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 Ordering [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 $\mathcal{O}(g(n))$. Additionally, if there are any pairs such that $f(n)$ is $\Theta(g(n))$, mark those pairs. All logs are base 2.

You will complete this problem directly on Gradescope instead of submitting a PDF. Like the other Gradescope submissions for this homework assignment, this will be open on Saturday October 5.

(a) $n \log(n)$

(b) $(n + 42100)^{0.421}$

(c) $\frac{1}{421} \log((\log(\log(n)))^{421})$

(d) $421 \cdot \log(\log(\log(n)))$

(e) $2^{(n^4+21)}$

(f) $\log(n)$

(g) $2^{\log(\log(\log(n)))}$

(h) 1.1^n

(i) $\log(n)^{\log(\log(n))}$

(j) $n^{\log(\sqrt{n})}$

2. Many degrees of Kevin Bacon [25 points]

In Mathematics and theoretical Computer Science, people proudly mention their Erdö s number - how far they are away in the co-authorship graph from Paul Erdös - who was famous for the wide range of his collaborations. (He died in 1996 at age 83, but some of the more than 1500 papers on which he is a co-author have publication dates as recent as 2015.)

Similarly, starting in the 1990's, people who noticed that Kevin Bacon was showing up in a lot of movies introduced the Six Degrees of Kevin Bacon game in which people would try to find a shortest path of co-appearances linking some arbitrary actor to Kevin Bacon. (There are still websites that can give you the answers.) In both of these situations there is a graph of people and edges between them if they are socially connected in some way. Of course a single short path between s and t does not necessarily describe a strong connection between them; it could be merely coincidental. However, if there are a lot of ways in which two people are connected by a shortest path then that suggests a much stronger connection.

Suppose that the task instead is to compute the *number* of shortest paths (counting only the number of hops) between s and t in an unweighted graph. It turns out that this can be done efficiently. Suppose that you are given a undirected graph $G = (V, E)$ and two nodes s and t in G. Give an algorithm that computes the number of shortest paths between

s and t in G in time $O(n+m)$ where n is the number of vertices and m is the number of edges in G. (The algorithm will not be able to list all such paths since there may be many more of them than the time bound.)

As with all problems, unless you are explicitly told otherwise, you should produce pseudocode, prove correctness, and justify the running time bound of your algorithm.

3. What Kind of Bear are You? [25 points]

You are at your cabin in the deep woods and have taken many pictures of bears – some of them are grizzly bears, some of them are brown bears, and some of them are black bears. You have no problem telling which ones are black bears, so you can easily tag all those photos as black bears, put them in a Black Bear folder, and be done with them, but you absolutely cannot remember what allows you to tell whether a photo is of a grizzly bear or a brown bear. Being in the deep woods, you also don't have cell service or internet and so can't run your favorite deep net image classification algorithms.

Having dealt with all the black bear photos already, you can look at two of the other bear photos and say "those are the same species" or "those are different species." You do a bunch of such comparisons and write them all down. You then start with one of the photos and say "let's call this species, species A" and put it in folder A. Your goal is to perfectly place every animal into a folder A ("all of these are species A") or folder B ("all of these are species B") where species B is the other remaining species (though you aren't sure whether A means grizzly bear or brown bear).

Given your data, you need to respond with one of three options:

- **Inconsistent:** you have analyzed the pictures in such a way that you can't possibly classify them all into species A and B correctly

  For example you said photo 1 and 2 were the same species, photo 2 and 3 were different species, and photos 1 and 3 were the same species.

- **Underspecified:** your answers aren't inconsistent, but there is at least one picture that (from the data you've collected so far) could validly be either species A or species B.

- **Exact answer:** your answers aren't inconsistent, and every photo can be only one of species A or B.

Describe an algorithm to return which of those three cases the input matches, and if you're in the exact answer case to store the species labels for each photo.

Sample Input I: There are 4 images.
Image 1 and 2 are different species
Image 3 and 4 are the same species
Image 1 is "species A"

Correct Output: Underspecified

Sample Input II: There are 4 images.
Image 1 and 2 are different species
Image 1 and 4 are the same species
Image 3 and 4 are different species
Image 1 and 3 are different species
Image 1 is "species A"

Correct Output: Exact Answer

Sample Input III: There are 4 images.
Image 1 and 2 are different species
Image 1 and 4 are the same species
Image 3 and 4 are different species
Image 1 and 3 are the same species
Image 1 is "species A"

Correct Output: Inconsistent

(a) 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 $\mathcal{O}(1)$ time each, and you can give an English/mathematical description of the vertices and edges (don't worry about code constructs for creating it). Any algorithm from class you use as a black box without modification does not need to be written out in full but you need to specify exactly how to modify it if you make changes to it.

(b) In proving that your algorithm is correct you may use known properties of any analysis of any algorithm from class as well as any of the data structures and their properties from 332.

(c) For analyzing the running time of your algorithm, assume that you have p photos that are either grizzly or brown bears, s pairs identified as "the same" and d pairs identified as "different." Give a big-$\mathcal{O}$ bound in terms of p, s, d.

4. A Package Deal [25 points]

You are responsible for configuring a custom software suite for a client. The client has included specific functionalities that they require to be present, which you can enable and support by installing specific packages. However, due to compatibility issues and dependencies, not all packages can be installed together. Your objective is to select an appropriate set of packages to install, ensuring that required functionalities are provided, all while respecting compatibility and dependency constraints.

To assist you in finding such a good configuration, you are provided a list of constraints. Each constraint involves an ordered pair (a, b) and a label that describes the relationship between the two packages. For each pair (a, b), the possible labels are:

- **At least one:** This means that at least one of a or b is required to be installed in order to satisfy a certain functionality. It is acceptable to install both packages if necessary.
- **Dependent:** Package a is dependent on package b (note the ordering here!). That is, including package a in the installation also requires that package b is installed.
- **Incompatible:** Package a and b cannot be selected together in the final set due to incompatibility issues.

As a student in CSE 421, you find that modeling this problem as a *directed* graph might be quite useful. You conclude that you need two vertices for each package – one indicating that the configuration includes the package and another indicating that the configuration does not include the package.

(a) For each of the three kinds of constraints, describe which edges must be added for that constraint type.
Hint: there may be more than one edge added per constraint!

(b) Suppose that there is a path from a vertex x to another vertex y in this graph. What does this tell you about the packages mentioned at x and at y in any full configuration meeting these constraints?

(c) Let C be a strongly connected component in this graph. Prove or disprove: Any good configuration is either consistent with all vertices in C or none of them.

(d) Describe an algorithm that, given a list of pairwise constraints as indicated above, finds a set of packages that can fulfill all required functionalities without causing dependency or incompatibility issues, if one exists. If one does not exist, your algorithm should return that there isn't such a set. You should think about how the prior parts can be key pieces to your algorithm.

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

(f) What is the running time of your algorithm? Briefly justify (2-3 sentences). Let n be the number of packages and p be the number of pairs.

5. Efficient Product Testing [Extra Credit]

You are not required to submit attempts at extra credit problems (they do not count toward the dropped/counted problems at the end of the quarter). They will have much *smaller effects on grades than the main problems, so we do not recommend attempting them until you've done all the other problems on the assignment. At the end of the quarter, after determining grade breaks, we will add in extra credit points.*

You are working with a consumer testing company doing some stress-testing on various models of a consumer product to determine the height from which they can be dropped and still not break.

The setup for this experiment, for a particular manufacturer's model, is as follows: You have a ladder with n rungs, and you want to find the highest rung from which you can drop a particular model and not have it break. We call it the *safety-height* for this model.

If you wanted to reduce the number of trials to a minimum, it might be natural to try binary search: drop the model from the middle rung, see if it breaks, and then recursively try from rung $n/4$ or $3n/4$ depending on the outcome. This would require only $\log n$ drops but has the major drawback that your testing company has to pay for each copy of the model in order to continue to perform the tests after each one breaks.

If your primary goal were to keep the number of copies of the model you need to a minimum, you could instead simply start at the 1st rung, then the 2nd, 3rd, and so on, until the model breaks. This only requires 1 copy but could take up to n drops, each of which is tedious to perform and record.

The general question here is what the optimal trade-off is between the best choices for the # of drops would be to identify the safety-height of a model given a fixed # of copies of the model. For this question, we'll just stick with 2 copies of the model.

(a) Describe the best strategy that you can think of that achieves $o(n)$ drops given a budget for 2 copies of the model.

(b) Try to figure out the exact number of drops your solution requires for $n = 21$ and $n = 22$.

(c) What is the asymptotic rate of growth of the worst-case number of drops for your solution from part (a) as a function of n?