CS 170 Efficient Algorithms and Intractable Problems
Spring 2016 Alessandro Chiesa and Umesh Vazirani

HW 6

Due February 29, 11:59pm

1. (15 pts.) Set Cover

In class we saw a greedy algorithm for finding an approximate solution to the set cover problem. In this exercise, you will show that the approximation ratio we found for that algorithm is tight. Prove that for any integer k greater than 1, there is an instance of the set cover problem such that:

i. There are $2^k - 2$ elements in the base set

ii. The optimal algorithm uses just two sets

iii. The greedy algorithm picks at least k sets

Note: "base set" just means the union of all the sets in the instance (sometimes also called the "universe"). Also, you need to prove that for each k greater than 1 there is some instance of the set cover problem that satisfies all three conditions at once.

2. (15 pts.) Amortized Analysis

Recall that Queues and Stacks are both data structures that support adding and removing elements, and that in a Queue items are removed in FIFO order (that is, the item removed is the first item to enter the Queue) and in a Stack items are removed in LIFO order (that is, the item removed is the most recent item added to the Stack). As it turns out, it is possible to implement a Queue using two Stacks according to the procedure outlined below (you should take a moment to verify for yourself that this correctly simulates a Queue).

```
INITIALIZATION
1. Initialize two empty Stacks, S1 and S2

ADD(X)
1. Push x onto S1

REMOVE()
1. If S1 is empty and S2 is empty:
2.       Return null
3. If S2 is empty:
4.       While S1 is not empty:
5.             Push S1.pop() onto S2
6. Return S2.pop()
```

In this question you will analyze the amortized running time of the above operations.

(a) Suppose that every time an element x is added to the Queue we put some money beside it. Any time x is pushed onto or popped off of one of the Stacks, we must pay one dollar. How much money do we need to put next to x to ensure that we always have enough money to pay for all pushes and pops of x (including the initial push onto S_1)? Prove your answer correct.

(b) Using part (a), show that the running time of a series of n add and n remove operations, starting from an empty Queue and taking place in any order, takes $O(n)$ time.

3. (20 pts.) Updating MSTs

You are given a graph $G = (V, E)$ with positive edge weights, and a minimum spanning tree $T = (V, E')$ with respect to these weights; you may assume G and T are given as adjacency lists. Now suppose the weight of a particular edge $e \in E$ is modified from $w(e)$ to a new value $\hat{w}(e)$. You wish to quickly update the minimum spanning tree T to reflect this change, without recomputing the entire tree from scratch. There are four cases. In each, give a linear time algorithm for updating the tree. For each, use the four part algorithm format. However, for some of the cases the main idea, proof and justification of running time may be quite brief. Also, you may either write pseudocode for each case separately, or write a single algorithm which includes all cases.

(a) $e \notin E'$ and $\hat{w}(e) > w(e)$

(b) $e \notin E'$ and $\hat{w}(e) < w(e)$

(c) $e \in E'$ and $\hat{w}(e) < w(e)$

(d) $e \in E'$ and $\hat{w}(e) > w(e)$

4. (15 pts.) Horn Implementation

In class, we saw a greedy algorithm to find a satisfying assignment for a Horn formula, but we did not actually see how to implement it efficiently. Show how to implement the stingy algorithm for Horn formula satisfiability in time linear in the length of the formula (the number of occurrences of literals in it). You should use the four part algorithm format, but your proof of correctness can just involve showing that your pseudocode is doing the same thing as the algorithm we saw in class.

5. (15 pts.) Ternary Huffman

Trimedia Disks Inc. has developed ternary hard disks. Each cell on a disk can now store values 0, 1, or 2 (instead of just 0 or 1). To take advantage of this new technology, provide a modified Huffman algorithm for compressing sequences of characters from an alphabet of size n, where the characters occur with known frequencies $f_1, f_2, \ldots, f_n$. Your algorithm should encode each character with a variable-length codeword over the values 0, 1, 2 such that no codeword is a prefix of another codeword and so as to obtain the maximum possible compression. Your proof of correctness should prove that your algorithm achieves the maximum possible compression.

6. (20 pts.) Finding Palindromes

A subsequence is *palindromic* if it is the same whether read left to right or right to left. For instance, the sequence

$$A, C, G, T, G, T, C, A, A, A, A, T, C, G$$

has many palindromic subsquences, including A, C, G, C, A and A, A, A, A (on the other hand, the subsequence A, C, T is *not* palindromic). Devise an algorithm that takes a sequence $A[1...n]$ and returns the length of the longest palindromic subsequence. Its running time should be $O(n^2)$.

Note: A subsequence does not have to be contiguous in the original sequence. For instance, A, G, G, A is a subsequence of the above sequence.

7. (TBA pts.) Programming Assignment: Finding Counterexamples to Greedy Algorithms

Attention: This problem is not due this week; you should turn it in with homework 7 on 3/7. Please do not submit it with this week's homework.

Consider the following solitaire game: you are given a list of *moves*, $x_1, x_2, \ldots, x_m$, and their *costs*, $y_1, y_2, \ldots, y_m$. Moves and their costs both take the form of positive integers. Such a list of moves and their costs is called an instance of the game. To play the game, you start with a positive integer n. On each turn of the game you have two choices: subtract 1 from n incurring a cost of 1 or pick some move x_i such that n is divisible by x_i and divide n by x_i incurring a cost of y_i. The goal is to get to 0 while incurring the minimum possible cost.

For instance, suppose that the set of moves is $4, 5, 7$ and the corresponding penalties are $1, 3, 4$. If n is 20 then the optimal strategy is as follows:

1. Divide 20 by 4 to get 5, incurring a cost of 1
2. Subtract 1 from 5 to get 4 incurring a cost of 1
3. Divide 4 by 4 to get 1, incurring a cost of 1
4. Subtract 1 from 1 to get 0, incurring a cost of 1

So the final cost is 4. One suboptimal strategy for this game is as follows:

1. Divide 20 by 5 to get 4, incurring a cost of 3
2. Divide 4 by 4 to get 1, incurring a cost of 1
3. Subtract 1 from 1 to get 0, incurring a cost of 1

which gives us a total cost of 5.

One possible greedy algorithm for this game is as follows: On each turn, simply divide the current number by the legal move x of cost y which maximizes x/y– the ratio of move to cost (and subtract 1 from the current number if it is not divisible by any move). If two legal moves have the same ratio of move to cost then the greedy algorithm will take the larger move. Unfortunately, this greedy algorithm does not always give the optimal strategy. In this problem you will find counterexamples that show the greedy algorithm is not optimal in different instances of this game.

Instructions

Like all written homework questions in this class, this assignment should be completed by you alone. You may discuss it with your friends and study partners, but all the work that you submit should be yours.

If you have not done so already, please register for an instructional account for CS 170. This can be done by following this link: https://acropolis.cs.berkeley.edu/ account/webacct.

On your instructional account, you will find a file called hw6/instance.txt that contains ten instances of this game (note: these instances are unique to your login– different people will have different instances). For each, you should find the smallest value of n such that the greedy algorithm fails to find the best strategy.

Additionally, two python scripts can be found on piazza: generate_instances.py and check_strategy.py. The first will take your login and generate the appropriate instances. The second will take a list of moves, a value of n and a strategy and determine if the strategy is valid. These scripts are provided only for your benefit and are not required to complete the project in any way. The script check_strategy in particular is provided only to make sure that you understand the rules of the game and to help you check if your code is generating legal strategies; it will *not* check that your strategy is optimal.

What to Submit

Your solution to this question should be submitted as part of your solutions to homework 7. Please do not submit it with this homework (homework 6). In the pdf that you submit to gradescope for homework 7, you should submit the following items:

- Your login
- The smallest value of n for which the greedy algorithm fails for each of the instances of the game corresponding to your login (which you can find either on your instructional account or by running the python script on piazza)
- An explanation of how you found the counterexamples, including a proof that your method finds the smallest counterexample for each instance. If you wrote a program to find the smallest counterexample for each instance then explain the main idea of you algorithm, provide concise pseudocode and prove that it is correct. If you used any clever tricks or optimizations to make your code more efficient then include a brief description (although you are not required to optimize your code in any way– the only requirement is that you find the smallest counterexample to each instance above). If you used some other method (besides writing a program) then justify that it is correct.
- Any code you wrote (include this after the rest of your write-up)

It is okay for the write-up to be handwritten, but your pdf should include any code you wrote (you can merge two pdfs using Adobe Acrobat, which you can download for free if you are a student at Berkeley).

Additionally, using the "submit" command on the instructional servers, you should submit your code and a text file containing the values of n you found. The text file should be called "solutions.txt" and should have one value of n per line, in the same order as in the instance file (with nothing else on each line). Specifically, in the directory that contains your code and the solutions.txt file, enter the command "submit hw6" and follow the instructions to submit the files containing your code (if you wrote your code on your own computer then you may want to use the scp command to copy it to the instructional servers).

Tips:

- To find instances where the greedy algorithm fails, you will probably need to be able to find the optimal strategy in all instances. What kind of algorithm can you use to do this?
- Most of the difficulty of this assignment is in coming up with the right approach. The code you write does not need to be especially long or complex to solve the problem.
- It may take some time for your code to find a counterexample, particularly for the last couple instances. But if it takes more than a couple of minutes, you probably need to rethink your approach.
- When searching for instances of where the greedy algorithm fails, it is useful to avoid redoing work.
- For reference, the optimal cost for the instance (3, 1), (2, 1) when n is 321 is 10, and the greedy algorithm gives 11. The smallest value of n for which the greedy algorithm fails on the instance $(211,2),(210,1)$ is 4740960.
- Start early. You have two weeks to do this problem, so you shouldn't try to do it all in the last day.