CSE 101 Homework 4

Winter 2015

This homework is due Friday February 20th *at the start of class*. Remember to justify your work even if the problem does not explicitly say so. Writing your solutions in LaTeXis recommend though not required.

Question 1 (Making Change, 30 points)**.** *For each of the following sets S consider the problem of given a positive integer n writing it as a sum of as few elements of S as possible. There is a natural greedy algorithm for this problem where given n we repeatedly use the largest element of S that we can without going over. So for example of $n = 27$ and $S = \{1, 5, 12\}$, the greedy algorithm would write n as $27 = 12 + 12 + 1 + 1 + 1$. For each of the sets S below, either prove that the greedy algorithm provides the optimal solution for all n or produce a counterexample.*

(a) $S = \{1, 2, 3, 4, 5, \ldots\} = \{$*all positive integers*$\}$ *[5 points]*

(b) $S = \{1, 4, 9, 16, 25, \ldots\} = \{$*all squares*$\}$ *[5 points]*

(c) $S = \{1, 2, 4, 8, 16, \ldots\} = \{$*all powers of* $2\}$ *[10 points]*

(d) $S = \{1, 2, 3, 5, 7, 11, 13, 17, \ldots\} = \{1\} \cup \{$*all prime numbers*$\}$ *[10 points]*

Solution 1.

(a) The greedy algorithm always works in the case that $S = \mathbb{N}$, since any possible amount of change c is in S, and the highest possible denomination that can be selected without going over is c, which the greedy algorithm will select first. Therefore, the algorithm will always make change with only one unit, which is clearly optimal.

(b) The greedy algorithm is not always optimal when S is the set of squares. Consider 41, which is the sum of 16 and 25. The greedy algorithm will first select 36, leaving 5; then 4, leaving 1; and finally 1. Therefore, the greedy algorithm chooses a solution of three elements, when a solution of size two exists.

Note: By Lagrange's Four Squares Theorem it is always possible to make change with this set using at most four elements of S. However, the greedy algorithm is not bad. It can be shown to make change for n using at most $O(\log\log(n))$ coins.

(c) The greedy algorithm always succeeds when S is the set of powers of two. To show this, we first characterize the solution that the greedy algorithm will select. Then, we show that the solution with these characteristics is unique. Finally, we show that if a solution does not satisfy these characteristics, then it is less optimal then the greedy solution.

Claim. *The greedy algorithm never selects more than one coin of each denomination.*

Proof. Suppose the greedy solution chose more than one coin of the same denomination d. Then it could also have chosen one coin of denomination $2d$ before choosing more than one of d. Thus, selecting more than one coin of denomination d means that the algorithm did not choose the greatest possible denomination on an earlier step, which contradicts the assumption that the algorithm was greedy. □

Claim. *There is a unique solution that makes change using coins of distinct sizes.*

Proof. The binary representation of any positive integer, wherein each bit indicates whether the power of two corresponding to its position should be included in the sum (once) with a "1" or "0", is unique. This is the same format in which the greedy algorithm will select a solution − it either selects one coin of denomination 2^k (for some k), or it selects no coins of that denomination. Therefore, this solution is unique. □

Claim. *Any solution which selects more than one of each denomination is sub-optimal.*

Proof. Suppose we select more than one coin (i.e., two or more) of denomination d. As before, we can exchange two coins of d for one coin of $2d$ (and $2d \in S$ since d is a power of two and S is the set of powers of two), thereby improving the solution. □

Therefore, the greedy algorithm selects the unique optimal solution.

(d) The smallest counter-example here is 122*. The greedy algorithm produces the solution* $122 = 113 + 7 + 2$*, while* $122 = 61 + 61$ *is a solution using fewer coins.*

Note: The Goldbach Conjecture would imply that change can be made for any number using at most 4 coins. Recent work by Harald Helfgott implies that unconditionally at most 4 coins are necessary. However, the greedy algorithm still does quite well on this problem. It is known that the greedy algorithm makes change for n using at most $O(\log\log(n))$ *coins. Furthermore, given Cramer's Conjecture on the size of gaps between primes, it would always use at most* $O(\log^*(n))$ *(the number of times you need to apply* log *to n until you get a number less than* 1*) coins.*

Question 2 (Minimizing Maximum Lateness, 35 points)**.** *Consider the following problem. You are given n different tasks to perform. The* i^{th} *task takes total time* t_i*. These tasks must be done sequentially, but can be done in any order (so for example if you do tasks 3 then 5 then 2, task 2 will finish after* $t_3 + t_5 + t_2$ *time has passed). The* i^{th} *task has a deadline* d_i*. If you finish task i at time* f_i*, we define the lateness of the task to be* $\max(f_i - d_i, 0)$*. You objective is to find an order to perform the tasks in that minimizes the maximum lateness.*

It turns out that this problem has a simple greedy algorithm. In particular, performing the tasks in order of increasing deadline (so the task with the earliest deadline is performed first, the next earliest deadline is done next and so on) will always produce an optimal solution. Prove that this is the case.

Hint: You may want to employ an exchange argument. In particular, if two adjacent tasks are performed out of order, show that by swapping them you are no worse off.

Solution 2. *Given a set of n tasks, if any two tasks i and j have deadlines* $d_i > d_j$ *and are performed consecutively in the order* i, j*, then by swapping the order in which they are performed, we can reduce the maximum lateness of both tasks,* l_i *and* l_j*, without affecting the contribution of the lateness of any other jobs to the global maximum. In particular, we show that this swap either improves the maximum lateness or leaves it the same. In either case, by performing a series of these swaps we can transform an arbitrary ordering into the deadline ordering without increasing the maximum lateness. Therefore, the deadline ordering must have the minimum possible maximum lateness.*

Let the current task ordering be a permutation π *of size n. Let* $\pi[k] = i$ *and* $\pi[k+1] = j$*, where* $d_i > d_j$*, and tasks i and j have end times* f_i *and* f_j *and lateness* l_i *and* l_j*. Let the new task ordering* π' *be such that* $\pi'[x] = \pi[x]$ *for* $x \notin \{k, k+1\}$*,* $\pi'[k] = j$*, and* $\pi'[k+1] = i$*, and tasks i and j have end times* f'_i *and* f'_j *and lateness* l'_i *and* l'_j*. That is,* π' *is* π *with tasks i and j swapped in time.*

Claim. $l_i \leq l_j$

Proof. This is always true because j finishes later than i, but has an earlier deadline. □

Lemma. $\max\{l_i, l_j\} = l_j$

Claim. $l'_i \leq l_j$

Proof. When i and j are swapped, the end time of the later task does not change, since the start time of the first task is only a function of the previous tasks which we do not rearrange, and the end time of the second task is the start time plus the sum of both times, which does not change with order. Therefore, $f'_i = f_j$, but $d_j < d_i$, so $l_j > l'_i$. □

Claim. $l'_j \leq l_j$

Proof. $f'_j \leq f_j$, since j begins before i in π'. Since j's deadline does not change, $l'_j \leq l_j$. □

Claim. *For $x \notin \{k, k+1\}$, $l_{\pi[x]} = l_{\pi'[x]}$.*

Proof. That is, the lateness of other tasks is not affected by swapping the order of adjacent tasks i and j. Obviously, any task before either tasks is not affected. For tasks after i and j, the start time of the first task after them is the same as it was before, that is, $S_{\pi[k+2]} = S_{\pi'[k+2]}$. This is because the start time of i in π and of j in π' is the sum of the times of all previous jobs, which do not change, and the tasks together take $t_i + t_j$ time, no matter what order they are in.

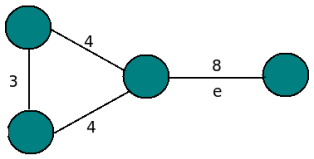
Therefore, any tasks which are not i and j start and finish at the same time in both π and π', so their lateness cannot be affected by swapping i and j. □

These facts are sufficient to show that the maximum lateness in π' is less than or equal to that in π. In π, $\max\{l_i, l_j\} = l_j$, which, since $l_j \geq l'_i$ and $l_j \geq l'_j$, it follows that $l_j \geq \max\{l'_i, l'_j\}$, and therefore the contribution of the maximum lateness of i and j to the overall lateness is smaller (or no worse) in π' than it is in π.

Question 3 (Properties of Minimum Spanning Trees, 35 points)**.** *Desgupta, Papadimitriou, Vazirani Problem 5.9 parts a, d, e, f, h, i, j. [5 points each]*

Solution 3.

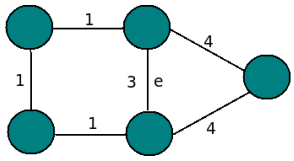
(a) ***Claim:*** *If graph G has more than $|V| - 1$ edges, and there is a unique heaviest edge, then this edge cannot be part of a minimum spanning tree.*
Answer: False. *Since this is not a complete graph, it is possible that this unique heaviest edge is the only edge across a cut of G. The following is a counterexample:*



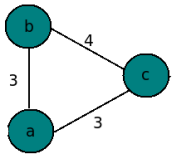
(d) ***Claim:*** *If the lightest edge in a graph is unique, then it must be part of every MST.*
Answer: True. *Let e be the lightest edge in a graph and assume there is some MST T' such that $e \notin T'$. Let $(S, V \setminus S)$ be a cut such that e crosses the cut. Since e is not in T', there must be some some other edge e' which crosses the cut and $e' \in T'$. Let T be the tree $T = T' \setminus \{e'\} \cup \{e\}$. Then $cost(T) = cost(T') - weight(e') + weight(e) < cost(T')$ since $weight(e) < weight(e')$. Then T' cannot be an MST.*

(e) ***Claim:*** *If e is part of some MST of G, then it must be a lightest edge across some cut of G.*
Answer: True. *Assume e is in an MST T, but there is no cut in G for which e is a lightest edge across the cut. Pick any cut $(S, V \setminus S)$ such that e crosses the cut, and let e' be another edge crossing the cut such that $weight(e') < weight(e)$. Then an MST T' using edge e' instead of e will be of lower weight than T since $weight(T') = weight(T) - weight(e) + weight(e') < weight(T)$. Therefore e but be a lightest edge across some cut of G.*

(f) ***Claim:*** *If G has a cycle with a unique lightest edge e, then e must be part of every MST.*
Answer: False. *Since the heaviest edge in any cycle cannot be part of any MST, and it is possible that the lightest edge in one cycle could be the heaviest edge in another, the lightest edge in one cycle may not necessarily be in any MST of G. The following is a counterexample:*



(h) ***Claim:*** *The shortest path between two nodes is necessarily part of some MST.*
Answer: False. *Imagine a graph of 3 nodes, a, b, and c, with edges $\{a, b\}$ of weight 3, $\{a, c\}$ of weight 3, and $\{b, c\}$ of weight 4. The shortest path from b to c is along the edge of weight 4, but the unique minimum spanning tree consists of the two 3-weight edges.*



(i) ***Claim:*** *Prims algorithm works correctly when there are negative edges.*
Answer: True. *The order in which edges are added in Prim's algorithm do not change, and the cut property holds for negative weight edges as well. Therefore the analysis does not change with negative weight edges.*

(j) ***Claim:*** *(For any $r > 0$, define an r-path to be a path whose edges all have weight $< r$.) If G contains an r-path from node s to t, then every MST of G must also contain an r-path from node s to node t.*
Answer: True. *Assume for the sake of a contradiction that G contains an r-path from s to t and there exists some MST T which does not. Then the unique $s - t$ path in T must contain some edge e with weight $weight(e) \geq r$. The removal of this edge leads to a cut with s on one side and t on the other. Since there is an r path in G, there must be some edge e' along this path which crosses the cut such that $weight(e') < r$. Then any MST T' containing the edge e' will have smaller weight than T, a contradiction to T being an MST.*

Question 4 (Extra credit, 1 point)**.** *Approximately how much time did you spend working on this homework?*