Homework 4

Frederick Robinson

4 February 2010

1 Problem 6.3

1.1 Counterexample

Consider an ordered graph with edges $\{n_1, n_2, \dots, n_5\}$ and connections $\{n_1, n_2\}$, $\{n_1, n_3\}$,$\{n_2, n_5\}$, $\{n_3, n_4\}$, $\{n_4, n_5\}$.

The greedy algorithm will pick the $\{n_1, n_2\}$ edge first, and then be forced to go to n_5 from there, in a total of only 2 steps. However, the only other possibility is better, $\{n_1, n_3\}\{n_3, n_4\}\{n_4, n_5\}$ takes 3 steps instead.

1.2 Algorithm

We use a dynamic programming algorithm, taking advantage of the fact that if we memoize the longest path from a given node to the end we need only compute a relatively small amount of things for each node.

Initialize an array *memo* of length n with -1 as each value but the last, which we set to 0. Also we use a variable *temp* with initial value -1.

```
Distance($node$)

1. For each edge of $node$
    (a) Set $x$ with the number of the node pointed to by the edge
    (b) If the entry of $memo[x]$ is -1 call Distance($x$).
    (c) If $1+memo[x] > temp$ set $temp = memo[x] + 1$.
2. Set $memo[node] = temp$ then clear $temp$ by setting $temp = -1$
```

In order to compute the maximal distance from the first node to the last just call Distance[1] and then read out the value of *memo*[1] when the program terminates.

1.3 Correctness

I claim that a call to Distance[i] returns the correct value if all necessary values of Distance[k] with $k > i$ have been set.

Proof. We use complete (strong) mathematical induction. So, assume that *memo*[k] gives the longest path from k to n for each $k > i$. Then, since the longest path from i is 1 more than the longest path from a node to which i is connected our algorithm correctly finds this value. After all Step 1 just computes the maximum of all such values.

Base Case: When our algorithm starts it is the case that each value of memo is either unset or correct since in particular only the value corresponding to the last node is set, and that value is set correctly to 0. □

It now remains only to establish that the algorithm terminates. This is easy to see however since Distance[i] is called at most once for each node $i \leq n$. After Distance finishes for a given node it stores the computed value in the array, and so Distance is never again called for that node.

So, since a complete run of the program requires, in the worst case a call of distance for each of the nodes, it must terminate.

1.4 Runtime

The algorithm above runs in $O(n^2)$ time in the worst case. Distance is called at most once for each node i. Also, since we know that a node may only be connected to a node of greater value, a given run of distance may at most access and compare the memo entry corresponding to every node $k > i$. This access to the memo array takes n^2 times then. Therefore, the algorithm runs in $t = n^2 + n = O(n^2)$

2 Problem 6.5

2.1 Algorithm

We take input in the form of a string $y_1 y_2 \ldots y_n$ and output a segmentation of the string which maximizes the total quality. To do this we utilize two memoing arrays one called *value* of length $n + 1$ which we initialize to $-\infty$ except for the last element which will be initialized to 0, and another called *nextword* which we initialize to 0. Additionally we'll use a variable called *counter* to keep track of where we are in the string and variables *temp*, *wheretemp*, and x for temporary information. Initialize $counter = n$ and $temp = -\infty$.

The algorithm we use is an iterative implementation of a dynamic programming algorithm.

```
1. While $counter > 0$
    (a) For each $x > counter$ with $x \leq n + 1$
```

```
            i. If $temp < quality(counter, counter+1, \ldots, x-1) + value[x]$ then
               set $temp = quality(counter, counter+1, \ldots, x-1) + value[x]$ and
               $wheretemp = x$
        (b) Set $value[counter] = temp$ and $nextword[counter] = wheretemp$
        (c) Deincrement $counter$
    2. EndWhile
```

After this loop has run the maximal value of any segmentation will be stored in $value[1]$ and the segmentation can be read out in the following manner (Initialize $x = 1$)

```
    1. While $x \leq n$
        (a) Output(x)
        (b) Set $x = nextword[x]$
    2. EndWhile
```

2.2 Correctness

Since this is a dynamic programming algorithm we will again use Strong Mathematical Induction to prove its validity

Proof. Assuming that the memo array $value[n]$ $n > k$ contains the maximum value of any segmentation of the letters from n to the end our algorithm will compute the maximal value of any segmentation of the letters from k to the end.

Since any optimal segmentation of $k \ldots$ is a word including the first character together with an optimal segmentation of some subset of this string our search will find it. Moreover, since we set $value[n+1] = 0$ if the optimal segmentation is the entire string as a word, this too will be found.

Base Case: When we start the algorithm all of the information in $value$ is correct since it doesn't start out with any information. We see in particular that, $value$ contains the values of the optimal segmentation for each n on since n is initialized with the value of the last character and the only segmentation of this character is the character itself. This is identified and stored in the value array. □

2.3 Runtime

The algorithm runs in $O(n^2)$ time since it memoizes information about the optimal segmentation of a given substring form i to n at most once. Moreover for a given calculation of this segmentation it checks once for each $i < j \leq n$. Since this latter takes on the order of n^2 calls to Quality and the former takes n storage actions the algorithm is $t = n^2 + n = O(n^2)$

3 Problem 1

3.1 Algorithm

We will use the following algorithm to compute the optimal advertising selection given an input. It follows a dynamic programming approach. We will maintain two arrays of length n, one initialized to 0s called *memo* and another, also initialized to 0s called *lastcommercial*. Lastly we will use a variable called temp.

The algorithm compute the value of the value maximizing set of commercials with maximal duration element x and stores this in $memo[x]$. Then, at the end it reads off the members of the maximum value such set.

```
1. Sort the bids in ascending order by duration
2. For each bid $x$ from 1 to $n$
   (a) Set $temp = \text{value}(x)$ and set $templast = x$
   (b) For each bid $i$ from 1 to $x - 1$
       i. If $memo[i] + \text{value}(x) > temp$ and $\text{value}(i) \leq \text{value}(x)$ set $temp = memo[i] + \text{value}(x)$ and $templast = i$
   (c) Store $memo[x] = temp$ then reset $temp$ by putting $temp = 0$
   (d) Store $lastcommercial[x] = templast$ then reset $templast$ by putting $templast = 0$
3. End For Each
4. Set $temp = 0$
5. For $x$ from 1 to $n$
   (a) If $memo[x] > temp$ then set $temp = memo[x]$ and $templast = x$.
6. End For
7. While $templast > 0$
   (a) Output $templast$ them set $templast = lastcommercial[templast]$
8. End While
```

3.2 Correctness

The algorithm compute the value of the value maximizing set of commercials with maximal duration element x and stores this in $memo[x]$.

If we begin by assuming that the algorithm has done this correctly for each $m < n$ it is easy to show that it does this correctly for the nth element

Proof. The selection of commercials with maximal duration element n may not include any elements with $m > n$ since the list is ordered by duration

Moreover, the only feasible choices with $m < n$ are those which have price less than the price of n to satisfy the fairness constraint.

Since the selection of commercials with maximum duration element n must have a next-smaller duration element the selection must in particular be n together with a solution to one of the subproblems corresponding to $m < n$. Since the solution to such a subproblem excludes commercials with price greater than m it in particular has no commercials with price greater than n if and only if the price of m is less than the price of n.

This established we see that we are correct to set the solution of the subproblem corresponding to n as n together with the maximum value solution to a subproblem $m < n$ which has price less than or equal to n.

Base Case: To begin with the algorithm correctly identifies that the value maximizing selection containing commercials whose duration is less than the commercial of least duration is just that commercial by itself

Thus, by strong induction our algorithm has the property claimed. □

Moreover, since the optimal arrangement of commercials must have some element with maximal duration x the optimal choice must be represented in the memo array.

This implies that our algorithm finds the best arrangement of commercials. Since all throughout it has been maintaining a list of the next shorter commercial we can obtain the set corresponding to this best arrangement by simply working backwards from the commercial with the highest memo value.

3.3 Runtime

The sort at the beginning takes $n \log n$ time. Next, the table generating portion of the algorithm generates a table value for each of the n element. Generating such a value takes m checks of the table for the mth element. So together the table generating step takes n^2 time. The scan through the list to find the maximal element takes time on order n. The procedure to read out the set also takes time on order n. Thus, all together we get $T = n \log n + n^2 + 2n = O(n^2)$.

3.4 Algorithm

The algorithm employed in this more complex version of the problem is an extension of the previous algorithm. We will employ the same basic algorithm with some modifications so as to limit the time of the advertisements picked. This time however we take a recursive approach. The routine opt produces the optimal arrangement of the first n advertisements given a total time budget of C. In particular it returns the value of said optimal selection. In keeping with the more complex nature of the problem we need to memoize more information. Towards this initialize a two dimensional array *memo* with dimension $n \times c$ and

-1 in all positions and another two dimensional array *last* with dimension $n \times c$ and -1 in all positions

```
1. Sort the bids in ascending order by duration
2. Set variables say $maximalvalue = 0$ and $place = 0$
3. For each $m$ from 1 to $n$
4. If $maximalvalue < \text{Opt}(n, C) \leq C$ then let $maximalvalue = \text{Opt}(n, C)$ and let $place = n$
5. While $place \neq 0$
6. Output $place$
7. set C = C-value(place)
8. Set $place = last(place, C)$
9. End While
```

Opt(n,C)

```
1. If $n = 1$ and value(1) $\leq C$ return 1
2. If $n = 1$ and value(1) $> C$ return 0
3. if memo$(n, C) \neq -1$ return memo$(n, C)$
4. Set $temp$ = value$(n)$
5. Set $placetemp = n$
6. for x from n-1 to 1
   (a) if value$(x) \leq$ value (n)
       i. If memo$(x, c - value(n)) = -1$ then Set memo$(x, c - value(n))$=Opt$(x, c - value(n))$
       ii. If memo$(x, c - value(n)) + value(n) > temp$ set temp = value(n)+memo(x,c-value(x)) and set placetemp = x
7. Set last$(n, C) = placetemp$
8. Return temp
```

3.5 Correctness

First observe that the solution to a given problem (n,C) to find the maximal profit using advertising that is only shorter or equal in duration than n and using at most C time while obeying the fairness condition is found in the form of another such problem. Namely, finding the maximal profit using advertising shorter or equal in duration to n, with a total time of C-time(n) and obeying the fairness condition.

This observation being made we seek to prove that our algorithm does this, and memoizes this information as "memo(n,C)"

Proof. Since we know that n must be in the set of advertisements corresponding to memo(n,C) all that remains to be verified is that the rest of the advertisements are those which optimize profit, pursuant to the fairness constraint and time limit (less the time needed for n)

This is precisely what our algorithm does. Since each value of opt(m,c-value(n)) for m¡n must not use too much time, and as proved above the fact that m is less in value than n ensures that the fairness condition is upheld we need only maximize profit.

This is precisely what we do though. We check over each such opt, and select that one which maximizes profit.

□

Now that we have proven that our memo contains a table of optimal values it is clear that we produce the optimal solution over all. In particular there must be some least duration for advertisement in the optimal solution. Since our table contains information for any least duration it must be that it contains information about the optimal one. Since we maximize over all the possible least durations we will of necessity find the optimal one.

Similarly, since we just keep track of the next least (duration) element of our solution and follow this backwards we must output the correct elements in the optimal set.

It is worth noting that although the table may include information about solutions which are too long it does not contain information about solutions which are optimal and too long, only those whose initial element is larger than the entire set C. This is okay however, since when we are sorting through the table we are sure to exclude such solutions.

3.6 Runtime

The proof of runtime is very similar to that used in the previous part. However, instead of memoizing just n items we are potentially memoizing as many as $c \cdot n$ things. Thus, the runtime breaks down as follows:

The sort at the beginning takes $n \log n$ time. Next, the table generating portion of the algorithm generates a table value for each of the n element. Generating such a value for fixed c takes m checks of the table for the mth element. So together the table generating step takes cn^2 time. The scan through the list to find the maximal element takes time on order n. The procedure to read out the set also takes time on order n. Thus, all together we get $T = n \log n + cn^2 + 2n = O(cn^2)$.

4 Problem 2

4.1 A Proposition

Theorem 4.1. *If it is possible to add a set of advertisements which would break the fairness condition to a set which is optimal (in terms of price maximization given a duration cap with unadjusted prices) while maintaining fairness and maximizing profit this is done optimally by adjusting each added price to the price of the next longer advertisement*

Proof. It is easy to see that adjusting the price of any of the newly added advertisements so as to be smaller than that of the advertisement which is next longer in duration is not optimal for, in particular adjusting the price so that it is equal to that of the next longer advertisement (if this is possible) will increase profit while maintaining fairness.

Moreover it is always possible to make this adjustment as, were it not, the price of the added advertisement would necessarily be smaller than that of the next larger advertisement or the next shorter advertisement. The former would imply that the added advertisement could have been in the original set to begin with without breaking fairness thereby increasing profit. This is a contradiction however as we assumed that the original set was optimal. The second case is a contradiction as well since it implies that the given advertisement can not be added to the given optimal set without removing at least the preceding element of the optimal set, again, decreasing the profit.

So, we have established that the price adjustment on a given member of the set to be added must lead to a price which is greater than or equal to that of the next longer advertisement which is already present in the set.

This established we will show that it must necessarily also yield a price which is less than or equal to the one of the next longer advertisement as well. For, take the two maximal elements of the set which is already optimal, which contain between them some nonzero collection of elements which we wish to add. Call them respectively $o_0, o_1, a_1, \ldots a_n$ Each a_i has $a_i \geq o_1$ as has been established previously. Suppose towards a contradiction that there is some adjustment of each a_i which would produce optimal $a'_1, \ldots a'_n$ so that each one of these is at least o_1. This however is a contradiction since, if this is the case it means in particular that

$$\sum_{i=1}^{n} a'_i \geq \sum_{i=1}^{m} o_i + \sum_{i=1}^{n} o_1 = n \cdot o_1 + \sum_{i=1}^{m} o_i$$

where m is the least member of O whose value is greater than $\max a_i$. (In words: "The value of the a's together exceeds the value of the next o_ns together with

n times o_1 " since in order to insert them at their new values we must delete each o_n $n \geq 1$ which does not exceed the maximal new a)

This summation implies that the mean value of the a'_i is greater than or equal to o_1 plus the mean of each o_i for $m \geq i \geq 1$. Moreover it means that since each $a'_i \leq a_i$ we have $\bar{a} \geq 2\bar{o}$ where bars represent the resepective means and $\min a_i \geq \min o_{m \geq i \geq 1}$.

So it must be that there are too few a_i to compensate for the necessary exclusion of each o_i $m \geq i \geq 1$ or that there are too many, and the set which we claimed to be optimal was not.

Hence, by induction no member of the optimal adjusted set may be greater than the next longer member of the original set.

□

Corollary 4.2. *Any optimal (including price adjustment) set taking into account adjusted pricing is the same as one without adjusted pricing with the addition of prices changed in the above way.*

Suppose there exists an algorithm which is not of the form above. The endpoint must be the same as in some optimal set as, if not then its price could be increased without breaking the fairness condition. Moreover there is some first element which is full price. These elements fixed there is no way to get a higher value than to first compute the optimal set without price change and then add in prices in the manner described above.

For suppose towards a contradiction there were. If the arrangement has some fully priced ads they must be those which would be created by picking the optimal fully priced ads. For, if not then there is a better way to pick the fully priced ads.

These fully priced ads fixed the remaining ones must be picked/price adjusted as described above by Theorem 4.1

4.2 Algorithm

Now that we have established the quite lengthy Theorem 4.1 there is a remarkably easy algorithm which we can run. Simply run the first algorithm from the previous problem then, for each of the unused advertisements purchase it at a price discounted as outlined in the Theorem. That is more formally

1. Run the previous algorithm and create a corresponding array of flags: 1 for in, 0 for out

2. For x from 1 to n

3. If the element x has flag 1 set $temp = item(x)$

4. If the element x has flag 0 and its monetary value is greater than $temp$ set its flag to 2

5. For x from n to 1

6. If the element x has flag 1 set $temp = item(x)$
7. If the element x has flag 2 set its monetary value to $temp$ set its flag to 1

4.3 Correctness

The correctness of this algorithm follows from Theorem 4.1 and the validity of the algorithm from the preceding problem. Moreover it is worth noting (and should be clear) that the algorithm from last problem will never lead to a situation in which there is some element at the end of the list which is greater in price than the last included advertisement and yet is not included. Were this the case then the solution would be suboptimal.

Thus, our algorithm is correct

4.4 Runtime

The runtime is just given by the runtime of the algorithm from last problem plus a few linear time terms created by price adjusting in our extension to the algorithm above.

Thus, our algorithm is $O(n^2)$ like the one from last problem.

4.5 Algorithm

This algorithm is again very similar to the one in the previous problem. The only difference in this case is that when we call opt we perform the procedure outlined in the previous part on the return: that is, instead of merely finding the optimal set costing less than C and using elements with duration less than n without taking into account the fact that we can change prices we do a procedure which is the same but with the following key difference.

When computing the price or duration of a set with more than one element first add elements in way outlined above.

4.6 Runtime

The procedure of adding elements as we outlined above is only linear however it must be computed for each time we add an element to the table. Hence we get runtime of $O(cn^3)$

4.7 Correctness

By Corollary 4.2 all optimal sets must be of the form checked. Hence, we compute the optimal solution to each subproblem and our algorithm is correct.