Homework 2: Graph Search

Be sure to read the grading guidelines and style guidelines. Especially to see the suggested format for describing algorithms.

We sometimes describe how long are justifications or proofs are. These lengths are intended to help you estimate how much detail we're expecting; your proofs are allowed to be longer.

You are allowed (and encouraged!!) to collaborate with each other. Brainstorming is much easier to do in a group than alone! But you must follow the collaboration policy (which includes needing to write your submission on your own).

You will submit to gradescope; we will have a different box for each problem, so please give yourself extra time to submit.

1. Function Connoisseur [10 points]

Put these functions in increasing order. That is, if f comes before g in the list, it must be the case that $f(n)$ is $\mathcal{O}(g(n))$. All logs are base 2.

Note: We're working on a way to autograde this problem; we'll update you when we have details of who it will be submitted.

- $f_1(n) = 5n + n(\log(\log(n)))^2$
- $f_2(n) = \log(n^{1000})$
- $f_3(n) = 2^{0.1n}$
- $f_4(n) = n^2$
- $f_5(n) = \dfrac{n^{1000}}{1000}$
- $f_6(n) = n^{2\log(n)} - n^2$
- $f_7(n) = \log((n!)^3)$
- $f_8(n) = n^{\log(\log(\log(n)))-999}$
- $f_9(n) = n\log(n)$

2. Coupon Connoisseur [25 points]

You are in Seattle and are trying to fly to NYC. Being the budget-conscious person you are, you don't mind layovers, and you want to find the cheapest series of flights to take to get to NYC. No matter how many layovers it takes. Due to recent technology issues at the airline, you had a previous series of flights cancelled. You were compensated for those flights with k coupons, each of which entitles you to a flight free of charge. You are prepared to use as many of them as it takes to make the trip as cheap as possible.

You are given a list of flights. Each flight has a departure city, an arrival city, and an integer cost in dollars. You can take flights in any order, so long as you are at the departure city of the flight. You can only move between cities by taking flights.

Describe an algorithm to find the minimum dollar cost it takes to fly from Seattle to NYC, where you have coupons allowing you to fly for free k times. If it is impossible to fly from Seattle to NYC, return infinity.

You must call Dijkstra's algorithm exactly once in your algorithm.

(a) Give a 1-2 sentence summary of the main idea of your algorithm (this helps us when grading).

(b) Give pseudocode (and/or English) to solve this problem. You will probably want to construct at least one graph, you can assume creating vertices and edges takes $O(1)$ time each, and you can give an English/mathematical description of the vertices and edges (don't worry about code constructs for creating it).

(c) Give a 1-2 sentence of intuition for why your algorithm will work (this helps us when grading and should help you when proving, too!).

(d) Prove your algorithm is correct. You may use as a fact that any algorithm from this list works (as well as any analysis of any algorithm from class).

(e) Give the running time of your algorithm and briefly justify (2-3 sentences). For analyzing the running time, assume that there are n cities, m flights, and k coupons. Give a big-$\mathcal{O}$ bound in terms of n, m, k.

3. Activity Connoisseur [25 points]

After arriving in NYC you realize you are there for only d days and every day you want to do exactly one of a activities. In order to fully enjoy all the activities, you've set some guidelines for yourself regarding the following day after some activity (i.e. you won't follow an art gallery day with another art gallery day day, a Broadway day is necessarily followed by a museum day, etc). You've written these r restrictions as a list of pairs of activities (a_1, a_2) where a_2 is an activity that can be done the day after a_1 is done. (Note this **doesn't** necessarily mean you can perform activity a_1 after a_2. Order matters!)

You know that during your trip to NYC you will drink do exactly d activities, and your trip necessarily begins with a visit to the Statue of Liberty and ends with watching the National Chess-Boxing Championship at Radio City Hall. You wish to know how many different activity itineraries you could create. Specifically:

- The first activity must be Statue of Liberty, the last Chess-Boxing, and you must do **exactly** d activities (including the already-prescribed first and last activities) during the trip and exactly 1 activity each day.
- You may repeat activities, but you must not repeat the same activity two days in a row.
- Two itineraries are different if there is at least one i, such that the i^{th} activity you do is different (notice order does matter here! Changing the order of activities gives two different itineraries).

Design an $\mathcal{O}\left((a+r)^d\right)$ algorithm to **count the number** of possible itineraries sessions. You only need to count the number, you don't need to have a full list of itineraries.

(a) Give 1-2 sentences of intuition for your algorithm (this helps us when grading).

(b) Give pseudocode (and/or English) to solve this problem. You will probably want to construct at least one graph, you can assume creating vertices and edges takes $O(1)$ time each, and you can give an English/mathematical description of the vertices and edges (don't worry about code constructs for creating it).

(c) Explain why your algorithm is correct. This does not need to be a formal proof, but should clearly describe why the thing you find in the graph is the count of valid itineraries.

(d) State the big-$\mathcal{O}$ running time and justify it with 2-3 sentences. You may assume adding an edge to a graph takes constant time.

Hint: There are multiple different approaches here; we are imagining one that recursively walks through the graph, but other (clever) options are out there.

4. Satisfaction Connoisseur [25 points]

Recall (from 311) an expression in propositional logic notation is in "Conjunctive Normal Form (CNF)" if it is

- Formed of subexpressions, where each subexpression is 'AND'ed with the next.
- Each subexpression (called a clause) is the OR of literals.

Where a literal is either a variable or the negation of a variable.

For example,

$$(p \vee q) \wedge (\neg p \vee \neg r) \wedge (\neg r \vee p)$$

is in conjunctive normal form.

The expression above is in 2-CNF form, because every clause contains exactly two literals.

In this problem you will design a **graph** algorithm to find a variable setting that causes the expression to evaluate to true (or determine none exists).[1] We'll tell you what the vertex set will be; your job will be to figure out the edges and then how to finish up the algorithm.

(a) Consider the clause $(p \vee q)$. There are two implications that are each equivalent to that clause. What are they?

(b) Let V be a set of vertices. V will have one vertex for each possible literal (but if a literal appears more than once you only have one vertex for it, not one for each copy). What set of edges will you add to the graph? **Hint:** Use part (a)!

(c) Suppose that you have two literals x, y such that there is a path from x to y in the graph you describe. What can you conclude about x and y in a setting of the variables where the full expression evaluates to true? Prove your assertion.

(d) What can you say about two literals in the same strongly-connected component? Prove your assertion.

(e) Describe an algorithm that given a 2-CNF expression, finds a setting of the variables that causes the expression to be true (if one exists) or correctly returns that there isn't such a setting. You should think about how the prior parts can be key pieces in your algorithm.

- Finds a setting of the variables that makes the 2-CNF expression true, if there is one.
- returns "no solution" if there is not one.

(f) Prove your algorithm correct. You may reference the results of previous parts without reproving them.

(g) What is the running time of your algorithm? Briefly justify (2-3 sentences). Let m be the number of clauses in the expression and n the number of variables.

[1] That's right, a graph algorithm! Even though it really doesn't look like a graph problem on the surface.