-negative.