CSCI-GA.1170-001/002 Fundamental Algorithms November 12, 2014

Solutions to Problem 1 of Homework 8 (3(+5) points)

Name: Keeyon Ebrahimi *Due: Wednesday, November 12*

For each of the following suggested greedy algorithms for the ACTIVITY-SELECTION problem, give a simple example of the input where the proposed greedy algorithm fails to compute the correct optimal solution.

(a) (3 points) Select the activity a_i with the shortest duration $d_i = f_i - s_i$. Commit to scheduling a_i. Let S'_i consist of all activities a_j which do not overlap with a_i: namely, either $f_j \leq s_i$ or $f_i \leq s_j$. Recursively solve ACTIVITY-SELECTION on S'_i, scheduling the resulting activities together with a_i.

Solution:

This greedy approach will fail if the shortest activity disqualifies 2 other disjoint activities by overlapping both. Here is an example of this type of problem.

i	1	2	3	4	5
si	1	9	10	18	20
fi	10	12	20	22	34

In this example, the first block that will be picked is a_2, for it has the shortest duration of 3. The selection of a_2 disqualifies a_1 and a_3, for they have intersecting start and end times. The next shortest time will then be a_4 with a total time of 4. This will then disqualify a_5, which then in total gives this greedy algorithm a total of 2 activities.

We can see though that is is not the optimal, for the optimal is actually $a_1 \cup a_3 \cup a_5$
This has a total of 3 activities, while the previously suggested greedy algorithm only gives 2 activities. □

(b)* **(Extra Credit; 5 points)** For each activity a_i, let n_i denote the number of activities which do not overlap with a_i (e.g., n_i is the cardinality of the set S'_i defined above for specific a_i). Select the activity a_i with the largest number n_i of non-overlapping activities. Commit to scheduling a_i. Recursively solve ACTIVITY-SELECTION on the n_i activities in S'_i, scheduling the resulting activities together with a_i.
(**Hint**: Unlike part (a), you might need a lot of activities for this counter-example. The smallest I know uses $n = 11$ activities. So don't be discouraged if small examples are all bad.)

Solution: □

CSCI-GA.1170-001/002 Fundamental Algorithms November 12, 2014

Solutions to Problem 2 of Homework 8 (11 Points)

Name: Keeyon Ebrahimi *Due: Wednesday, November 12*

Consider the problem of storing n books on shelves in a library. The order of the books is fixed by the cataloging system and so cannot be rearranged. The i-th book b_i, where $1 \leq i \leq n$ has a thickness t_i and height h_i stored in arrays $t[1 \ldots n]$ and $h[1 \ldots n]$. The length of each bookshelf at this library is L. We want to minimize the sum of heights of the shelves needed to arrange these books.

(a) (5 points) Suppose all the books have the same height h (i.e., $h = h_i$ for all i) and the shelves are each of height h, so any book fits on any shelf. The greedy algorithm would fill the first shelf with as many books as we can until we get the smallest i such that b_i does not fit, and then repeat with subsequent shelves. Using either the Greedy Always Stays Ahead or Local Swap method, show that the greedy algorithm always finds the shelf placement with the smallest total height of shelves, and analyze its time complexity.

Solution:
With a constant height for all shelves, we have this equation for the Total Shelve Height.
S = Number of Shelves
h = constant
$TotalHeight = h * S$
Because h is constant, we need to minimize S to get the minimal $TotalHeight$

Proof by GASA
b_m = total number of books on shelf
$b_i(g) = i$ books on shelf in greedy approach
$b_i(z) = i$ books on shelf in optimal approach
S_i = Amount of shelves needed with i books.

First we will be showing that the approach of if it fits it sits puts the maximum books on a shelf.

1. $b_1(g) \geq b_1(z)$

   We know this by definition. If the first book fits, it sits. Which has atleast as many as the optimal, because there is no greater for 1.

2. $b_m(g) \geq b_m(z)$

   If this happens with one, this happens with m as well.

3. $b_{m+1}(z) \geq b_m(g) \geq b_m(z)$

4. $b_{m+1}(g) \geq b_{m+1}(z)$

This shows that we can fit the most books on the shelf by going with the approach, if it fits it sits.

Now we know

$$totalBooks = BooksOnShelves + BooksRemaining$$

Because $totalBooks$ is constant, the greater $BooksOnShelves$ is, the smaller $BooksRemaining$ will be. We have proven that our greedy approach gives the maximum $BooksOnShelves$, so we know need to prove that a smaller $BooksRemaining$ leads to smaller $TotalShelves$

Lets set S_i to be the total number of shelves it takes to store i books. By laws of nature we know that $S_1 \leq S_2 \leq ... \leq S_n$

We know that we have given the least remaining books with our greedy algorithm, which in turn has the least amount of shelves as shown directly above this, and we also know that the less Shelves we have the less our Total height will be, so with all this we know that our greedy algorithm gives the optimal number of shelves.

Running Time: All we have to do here is at every book, see if it will fit or not and then either place on existing shelf or next shelf. We only have to iterate through each book once in order to do this for all books, so because we only iterate through all books once, we know that we have a running time of

$$O(n)$$

□

(b) (6 points) Now assume that the books are not of the same height, and hence the height of any shelf is set to be the height of the largest book placed on that shelf. Show that the greedy algorithm in part (a) doesn't work for this problem. Give an alternative dynamic programming algorithm to solve this problem. What is the running time of your algorithm?

Solution:
Here is an example of when this will not work.
Lets say he have two heights, B for tall and b for short.

Let say our greedy algorithm first whatever we can and we get 2 shelves shaped like this.

$Shelf\ 1 \quad b \quad b \quad b \quad B$
$Shelf\ 2 \quad B$

The tallest in both shelves is B, so the total height would be $2B$.
There is a better solution though. What if we placed the B at the end of shelf on on the next shelf. We would get this solution.

$Shelf\ 1 \quad b \quad b \quad b$
$Shelf\ 2 \quad B \quad B$

Which gives us a total Height of $b + B$, which is less than $B + B$, so the greedy approach is not optimal.

Here is the algorithm to figure out the best solution. We set up a Memoization Array TH with $TH[0] = 0$ and $TH[1..n] =$ infinity.

If the next book will not fit on the current shelf, we put it on the next Shelf.
If the book does fit, then we must look at the books height vs the Current Shelf Height. If book height, $h[i]$ is less than Current Shelf Height, then we just place the book on the shelf.

If the book height does fit, but it's height is greater than the current Row Height. We have to make a decision.
We pick Min(Placing book on current shelf, Placing book on next shelf), which is where the recursion and memoization come in to help.

This will have a running time of $O(n^2)$, which is significantly better than the 2^n running time we would usually have if we didn't use memoization. It is n^2 because for each problem n, we have to look at all previous books and check if it is better for them to go on the current row or to start a new row off.

□

CSCI-GA.1170-001/002 Fundamental Algorithms November 12, 2014

Solutions to Problem 3 of Homework 8 (12 Points)

Name: Keeyon Ebrahimi *Due: Wednesday, November 12*

You want to travel on a straight line from from city A to city B which is N miles away from A. For concreteness, imagine a line with A being at 0 and B being at N. Each day you can travel at most d miles (where $0 < d < N$), after which you need to stay at an expensive hotel. There are n such hotels between 0 and N, located at points $0 < a_1 < a_2 < \ldots < a_n = N$ (the last hotel is in B). Luckily, you know that $|a_{i+1} - a_i| \leq d$ for any i (with $a_0 = 0$), so that you can at least travel to the next hotel in one day. You goal is to complete your travel in the smallest number of days (so that you do not pay a fortune for the hotels).

Consider the following greedy algorithm: "Each day, starting at the current hotel a_i, travel to the furthest hotel a_j s.t. $|a_j - a_i| \leq d$, until eventually $a_n = N$ is reached". I.e., if several hotels are within reach in one day from your current position, go to the one closest to your destination.

(a) (6 points) Formally argue that this algorithm is correct using the "Greedy Stays Ahead" method.
(**Hint**: Think how to define $F_i(Z)$. For this problem, the name of the method is really appropriate.)

Solution: Proof by GASA
$h_i(g) = i$ Hotel number Visited on Day i by Greedy Solution $h_i(z) = i$ Hotel number visited on Day i by optimal solution

1. $h_1(g) \geq h_1(z)$

   We know this because of definition. We go to the furthest one each day in the greedy approach, so we go the the furthest, so greedy $\geq$ optimal for day 1.

2. $h_m(g) \geq h_m(z)$

   Same logic as step 1. If we know it is true for day 1, we know it is also true for day m

3. $h_{m+1}(z) \geq h_m(g) \geq h_m(z)$

   The optimal hotel number we visit on day $m + 1$ has to be $\geq$ the hotel number we visit on day m with both the greedy and optimal approach.

4. $h_{m+1}(g) \geq h_{m+1}(z)$

This step shows us finally that for the entire solution, greedy does best.

□

(b) (6 points) Formally argue that this algorithm is correct using the "Local Swap" method. More concretely, given some hypothetical optimal solution Z of size k and the solution Z^* output by greedy, define some solution Z_1 with the following two properties: (1) Z_1 is no worse than Z; (2) Z_1 agrees with greedy in the first day travel plan. After Z_1 is defined, define Z_2 s.t.: (1) Z_2 is no worse than Z_1; (2) Z_2 agrees with greedy in the first two days travel plan. And so on until you eventually reach greedy.

Solution:

1. Z_1 picks the furthest we can go on day 1. This can't be worse than the optimal, because it cannot be less then the optimal.

2. Z_1 agrees with optimal because there is no further you can go for option 1 than the greedy so Z_1 agrees with greedy.

3. Z_2 First two days selected greedy wont be worse than the optimal because it is the furthest possible for 2 days, so it cannot be less than optimal.

4. Z_2 agrees with Optimal Z because for the only 2 days that are different, we are picking the furthest we can go, which is the definition of optimal

5. Z_k is the optimal. We keep on moving one by one and see that with each replaced item, the greedy is the optimal. For every element greedy replaced works as good as optimal, making it the optimal

□

CSCI-GA.1170-001/002 Fundamental Algorithms November 12, 2014

Solutions to Problem 4 of Homework 8 (10 points)

Name: Keeyon Ebrahimi *Due: Wednesday, November 12*

Recall, Fibonacci numbers are defined by $f_0 = f_1 = 1$ and $f_i = f_{i-1} + f_{i-2}$ for $i \geq 2$.

(a) (2 points) What is the optimal Huffman code for the following set of frequencies which are the first 8 Fibonacci numbers.

Solution:
Lets first look at the frequency of each element in a Fibonacci Sequence

Value	0	1	2	3	4	5	6	7	8
Frequence	13	21	13	8	5	3	2	1	1

Now we need to keep on adding the two minimum up and creating a tree out of those two elements. We keep doing this until we get the full tree.

What makes this tree not be a tree with a left element always having only one element is the fact that Value 1 has a higher frequency than value 0. This makes it so from our top root node, the left subtree contains 0 and 2, while the right subtree contains the rest of the elements in incremental order with at each sublevel, the left tree has the next min while the right tree has the remaining elements in a tree until we get to leaves 7 and 8. Here is the encoding for Fibonacci sequence

Value	0	1	2	3	4	5	6	7	8
Encoding	00	10	01	110	1110	11110	111110	1111110	1111111

□

(b) (4 points) Let $S_1 = 2 = f_0 + f_1$ and $S_i = S_{i-1} + f_i = \ldots = f_i + f_{i-1} + \ldots f_1 + f_0$ (for $i > 1$) be the sum of the first i Fibonacci numbers. Prove that $S_i = f_{i+2} - 1$ for any $i \geq 1$.

Solution:
We are going to prove this by induction. We need to prove that

$$f_{i+2} - 1 = f_i + f_{i-1} + ... + f_0$$

1. Prove base case of when $i = 1$

$$f_3 - 1 = f_1 + f_0$$
$$2 = 2$$

This is true, so base case passes, we may now move on to f_k

2. We need to find the equation for f_k which is

$$f_{k+2} - 1 = f_k + f_{k-1} + ... + f_0$$

3. Time for inductive step. We assume that this is true up to k, we must prove for $k + 1$

$$f_{k+2+1} - 1 = f_{k+1} + f_k + ... + f_0$$

For $f_k + ... + f_0$, we replace this value with the left hand side of equation from item 2, for we assume is correct.

$$f_{k+2+1} - 1 = f_{k+1} + f_{k+2} - 1$$

This equals

$$f_{k+3} = f_{k+2} + f_{k+1}$$

This is the definition of a Fibonacci sequence, so we know that we have come to the correct value

□

(c) (4 points) Generalize your solution to part (a) to find the shape of the optimal Huffman code for the first n Fibonacci numbers. Formally argue that your tree structure is correct, by using part (b).

Solution:
Here is a generalized version the optimal Huffman Code for n Fibonacci numbers.

Value	0	1	2	3...(n-1)	n
Encoding	00	10	01	(value - 1) 1's + 0	(n-1) 1's

The last two columns might not make sense, but this is what I mean. Lets pretend that $n = 5$ Elements $3..n - 1$ or 3 and 4 will be their (value - 1) 1's with a 0 concatenated at the end. So $3 = 110; 4 = 1110$; and this repeats up to $n - 1$. Then for value n, we get $n - 1$ 1's. So if $n = 5$, $5 = 1111$.

To show this. we look at

$$S_k = S_k - 1 + f_k$$

This shows that the change at every level is a change of f_k. The definition of this is

$$f_k = f_{k-1} + f_{k-2}$$

Number smaller will increasing at the repeating rate of the Fibonnacci sequence. This pattern will be matched with each next level of element, so our key will work throughout.
Here is an inductive proof of the sum of each new level actually equals the next level of Fibonacci

1. Prove:
$$S_{i-1} + f_i = f_i + f_{i-1} + ... + f_0$$

2. Prove for $i = 2$
$$S_1 + f_2 = f_2 + f_1 + f_0$$
$$2 + 2 = 2 + 1 + 1$$
$$4 = 4$$

This shows that we pass the base case. Now need to show the k step

3.
$$S_{k-1} + f_k = f_k + f_{k-1} + ...f_0$$

4. We now assume that up to k is true, and we do an inductive step of $k + 1$

5.
$$S_k + f_{k+1} = f_{k+1} + S_{k-1} + f_k$$
$$S_k = S_{k-1} + f_k$$

This is our original equation proving that at each level, the sum goes up by that Fibonacci value, which keeps our pattern for our general encoding. □

CSCI-GA.1170-001/002 Fundamental Algorithms November 12, 2014

Solutions to Problem 5 of Homework 8 (14 Points)

Name: Keeyon Ebrahimi *Due: Wednesday, November 12*

Little Johnny is extremely fond of watching television. His parents are off for work for the period $[S, F)$, and he wants to make full use of this time by watching as much television as possible: in fact, he wants to watch TV non-stop the entire period $[S; F)$. He has a list of his favorite n TV shows (on different channels), where the i-th show runs for the time period $[s_i, f_i)$, so that the union of $[s_i, f_i)$ fully covers the entire time period $[S, F)$ when his parents are away.

(a) (10 points) Little Johnny doesn't mind to switch to the show already running, but is very lazy to switch the TV channels, and so he wants to find the smallest set of TV shows that he can watch, and still stay occupied for the entire period $[S, F)$. Design an efficient $O(n \log n)$ greedy algorithm to help Little Johny. Do not forget to carefully argue the correctness of your algorithm, using either the "Greedy Always Stays Ahead" or the "Local Swap" argument.

Solution:
The algorithm we are going to pick a show that is currently running and it ends last. Mathematically we will

1. For all i, find shows where $s_i \leq S_i \leq f_i$ where $f_i - S_i$ is greatest.
2. Run algorithm again, but we update S_i to chosen show's f_i, which eliminates all shows that end before that f_i

We are going to be going through each show, checking if they are running, but with each update, we are eliminating all shows that end before chosen activities, making our sample sizes smaller and smaller by a variable amount.

This gives us the running time of $O(n\ log\ n)$

Now we want to prove that this is optimal
Proof by GASA
To prove this, we need to set up some variables R_i = Remaing Time after watching of channel i.
So, if we watched channel 1 for 20 units, we would set $R_1 = [S, F) - 20$.
Now we will set a difference for Greedy and optimal solution

$R_i(g)$ = Time remaining after watching i channels with greedy approach
$R_i(z)$ = Time remaining after watching i channels with optimal approach
S_i = Shows watched in time of i units.

1.
$$R_1(g) \leq R_1(z)$$

Prove for Initial case that greedy is atleast as good as optimal
We know that $R_1(g) \leq R_1(z)$ because we are picking the largest possible time unit for the greedy solution, so by definition, the remaining has to be the least possible amount. Because we are reducing the total first by the max for the first step, we know that $R_1(g) \leq R_1(z)$

2. $$R_{m+1}(z) \leq R_m(g) \leq R_m(z)$$

   This shows that the remaining time of watching $k+1$ episodes the optimal way is going to be less than the total time of watching only k channels both the greedy and the optimal way. Because of Step 1, we also know

   $$R_m(g) \leq R_m(z)$$

3. $$S_{i-1} \leq S_i$$

   This is a very important step. We show here that the amount of shows we watch in less time is going to be either the same or less than the amount of shows watched in more time. This gives importance to our R_i value, for now we know that R_i has a direct correlation with shows watched, or S_i. We know

   $$S_i \leq S_{i+1}$$

   which in turn means that

   Total Channels watched at $R_i(z) \geq$ Total Channels watched at $R_{i+1}(z)$, which is why we want to minimize our R_i, or remaining time left.

4. $$R_{m+1}(g) \leq R_{m+1}(z)$$

   Now we know that for all options, the remaining time of the greedy algorithm will be less or equal to the optimal solution. We know that the smaller the remaining time the less channels will be watched which is better. We know this thanks to Step 3.

□

(b) (4 points). Assume now that Little Johnny will only watch shows from the beginning till end (except show starting before S or ending after F), but now he fetches another TV from the adjacent room, so that he can potentially watch up to two shows at a time. Can you find a strategy that will give the smallest set of TV shows that he can watch on the two TVs, so that at any time throughout the interval $[S, F)$ he watches at least one (and at most two) shows. (**Hint**: Try to examine your algorithm in part (a).)

Solution: $Max(F_{i+1} - S_{i+1} - (F_i - S_{i+1}))$ where $S_{i+1} \leq F_i$

Here is what that is saying in regular words.

1. Initially turn on TV 1 to show that already running and has the latest finish time for f_1
2. Find the show 2 that starts before or at the same time as show 1 ends, and has the longest playing time after show 1 ends.
3. Play this show on the opposite tv
4. Repeat this process for the remaining i steps.

□