CSE 441T/541T: Advanced Algorithms **Fall Semester, 2003**

Practice Problems Solutions

September 9, 2004

Here are the solutions for the practice problems. **However**, reading these is far less useful than solving these yourself, writing up your solution and then either comparing your solution to these or showing your solution to one of us at our office hours to look over.

1. The greedy algorithm we use is to place the first interval at $[x_1, x_1 + 1]$, remove all points in $[x_1, x_1 + 1]$ and then repeat this process on the remaining points.

   Clearly the above is an $O(n)$ algorithm. We now prove it is correct.

   Greedy Choice Property: Let S be an optimal solution. Suppose S places its leftmost interval at $[x, x + 1]$. By definition of our greedy choice $x \leq x_1$ since it puts the first point as far right as possible while still covering x_1. Let S' be the scheduled obtained by starting with S and replacing $[x, x + 1]$ by $[x_1, x_1 + 1]$. We now argue that all points contained in $[x, x + 1]$ are covered by $[x_1, x_1 + 1]$. The region covered by $[x, x + 1]$ which is not covered by $[x_1, x_1 + 1]$ is $[x, x_1)$ which is the points from x up until x_1 (but not including x_1). However, since x_1 is the leftmost point there are no points in this region. (There could be additional points covered by $[x + 1, x_1 + 1]$ that are not covered in $[x, x + 1]$ but that does not affect the validity of S'). Hence S' is a valid solution with the same number of points as S and hence S' is an optimal solution.

   Optimal Substructure Property: Let P be the original problem with an optimal solution S. After including the interval $[x_1, x_1 + 1]$, the subproblem P' is to find an solution for covering the points to the right of $x_1 + 1$. Let S' be an optimal solution to P'. Since, $\text{cost}(S) = \text{cost}(S') + 1$, clearly an optimal solution to P includes within it an optimal solution to P'.

2. The greedy algorithm we use is to go as far as possible before stopping for gas. Let c_i be the city with distance d_i from St. Louis. Here is the pseudo-code.

   $S = \emptyset$
   $last = 0$
   for $i = 1$ to n
       if $(d_i - last) > m$
           $S = S \cup \{c_{i-1}\}$
           $last = t_{i-1}$

   Clearly the above is an $O(n)$ algorithm. We now prove it is correct.

   Greedy Choice Property: Let S be an optimal solution. Suppose that its sequence of stops is $s_1, s_2, \ldots, s_k$ where s_i is the stop corresponding to distance t_i. Suppose that g is the first stop made by the above greedy algorithm. We now show that there is an optimal solution with a first stop at g. If $s_1 = g$ then S is such a solution. Now suppose that $s_1 \neq g$. Since the greedy algorithm stops at the latest possible city then it follows that s_1 is before g. We now argue that $S' = \langle g, s_2, s_3, \ldots, s_k \rangle$ is an optimal solution. First note that $|S'| = |S|$. Second, we argue that S' is legal (i.e. you never

run out of gas). By definition of the greedy choice you can reach g. Finally, since S is optimal and the distance between g and s_2 is no more than the distance between s_1 and s_2, there is enough gas to get from g to s_2. The rest of S' is like S and thus legal.

Optimal Substructure Property: Let P be the original problem with an optimal solution S. Then after stopping at the station g at distance d_i the subproblem P' that remains is given by $d_{i+1}, \ldots, d_n$ (i.e. you start at the current city instead of St. Louis). Let S' be an optimal solution to P'. Since, $\text{cost}(S) = \text{cost}(S') + 1$, clearly an optimal solution to P includes within it an optimal solution to P'.

3. We give a counterexample demonstrating that the greedy algorithm does not yield an optimal solution. Consider the problem of making change for 30 cents. The optimal solution is to give three dimes. However, the greedy algorithm first chooses a 25 cents piece, and is then forced to use 5 pennies, leading to a non-optimal solution.

4. Here is the algorithm. Put as many songs (from song 1 to song g) on the first CD as possible without exceeding the m minute limit. Then recursively repeat this procedure for the remaining CDs. Since this just requires keeping a running sum in which each of the n song lengths are included once, clearly the above is an $O(n)$ algorithm. We now prove it is correct.

   Greedy Choice Property: Let S be an optimal solution in which the first CD holds songs 1 to k. The greedy algorithm puts songs 1 to g on the first CD. We now prove there is an optimal solution which puts the first g songs on the first CD. If $k = g$ then S is such a solution. Now suppose that $k \neq g$. Since the greedy algorithm puts as many songs on the first CD as possible, it follows that $k < g$. We construct a solution S' by modifying S to move all songs from $k+1$ to g onto the first CD. We now argue that S' is a legal solution which is optimal. First note that the number of CDs used by S' is at most the number of CDs used by S. All that remains is to argue that S' is legal (i.e. no CD holds more than m minutes of songs). By definition of the greedy choice the first CD can hold songs 1 to g. Since S is a legal solution all of the CDs in it hold at most m minutes. For all but CD 1 the number of songs is only reduced and hence must still hold at most m minutes. Hence S' is legal.

   Optimal Substructure Property: Let P be the original problem with an optimal solution S. Then after putting songs 1 to g on CD 1, let P' be the subproblem of songs $(g+1)$ to n. Let S' be an optimal solution to P'. Since, $\text{cost}(S) = \text{cost}(S') + 1$, clearly an optimal solution to P includes within it an optimal solution to P'.

5. We first argue that there always exists an optimal solution in which all of the events start at integral times. Take an optimal solution S — you can always have the first job in S start at time 0, the second start at time 1, and so on. Hence, in any optimal solution, event i will start at or before time $\lfloor t_i \rfloor$.

   This observation leads to the following greedy algorithm. First, we sort the jobs according to $\lfloor t_i \rfloor$ (sorted from largest to smallest). Let time t be the current time being considered (where initially $t = \lfloor t_1 \rfloor$). All jobs i where $\lfloor t_i \rfloor = t$ are inserted into a priority queue with the profit g_i used as the key. An extractMax is performed to select the job to run at time t. Then t is decremented and the process is continued. Clearly

the time complexity is $O(n \log n)$. The sort takes $O(n \log n)$ and there are at most n insert and extractMax operations performed on the priority queue, each which takes $O(\log n)$ time.

We now prove that this algorithm is correct by showing that the greedy choice and optimal substructure properties hold.

Greedy Choice Property: Consider an optimal solution S in which $x+1$ events are scheduled at times 0,1, ..., x. Let event k be the last job run in S. The greedy schedule will run event 1 last (at time $\lfloor t_1 \rfloor$). From the greedy choice property we know that $\lfloor t_1 \rfloor \geq \lfloor t_k \rfloor$. We consider the following cases:

Case 1: $\lfloor t_1 \rfloor = \lfloor t_k \rfloor$. By our greedy choice, we know that $g_1 \geq g_k$. If event 1 is not in S then we can just replace event k by event 1. The resulting solution S' is at least as good as S since $g_1 \geq g_k$. The other possibility is that event 1 is in S at an earlier time. Since $\lfloor t_1 \rfloor = \lfloor t_k \rfloor$, we can switch the times in which they run to create a schedule S' which has the same profit as S and is hence optimal.

Case 2: $\lfloor t_k \rfloor < \lfloor t_1 \rfloor$. In this case, S does not run any event at time $\lfloor t_1 \rfloor$ since job k was its last job. If event 1 is not in S, the we could add it to S contradicting the optimality of S. If event 1 is in S we can run it instead at time $\lfloor t_1 \rfloor$ creating a schedule S' that makes the greedy choice and has the same profit as S and is hence also optimal.

Optimal Substructure Property: Let P be the original problem of scheduling events $1, \ldots, n$ with an optimal solution S. Given that event 1 is scheduled first we are left with the subproblem P' of scheduling events $2, \ldots, n$. Let S' be an optimal solution to P'. Clearly $\text{profit}(S) = \text{profit}(S') + g_1$ and hence an optimal solution for P includes within it an optimal solution to P'.

6. We prove that the greedy algorithm described in problem 1, does yield an optimal here. It is easily seen that the greedy algorithm can be implemented in $O(n)$ time. This is all that was needed as far as the time complexity in order to get full credit. For those interested, you could also give an $O(1)$ solution as follows. First compute $x = n \text{mod} 50$. Then simply use $2x$ quarters followed by additional coins given by looking up the solution in a table with solutions for $n < 50$ cents.

   Greedy Choice Property: We show that there exists an optimal solution for making n cents that begins with the largest coin whose value is $\leq n$ cents. Let S be the set of coins given as change by some optimal solution to the problem of making change for n cents. Suppose that the largest denomination among the coins in S is $k \in \{Q, D, N, P\}$. Let i be the first coin chosen by the greedy algorithm. If $k = i$ then the solution begins with a greedy choice. If $k \neq i$ then we must show that there is another optimal solution S' that begins with coin i. Observe that since the greedy algorithm picks the largest denomination that can be used, it follows that $k < i$. Thus $i \neq P$. We now consider each of the remaining cases.

   Case 1: $i = N$. Thus the solution S must have value of 5 cents or more. Since the largest coin in S must be less than a nickel, S uses only pennies, and hence must

have at least 5 pennies. Create S' from S by replacing 5 pennies by a nickel. $|S'| < |S|$ contradicting the optimality of S.

Case 2: $i = D$. Thus the solution S must have value of 10 cents or more. Since the largest coin in S must be less than a dime, S uses only pennies and nickels. The only way to create 10 cents or more with pennies and nickels must use either 2 nickels, 1 nickel and 5 pennies, or 10 pennies. In all three cases we can create S' from S by replacing this subset of coins that equals 10 cents by a dime. Again, $|S'| < |S|$ contradicting the optimality of S.

Case 3: $i = Q$. First suppose, that there is some combination of coins in S that sum to 25 cents. Then let $S' = S - \{coins\ that\ sum\ to\ 25\} \cup \{Q\}$ and observe $|S'| < |S|$. Next we consider the case when no subset of coins in S sums to 25. This can only occur if there are at least 3 dimes in S since that is required to obtain 30 cents or more without using a nickel or at least 5 pennies (in which case some subset of coins would add to 25 cents). So in this case let $S' = S - \{D + D + D\} \cup \{Q + N\}$. Again $|S'| < |S|$ contradicting the optimality of S.

Hence in all three cases we proved not only that there exist some optimal solution that picks the same first coin as the greedy algorithm, but in fact, every optimal solution must pick this largest coin.

Optimal Substructure Property: Let P be the original problem of making n. Suppose the greedy solution starts with k cent coin. Then we have the subproblem P' of making change for $n - k$ cents. Let S be an optimal solution to P and let S' be an optimal solution to P'. Then clearly $\text{cost}(S) = \text{cost}(S') + 1$, and thus the optimal substructure property clearly follows.

7. The algorithm we use is at each point in time to schedule an available job with the least amount of remaining processing time. We implement this as follows. Let PQ be a priority queue which will be implemented as a binary heap.

   Create list L of all jobs sorted by start times
   Let t be the minimum start time of all jobs
   While L is not empty
   Remove all jobs with minimum start time from L
   Insert these jobs into PQ with a key of the processing time
   Let p =PQ.min()
   Let job i be a job in PQ with a minimum key (remaining processing time)
   Let s be the minimum start time of jobs remaining in L
   Run job i from time t to $t' = min(s, p)$
   Decrease the key of job i by $t' - t$
   If key for job i is 0 then remove job i from PQ
   Let $t = t'$

   The time to create L is $O(n \log n)$. The body of the while loop takes $O(\log n)$ time. The number of iterations of the loop is bounded by $2n$ since a change in t is made only

when a job completes or a new job arrives. Hence the main loop has a total execution time of $O(n \log n)$ giving an overall time complexity of $O(n \log n)$.

We now prove that the greedy algorithm is optimal here. We first argue that there exists an optimal solution that schedules some job at earliest arrival time t. Take an arbitrary optimal solution S_* in which the first job is scheduled at time $t' > t$. Let job i be a job with arrival time of t. Job i must be run somewhere in S_*. By moving a portion of i to run from t to t' (or until job i completes) we obtain a solution S which clearly has cost at most that of S_*.

Greedy Choice Property:

Let J_g be the first job schedule in the greedy solution and assume it is scheduled from t to t_g (at which time it is either completed or execution is preempted). As argued above, there must be some optimal solution S which schedules some job at time t. Let J_s be the first job scheduled in optimal solution S which runs from time t to t_s. We now argue that there is some optimal solution in which job J_g is run from time t to $t' = min(t_g, t_s)$.

Suppose not. That is, assume $J_g \neq J_s$. Let time t'' be the time in S when job J_g is completed. We proceed using a proof by cases.

Case 1: Job J_s completes after time t''. By definition of t'', job J_g completes by then. Thus at least $t' - t$ units of J_g must be run in S by time t''. We obtain S' from S by interchanging the last $t' - t$ units of J_g with the units of J_s run from t to t'. The completion time of J_g is only improved by this. Further, since J_s completes after t'' its completion time is unchanged. Hence $cost(S') \leq cost(S)$ and hence S' is an optimal schedule which makes the greedy choice (i.e. schedules job J_g from t to t').

Case 2: Job J_s completes before time t''. By the greedy choice property from t to t'' J_s must run at least as long as J_g or otherwise J_s would have been run instead of J_g in the greedy algorithm. Hence we can create schedule S' from S by interchanging the portion of the schedule when either J_g or J_s run in S to first run all of J_g and then run all of J_s. The completion time for J_g in S' is at most the completion time for J_s in S. Furthermore, the completion time for J_s in S' must be equal to the completion time of J_g in S. Hence $cost(S') \leq cost(S)$.

Optimal Substructure Property: Let P be the original problem. Subproblem P' is obtained by reducing the processing time of J_g by $t' - t$ and changing the start time of all jobs that are less than t' to be t'. (When the processing time of a job is 0 it is removed from the subproblem.) Let S be an optimal solution to P and let S' be an optimal solution to P'. We first consider when J_g complete by time t'. In this case

$$cost(S) = cost(S') + t'.$$

The other possibility is that J_g does not complete by time t'. In this case

$$cost(S) = cost(S').$$

Clearly in both cases, for a solution S which minimizes $cost(S)$, S' contained within S must minimize $cost(S')$ and thus the optimal substructure property follows.