CSE 3101 Design and Analysis of Algorithms
Practice Test for Unit 4
Greedy Algorithms

Jeff Edmonds

First learn the steps. Then try them on your own. If you get stuck only look at a little of the answer and then try to continue on your own.

1. Integer-Knapsack Problem:

   Instances: An instance consists of $\langle V, \langle v_1, p_1\rangle, \ldots, \langle v_n, p_n\rangle\rangle$. Here, V is the total volume of the knapsack. There are n objects in a store. The volume of the i^{th} object is v_i, and its price is p_i.

   Solutions: A solution is a subset $S \subseteq [1..n]$ of the objects that fit into the knapsack, i.e., $\sum_{i\in S} v_i \leq V$.

   Measure Of Success: The cost (or success) of a solution S is the total value of what is put in the knapsack, i.e., $\sum_{i\in S} p_i$.

   Goal: Given a set of objects and the size of the knapsack, the goal is fill the knapsack with the greatest possible total price.

   Failed Greedy Algorithms: Three obvious greedy algorithms take first the most valuable object, $\max_i p_i$, the smallest object, $\min_i v_i$, or the object with the highest value per volume, $\max_i \frac{p_i}{v_i}$. However, they do not work. For each of this greedy algorithms, give an instance on which it does not work.

2. We proved that the greedy algorithm does not work for the making change problem when the denominations of the coins are 4, 3, and 1 cent, but it does work when the denominations are 25, 10, 5, and 1. Does it work when the denominations are 25, 10, and 1, with no nickels?

3. Greedy Algorithms: A man is standing on the bank of a river that he must cross by jumping on stepping stones which are spread in a line across the river. Though he can't jump far, he wants to jump on as few stones as possible. An *instance* to the problem specifies $\langle d_0, d_1, d_2, \ldots, d_n\rangle$, where $d_0 = 0$ is the man's initial location, d_i for $i \in [1, .., n-1]$ is a real number giving the distance in meters that the i^{th} stone is from him, and d_n is the distance to the opposite shore. Assume that the stones are in order, i.e. $0 = d_0 \leq d_1 \leq d_2 \leq \ldots \leq d_n$. Also assume that the stones are not more than one meter apart, i.e. $\forall i\ d_{i+1} - d_i \leq 1$.

   A *solution* is a sequence $\langle i_0, i_1, i_2, \ldots i_\ell\rangle$ of the indexes of the stones indicating that at time t the man jumps onto the ${i_t}^{th}$ stone. Such a solution is *valid* if at time $t = 0$ the man is in fact on the close shore, i.e. $i_0 = 0$, he finishes on the far shore, i.e. $i_\ell = n$, and at each time step t' the distance he jumps is at most a distance of one meter, i.e. $d_{i_{t'}} - d_{i_{t'-1}} \leq 1$. The *cost* of a solution is the number ℓ of stones jumped onto.

   Note that the assumption on the input ensures that a valid solution always exists.
   Specify a greedy algorithm for finding an optimal solution for this problem. Imagine that you are crossing a river on stones. How would you choose which stone to jump on next.

   As done in the steps, prove that your algorithm always returns an optimal solution.
   Warning: To ensure that a solution is valid, you really must check the distance jumped at *each* time step t'.

4. Job/event scheduling problem with multiple rooms:
   Review the job/event scheduling problem from Section 16.2.1. This problem is the same except you have r rooms/processors within which to schedule that the set of jobs/events. An instance is $\langle r, \langle s_1, f_1\rangle, \langle s_2, f_2\rangle, \ldots, \langle s_n, f_n\rangle\rangle$, where as before $0 \leq s_i \leq f_i$ are the starting and finishing times for the i^{th} event. But now the input also specifies the number of rooms r. A solution for an instance is a schedule $S = \langle S_1, \ldots, S_r\rangle$ for each of the rooms. Each of these consists of a subset $S_j \subseteq [1..n]$ of the events that don't conflict by overlapping in time. The success of a solution S is the number of events scheduled, that is, $|\cup_{j\in[r]} S|$. Consider the following four algorithms:

(a) Find the greedy solution for the first room. Then find the greedy solution for the second from the remaining events. Then the third room. And so on.

(b) It starts by sorting the events by their finishing times, just like in the one-room case. Then, it looks at each event in turn, scheduling it, if possible. If it can be scheduled in more than one room, we assign it in the first room at which it fits. I.e./ first try room one, then room two, and so on until it fits. If it cannot be scheduled in any room, then it is not scheduled.

(c) Same except, the next event is scheduled in the room with the latest last-scheduled finishing time. For example, suppose the last event scheduled in rooms 1, 2, and 3 finishes at times 10, 15, and 18, and the next event starts at time 17. Then the next event could be scheduled into either room 1 or 2 but not in 3. This algorithm would schedule it in room 2. Note this is the room that minimizes the gap between starting time of the new job and the finishing time of the previously scheduled job for the room. Here a gap of size $17 - 15 = 2$ is better than one of size $17 - 10 = 7$ or of size $17 - 18 = -1$.

(d) Same except, the next event is scheduled in the room with the earliest last-scheduled finishing time. Note this is the room that maximizes the said mentioned gap.

Prove that three of these algorithms do not lead to an optimal schedule and that the remaining one does.