Algorithms
Computer Science 140 & Mathematics 168
Spring 2007
Homework 3b
Due Tuesday, February 6

- This is a challenging problem set but an important one as well - it will help build up your "dynamic programming" muscles! *Please start early.*
- You are encouraged, but no longer required, to use LaTeX to typeset your solutions to assignments in this course.
- This week we are covering Chapter 6 (sections 1 through 5) in our book.

1. [**20 Points**] **Coin Return!** You've been hired by International Coinage Machines (ICM), a manufacturer of high-performance cash registers. ICM has decided that its new cash registers should tell a cashier how to make change using the least number of coins possible for the particular collection of coin types in the local monetary system.

   For example, The People's Republic of Shmorbodia has a 1 cent coin, a 5 cent coin, a 30 cent coin, a 40 cent coin, and a 60 cent coin. If the ICM cash register is asked how to optimally make change for 70 cents, it will report that the least number of coins possible is 2 and that the cashier should use a 40 cent coin and a 30 cent coin.

   Your colleague at ICM, Dr. I.M. Greedy has proposed the following "greedy" change algorithm for making change for amount n: Find the largest coin value less than or equal to n. Call this coin value c. Issue one c coin and then repeat the process for amount $n - c$.

   Notice that the greedy algorithm is not optimal for the Shmorbodian monetary system. Making change for 70 cents greedily would use first a 60 cent coin. Then, there is 10 cents left to make and the algorithm would use a 5 cent coin followed by another 5 cent coin, for a total of 3 coins. We saw above that this amount of change could be made with just 2 coins.

   (a) Prove that the greedy algorithm is optimal for any currency system in which the coins have values $b^0, b^1, b^2, \ldots, b^k$ where b and k are positive integers and $b > 1$. (*Hint:* Induction!)

(b) **OPTIONAL BONUS PROBLEM:** Can you further generalize the kinds of coinage systems for which the greedy algorithm issues the minimum number of coins? Try to state the most general sufficient condition on the coinage system that you can and prove it! *(Please submit your solution to this problem on a separate sheet.)*

(c) Describe an efficient dynamic programming algorithm for minimizing the number of coins used in *any* coinage system. That is, the algorithm should take as input a list of coin types (from smallest to largest value) and an amount of change to be made and should return the least possible number of coins that can be used. The algorithm should return ∞ if it is not possible to make up the requested amount of change with the given coin types. Rather than giving pseudo-code, simply give the following description of your algorithm:

   i. Describe what the dynamic programming table looks like and its dimensions.

   ii. Explain what each cell represents.

   iii. Describe the order in which the cells in the table are filled in and the rule used to fill in each cell. Be sure to explain how to fill in the "easy" cells at the beginning as well as the rule for filling in the other cells.

(d) Implement your **dynamic programming** algorithm (not the recursive algorithm!) in your favorite programming language (formally defined as the set comprising C, C++, Java, Python, Scheme, rex, and ML). Your code is likely to be roughly between 15 and 40 lines long, depending on the language that you use. Your implementation should take as input a list or array of coin types (from smallest to largest denominations) and the amount of change to be made and should print both the minimum number of coins required **AND** *the actual list of coins that make up the change.* These inputs can be given in the command line, passed directly into the function, given by the user in response to prompts from your program, and/or read in from a file. The output should indicate the total number of coins used followed by list of these coins. You should submit the following:

i. A printout of your code (make sure to include ample explanatory comments).

ii. A screenshot (cutting and pasting from the terminal window is fine or - on Unix machines- use the `script` command to record your interaction) of your code running on the following inputs:

   A. Coin set: $1, 5, 30, 40, 60$ and change to be made of 70 cents.

   B. Coin set: $1, 5, 10, 25$ and change to be made of 90 cents.

   C. Coin set $1, 5, 10, 40, 60$ and change to be made of 91 cents.

For example, here is screenshot of my Python program on two of these inputs:

```
>>> coinsDP([1, 5, 30, 40, 60], 70)
2 coins
40 cent coin
30 cent coin
>>> coinsDP([1, 5, 10, 40, 60], 91)
4 coins
40 cent coin
40 cent coin
10 cent coin
1 cent coin
```

2. [**20 Points**] **Hurts Car Rental!** Hurts Car Rental is experimenting with a new type of vehicle. These vehicles use a modular fuel pack which allows the vehicle to travel exactly 100 miles. (The fuel is believed to consist of a mixture of liquified Spam, Mountain Dew, and Cheetos, although the technology is proprietary and this is just speculation.) When the car needs to be refueled, the fuel pack is removed from the car and is replaced by a new one. Thus, it is possible that a fuel pack will be replaced before it is completely used up. Moreover, the driver gets no credit for the remaining fuel in the fuel pack and can only purchase a new fully charged 100 mile pack.

   A driver may wish to travel on a long freeway from one point to another. Since only designated service stations sell the fuel packs, the driver needs to plan carefully where to refuel. Moreover, each service station charges a different amount for a fuel pack. Some drivers also don't want to stop very often to refuel. Consequently, these vehicles have a dial on the dashboard called the α dial. By turning the dial, the driver sets a value of α between 0 and 1. By doing so, the driver stipulates that she wishes to minimize the quantity $\alpha S + (1 - \alpha)C$ where S is the number of fuel stops made and C is the total cost paid for the fuel packs. Notice that when α is set to 0, the objective becomes that of minimizing the total cost paid for the fuel packs. When α is set to 1, the objective becomes that of minimizing the number of stops. When α is set somewhere between 0 and 1, the objective is a linear combination of these.

Consider a highway represented by a line segment. Let $p_1, p_2, \ldots, p_n$ be n points on the line segment sorted from left to right where p_1 is the starting point, p_n is the destination point, and each of these points has a service station selling fuel packs. Let d_i be the distance from point p_1 to point p_i, for $1 \leq i \leq n$. Let c_i be the cost of a fuel pack at the station at point p_i.

(a) Professor I. Lai claims that the following greedy algorithm minimizes the total cost when $\alpha = 0$: Buy a fuel pack at p_1 (we have no choice there - we need fuel to depart on the trip). Travel to the furthest point which is at most 100 miles away. Purchase a fuel pack there. Now repeat this process of traveling as far as possible on the current fuel pack before purchasing a new fuel pack. Explain briefly why this approach does not guarantee an optimal solution when $\alpha = 0$.

(b) Assume that a given value of α has been established by the driver. Give pseudo-code for a recursive algorithm called `optimize(`$[p_1, p_2, p_3, \ldots, p_j]$`, `$p_k$`)` which returns the minimum value of $\alpha S + (1 - \alpha)C$ for a trip from p_1 to p_k with fuel stops permitted at any of the consecutive service stations between p_1 and p_j. You should assume that $j < k$. Moreover, we must always purchase a fuel pack at p_1 to start the trip and this counts as one fuel stop. (Notice that once you've written this function, we can call it with `optimize`$([p_1, \ldots, p_{n-1}], p_n)$ to get our desired solution!)

Your pseudo-code will need to use the 100 mile travel limit per fuel pack and will need to refer to the d_i values which give the distances from p_1 to p_i and the c_i values which specify the cost of a fuel pack at point p_i. Finally, if there is no possible way to get from p_1 to p_k due to the distances between the given fuel stations, the function should return the value ∞.

(c) Describe a dynamic programming algorithm for this problem. In particular:

i. Describe what the dynamic programming table looks like and its dimensions.

ii. Explain what each cell represents.

iii. Describe the order in which the cells in the table are filled in and the rule used to fill in each cell. Be sure to explain how to fill in the "easy" cells at the beginning as well as the rule for filling in the other cells.

(d) What is the running time of your dynamic programming algorithm? Explain briefly.

3. **[20 Points] Ski Optimization!** Your job at Snapple is pleasant but in the winter you've decided to become a ski bum. You've hooked up with the Triple Black Diamond Ski Resort. They'll let you ski all winter for free in exchange for helping their ski rental shop with an algorithm to assign skis to skiers.

Ideally, each skier should obtain a pair of skis whose height matches his or her own height exactly. Unfortunately, this is generally not possible. We define the *disparity* between a skier and his or her skis to be the absolute value of the difference between the height of the skier and the pair of skis. Our objective is to find an assignment of skis to skiers that minimizes the sum of the disparities. We'll solve some interesting subproblems along the way to finding an efficient algorithm for the ski assignment problem.

(a) First, let's assume that there are n skiers and n skis. The previous computer scientist at the ski resort, Professor R. U. Kitting, proposed the following algorithm for this case. Consider all possible assignments of the n skis to the n skiers. For each one, compute the sum of the disparities. Finally, select the assignment that minimizes the sum of the disparities. Explain why Professor Kitting was fired. How much time would this algorithm take on a computer that performs 1 billion operations per second if there were 50 skiers and 50 pairs of skis?

(b) The next computer scientist at the resort, Dr. Anna Litik, was an excellent skier and spent most of her time on the slopes. She did however make an interesting discovery one day on the chair lift. She observed that if we have a short person and a tall person, it would never be better to give the shorter person a taller pair of skis than were given to the tall person. Show that this is always true. (If you like, you may split this into cases and consider each case separately.)

(c) After Dr. Litik's brief stint at the ski resort, you were hired. Your first job is to find a greedy algorithm that minimizes the sum of the disparities assuming that there are n people and n pairs of skis. (*Hint:* First sort the people by increasing height and sort the skis by increasing height.) Argue carefully that your algorithm does indeed find an optimal solution. In other words, show that no better solution is possible. What is the time complexity of your algorithm?

(d) Now, your task is to design an efficient dynamic programming algorithm for the more general case that there are m skiers and n pairs of skis and $m \leq n$. Again, start by sorting the skiers and skis by increasing height. Let h_i denote the height of the i^{th} skier in sorted order and let s_j denote the height of the j^{th} pair of skis in sorted order. Let $A[i, j]$ be the optimal cost (sum of absolute differences of heights) for matching the first i skiers with skis from the set $\{1, 2, \ldots, j\}$. The solution we seek is then simply $A[m, n]$. Fill in the blanks below to define A recursively:

$$A[i,j] = \begin{cases} \underline{\hspace{3cm}} & \text{if } i = 0 \\ \underline{\hspace{3cm}} & \text{if } 1 \leq i \leq j \end{cases}$$

(e) Now describe a dynamic programming algorithm for this problem. Specifically:

i. Describe what the dynamic programming table looks like and its dimensions.

ii. Explain what each cell represents.

iii. Describe the order in which the cells in the table are filled in and the rule used to fill in each cell. Be sure to explain how to fill in the "easy" cells at the beginning as well as the rule for filling in the other cells.

(f) What is the running time of your program? Explain.

(g) Describe how your algorithm can be modified to allow you to find an actual optimal assignment (rather than just the cost) of skis to skiers. How does this affect the running time of your algorithm?

(h) Illustrate your algorithm by explicitly filling out the $A[i, j]$ table for the following sample data:

- Ski heights: $1, 2, 5, 7, 13, 21$.
- Skier heights: $3, 4, 7, 11, 18$.

4. **[20 Points] The Millisoft Party Problem!** You've decided to accept a job as senior algorithm designer at the well-known Millisoft Corporation. One day, the President of Millisoft, Gill Bates, comes to you with the following problem. "I'm throwing a company party," Gill says excitedly, "And I need your help! As you know, Millisoft has a hierarchical structure. You can think of it as a tree. The president, that's me, is at the root of the tree. Oh boy, I love being at the root!" You take a sip of your luke-warm diet coke (which Millisoft provides for free - what a perk!) and listen patiently as Gill continues. "Below the root are supervisors, below them are managers, below them are team leaders, etc., etc., until you get down to the leaves - the summer interns. Anyhow, to make the party fun, I thought it best that we don't invite an employee along with their immediate boss (their parent in the tree). Also, I've personally assigned every employee a real number (actually it's a double precision floating point, but nevermind that!) called their *coefficient of fun*. My objective is to invite employees so as to maximize the total sum of the coefficients of fun of all invited guests, while not inviting an employee with his or her immediate boss."

(a) The first algorithm that Gill thought up was to simply enumerate all possible subsets of his employees, throw out those subsets that include an employee and his or her boss, find the score for each remaining subset, and finally choose the best one. There are 1000 employees at Millisoft. Millisoft has also just purchased a Crayfish

YMP that can process one trillion (10^{12}) subsets per second. How long will it take to find the optimal solution using this brute force approach on the Crayfish?

(b) Describe in detail an efficient dynamic programming algorithm for this problem. (*Hint:* Normally, we build a dynamic programming table or array. In this case, the structure in which you do your dynamic programming will be a tree! There's nothing wrong with that!)

(c) What is the asymptotic running time of your algorithm? Explain.

(d) How can you modify your algorithm to find the optimal solution in which Gill gets invited to his own party?