CS 362, HW 8

Prof. Jared Saia, University of New Mexico

1. Consider the following alternative greedy algorithms for the activity selection problem discussed in class. For each algorithm, either prove or disprove that it constructs an optimal schedule.
   (a) Choose an activity with shortest duration, discard all conflicting activities and recurse
   (b) Choose an activity that starts first, discard all conflicting activities and recurse
   (c) Choose an activity that ends latest, discard all conflicting activities and recurse
   (d) Choose an activity that conflicts with the fewest other activities, discard all conflicting activities and recurse
2. Now consider a weighted version of the activity selection problem. Imagine that each activity, a_i has a *weight*, $w(a_i)$, and weights are totally unrelated to activity duration. Your goal is now to choose a set of non-conflicting activities that give you the largest possible sum of weights, given an array of start times, end times, and values as input.
   (a) Prove that the greedy algorithm described in class - Choose the activity that ends first and recurse - does not always return an optimal schedule for this problem
   (b) Describe an algorithm to compute the optimal schedule in $O(n^2)$ time. Hint: 1) Sort the activities by finish times. 2) Let $m(j)$ be the maximum weight achievable from activities $a_1, a_2, \ldots, a_j$. 3) Come up with a recursive formulation for $m(j)$ and use dynamic programming. Hint 2: In the recursion in step 3, it'll help if you precompute for each job j, the value x_j which is the largest index i less than j such that job i is compatible with job j. Then when computing $m(j)$, consider that the optimal schedule could either include job j or not include job j.

3. Consider the following problem.
   INPUT: Positive integers $r_1, \ldots, r_n$ and $c_1, \ldots, c_n$.
   OUTPUT: An n by n matrix A with 0/1 entries such that for all i the sum of the ith row in A is r_i and the sum of the ith column in A is c_i, if such a matrix exists.
   Think of the problem this way. You want to put pawns on an n by n chessboard so that the ith row has r_i pawns and the ith column has c_i pawns. Consider the following greedy algorithm that constructs A row by row. Assume that the first $i-1$ rows have been constructed. Let a_j be the number of 1s in the jth column in the first $i-1$ rows. Now the r_i columns with maximum $c_j - a_j$ are assigned 1s in row i, and the rest of the columns are assigned 0's. That is, the columns that still needs the most 1's are given 1's. Formally prove that this algorithm is correct using an exchange argument.

4. Walt is making a device for his friend Hector that counts how many times Hector rings a bell. The software for the device requires a binary counter data structure with INCREMENT and RESET operators.

   In class we discussed an INCREMENT algorithm for incrementing a binary counter in $O(1)$ amortized time. Now we want to include a RESET algorithm that sets all the bits in the counter to 0. Below are the algorithms for INCREMENT and RESET. They use an array B of bits and an integer m giving the largest index in B set to 1.

Algorithm 1 INCREMENT(B, m)

```
1: i ← 0
2: while B[i] = 1 do
3:     B[i] ← 0
4:     i ← i + 1
5: end while
6: B[i] ← 1
7: if i > m then
8:     m ← i
9: end if
```

Algorithm 2 RESET(B, m)

```
1: for i ← 0 to m do
2:     B[i] ← 0
3: end for
```

   Let n be the number of operations on this binary counter. Give the following costs as a function of n.

   (a) What is the worst-case run time of INCREMENT?

(b) What is the worst-case run time of RESET?

(c) Prove that in an arbitrary sequence of calls to INCREMENT and RESET, each call has amortized cost O(1). Hint: Use the accounting method and save up dollars during INCREMENT for future calls to RESET.