CIS 320 Problem Set 3

Due: Wednesday 10/27/21

Welcome to Problem Set 3. A few notes before you begin.

- You may collaborate with up to 4 other people. All work must be written up individually. You may collaborate, but you cannot copy. E.g. you can talk to your friends about ideas for a problem together, but you cannot copy their solution and just change some words around. **List your collaborators on the top of your submission**.
- Googling or looking up solutions in any way is not allowed.
- These problems are designed to challenge you. Start them early; if we've done our job well writing them, you will have to chew on them for a while before finding the solutions, and that means you will do best if you can sleep on your solutions rather than starting them the night before the due date.
- Cheating is not worth it! Your grade in this course does not define your worth as a person, and in 10 years you will not care about your bad grade on a homework assignment. But you will care if you are caught plagiarizing: plagiarizing has serious consequences, including the potential of expulsion. It is not worth it.
- You do not need to implement anything in code. In fact, please do not. Pseudocode or a clear English explanation of your algorithm are both acceptable: in some cases pseudocode may be clearer than plain English, and in others the plain English might be better.
- All analysis must be mathematically rigorous.
- Remember, we can't evaluate your work if we can't understand it. Communicating mathematical and/or complex ideas is an important skill in computer science; treat your problem sets as practice.
- Have fun!!

Problem 1 (Billywig Woe)**.** Your dorm has become infested with Billywigs! Several of them have nested in holes along a corridor that is n feet long. You have Billywig traps, but they are very expensive, so you don't want to use more of them than you need to. A Billywig trap will catch any Billywig that has nested within 5 feet of it, in either direction. Give a linear time algorithm to place the Billywig traps that uses as few traps as possible. You can assume that the hallway is an array of length n, with each entry of the array corresponding to a 1 foot segment of hallway. An array element is set to 1 if a Billywig has nested in that segment, and is set to 0 otherwise. You need to choose the minimal number of array elements in which to set the traps so that every Billywig is within 5 entries of some trap.

Problem 2 (A Poet (But You Don't Know It))**.** You and your friend Steve run a successful business writing best-selling books of new age poetry. Here is how it works: You stay awake for 72 hours until you start hallucinating, at which point you generate a sequence of n words seemingly at random. You call this your "wordstream". Then you figure out how to chop up your wordstream into subsequences that will be your final collection of poems, which you will publish and earn royalties from. For example, if you decide to chop your wordstream at positions t_1 and t_2, you will end up with a collection of 3 poems: the sequence of words from 1 to t_1, the sequence from t_1 to t_2, and the sequence from t_2 to n. You never know ahead of time how many poems you will chop up your wordstream into. In fact, you don't really have an ear for poetry at all. This is where Steve comes in. After *he* stays up for 72 hours, he is able to hear an arbitrary sequence s of words, and then report back the poetic quality pq_s of that sequence. Querying Steve on a single string s takes constant time independently of the length of the string, since Steve experiences time strangely after staying up for so long. Suppose you have partitioned your wordstream into k poems s_i — then the value of your collection of poetry is simply the sum of the poetic qualities of each of the poems: $\sum_{i=1}^{k} pq_{s_i}$. Given a word-stream of length n, give as efficient an algorithm as you can for partitioning it into the collection of poems s_i of maximum value.

Problem 3 (Fluffy Fashion)**.** You have n Pokemon you care about. Why? Because each and every one of them is fluffy. But you want even more fluffiness! You are a hairdresser and have an integer budget $b \geq 0$ of creative energy. A Pokemon scholar tells you a secret. For each Pokemon i, if you invest $a_i \geq 0$ integer units of creative energy into doing Pokemon i's haircut then you get back $a_i + p_i$ units of creative energy (where $p_i \geq 0$ is integer) once the haircut is done! (Or, you can invest none and get no increase in creative energy; also, note that you cannot do any Pokemon's haircut more than once.) You must decide today, for each Pokemon i, whether or not to give them a haircut; you may not exceed your budget b or postpone your decision regarding any Pokemon until later. The unspent part of the energy stays with you, and tomorrow you receive dividends on the Pokemon you chose, as specified above — so that your new energy level will be some $c \geq b$. Design an $O(n \cdot \sum_{i=1}^{n} p_i)$ algorithm that outputs the maximum amount c of creative energy that you can guarantee yourself for tomorrow. *Note: Why are we asking you for this runtime, instead of the $O(nb)$ runtime which (check it yourself) easily follows from what was covered in lecture? It is because you believe that you have a super huge supply of creative energy to begin with and so runtime proportional to b would be prohibitive; and at the same time, you suspect that each Pokemon's net return on fluffiness p_i is pretty small, making the linear dependence on $\sum_i p_i$ rather desirable.*

Problem 4 (Divide and ... Greedy)**.** Let us have a fresh look at grade-school division. What do you do? To divide a by b, you:

1. Consider $b \cdot 10^{k_1}$ for the largest k_1 such that $b \cdot 10^{k_1} \leq a$. Then the lead digit of the answer is the maximum $1 \leq c_1 \leq 9$ such that $c_1 \cdot (b \cdot 10^{k_1}) \leq a$.

2. Subtract off $c_1 \cdot (b \cdot 10^{k_1})$ from a, and divide the remaining number by b recursively using Step 1, thus determining the remaining digits of the answer.

Our use of the words "largest" and "maximum" in the description of Step 1 above should indicate to you that this is a greedy algorithm! Why is this division algorithm so widely taught — surely it's not just simple but also optimal from some standpoint? We will now figure this out in even greater generality!

Part (a) Archie is battling a monster in her favorite computer game. The monster has health of a units (a is a positive integer). Archie has a range of n weapons that deal the following amounts of damage: $1, p_1, p_1p_2, \ldots, \prod_{i=1}^{n-1} p_i$, where $p_1, \ldots, p_{n-1}$ are all positive integers greater than 1. When the monster has

health x and is attacked using weapon of damage amount y, the monster's health becomes $x - y$. The monster is defeated when the health is at 0. Weapons can be reused. The game rules are tricky: at no point is Archie allowed to fire at the monster with a weapon that deals more damage than the monster's current health.

Archie wants to finish the game off as soon as she can, and follows a greedy strategy: At each step, she selects a weapon with the maximum available firepower among all weapons that she can currently attack the monster with. Prove that using this strategy, she defeats the monster with the least number of weapon uses.

Part (b) Does Archie always defeat the monster in the optimal number of steps using this algorithm if her weapons are allowed to have arbitrary distinct integer damage levels, with the only restriction being that one of the weapons has damage level 1? If so, prove it. If not, give a counterexample (a specific health level for the monster, a specific set of weapons and their damage levels, and a justification.)

Part (c) Show how to instantiate the grade-school division algorithm as a special case of part (a). (What corresponds to the monster's health? What corresponds to the weapons' damage amounts?) Given this instantiation, briefly state the sense in which grade-school division is optimal.

Problem 5 (More Marvelous Matchings?)**.** Having learned about the greedy method, you are now eager to apply it to earlier (not-so-greedy-looking) parts of this course to potentially come up with an original algorithm. You recall the maximum unweighted bipartite matching problem: Given a bipartite graph $G = ((L, R), E)$ with unit edge weights, one needs to find a matching of the largest possible cardinality.

You have come up with the following algorithm. Given any matching M, and a simple path $P = \{e_1, \ldots, e_{len(P)}\}$ (recall that in a simple path, vertices and edges never repeat; here, $e_1, \ldots, e_{len(P)}$ is the list of edges along the path from start to end), we refer to P as *M-nice* if two conditions hold. First, the edges on this path alternate between edges that are not in M and edges that are in M in the following precise sense: $e_1 \notin M, e_2 \in M, e_3 \notin M, \ldots, e_{len(P)-1} \in M, e_{len(P)} \notin M$. Note that a path satisfying this alternating condition must have an *odd* number of edges. Second, the first and last vertex along the path P are both unmatched in M.

Your algorithm is now simple to describe: Start with an empty matching $M_0 = \emptyset$; and for $i \geq 0$, while there exists an M_i-nice path $P_i = \{e^i_1, \ldots, e^i_{len(P_i)}\}$, obtain the next matching M_{i+1} by removing from it the edges $e^i_2, e^i_4, \ldots, e^i_{len(P_i)-1}$ and instead adding in the edges $e^i_1, e^i_3, \ldots, e^i_{len(P_i)}$. If there is no M_i-nice path, output M_i as the answer and terminate.

1. Briefly state: Can you justify calling this algorithm greedy? That is, is there some sense in which each step to brings us closer to optimality?

2. Either prove that this algorithm is correct (i.e. always returns a max-cardinality matching) or give a counterexample.