

## CS 561, HW7

Prof. Jared Saia, University of New Mexico

1. The count-min sketch discussed in class only provides count estimates of the number of times each item has been seen. What if we want to augment the sketch to also keep track of the  $x$  items with largest count estimates for some value  $x$ ? Describe how you would do this. You may find useful a heap; recall that a heap provides the functionality to (1) *insert* an item with a given value; (2) *increase the key* of an item already in the heap; and (3) *delete min*: delete the item in the heap with minimum key. Assume these heap operations all take  $O(\log x)$  time. What is the run time of your augmented count-min sketch?
2. 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
3. Now consider a weighted version of the activity selection problem. Imagine that each activity,  $a_i$  has a *weight*,  $w(a_i)$  (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  $a_1, a_2, \dots, a_j$ . 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  $j$ .
4. Consider the following problem.

INPUT: Positive integers  $r_1, \dots, r_n$  and  $c_1, \dots, c_n$ .

OUTPUT: An  $n$  by  $n$  matrix  $A$  with 0/1 entries such that for all  $i$  the sum of the  $i$ th row in  $A$  is  $r_i$  and the sum of the  $i$ th 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  $i$ th row has  $r_i$  pawns and the  $i$ th 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 1's in the  $j$ th column in the first  $i - 1$  rows. Now the  $r_i$  columns with maximum  $c_j - a_j$  are assigned 1's in row  $i$ , and the rest of the columns are assigned 0's. That is, the columns that still need the most 1's are given 1's. Formally prove that this algorithm is correct using an exchange argument.