CS 170 Homework 3

Due **2/12/2024, at 10:00 pm (grace period until 11:59pm)**

1 Study Group

List the names and SIDs of the members in your study group. If you have no collaborators, you must explicitly write "none".

2 Ice-Cream Loving PNPenguins

A group of PNPenguins are planning to run an ice-cream stand together for d consecutive days over the summer, which lasts $m > d$ days. Due to varying commitments, the group has agreed to staff a certain number of PNPenguins to sell ice-cream on each day of their operation period. Their goal is to find a starting date that can maximize their profit. To do this, they asked chatPenguin[1] to help them generate a list of predictions that contains the number of ice-creams each staffed PNPenguin can sell on that summer day. Their goal is to find a starting date that can maximize their profit.

Formally, we define the following:

- p_i represents the number of PNPenguins on duty to sell ice-cream on day i after their starting date ($1 \leq i \leq d$).
- a_j represents the amount of ice-cream that chatPenguin predicts each PNPenguin (on duty) to sell on day j ($1 \leq j \leq m$).

Design an $O(m \log m)$ algorithm to find the maximum number of ice-creams the group of PNPenguins can sell, as well as the starting date that allows them to sell that maximum amount. **Please provide a 3-part solution.**

[1] State-of-the-art LLM designed specifically for making ice-cream selling predictions.

3 Counting k-inversions

A *k-inversion* in a bitstring x is when a 1 in the bitstring appears k indices before a 0; that is, when $x_i = 1$ and $x_{i+k} = 0$, for some i. For example, the string 010010 has two 1-inversions (starting at the second and fifth bits), one 2-inversion (starting at the second bit), and one 4-inversion (starting at the second bit).

Devise a $O(n \log n)$ algorithm which, given a bitstring x of length n, counts all the k-inversions, for each k from 1 to $n - 1$. You can assume arithmetic on real numbers can be done in constant time. **Please give a 3-part solution; for the proof of correctness, only a brief justification is needed.**

4 Distant Descendants

You are given a tree $T = (V, E)$ with a designated root node r and a positive integer K. For each vertex v, let $d[v]$ be the number of descendants of v that are a distance of at least K from v. Describe an $O(|V|)$ algorithm to output $d[v]$ for every v. **Please give a 3-part solution; for the proof of correctness, only a brief justification is needed.**

Hint 1: write an equation to compute $d[v]$ given the d-values of v's children (and potentially another value).

Hint 2: to implement what you derived in hint 1 for all vertices v, we recommend using a graph traversal algorithm from lecture and keeping track of a running list of ancestors.

5 Where's the Graph?

Each of the following problems can be solved with techniques taught in lecture. Construct a simple directed graph, write an algorithm for each problem by black-boxing algorithms taught in lecture, and analyze its runtime. You do not need to provide proofs of correctness.

(a) The CS 170 course staff is helping build a roadway system for PNPenguin's hometown in Antarctica. Since we have skill issues, we are only able to build the system using one-way roads between igloo homes. Before we begin construction, PNPenguin wants to evaluate the reliability of our design. To do this, they want to determine the number of *reliable* igloos; an igloo is reliable if you are able to leave the igloo along some road and then return to it along a path of other roads. However, PNPenguin isn't very good at algorithms, and they need your help.

Given our proposed roadway layout, design an efficient algorithm that determines the number of reliable igloos.

(b) There are x different species of Pokemon, all descended from the original Mew. Also, all species have at most 1 parent. For any species of Pokemon, Professor Juniper knows all of the species *directly* descended from it. She wants to write a program that answers queries about these Pokemon. The program would have two inputs: a and b, which represent two different species of Pokemon. Her program would then output one of three options in constant time (the time complexity cannot rely on x):

(1) a is descended from b.

(2) b is descended from a.

(3) a and b share a common ancestor, but neither are descended from each other.

Unfortunately, Professor Juniper's laptop is very old and its SSD drive only has enough space to store up to $O(x)$ pieces of data for the program. Give an algorithm that Professor Juniper's program could use to solve the problem above given the constraints.

Hint: Professor Juniper can run some algorithm on her data before all of her queries and store the outputs of the algorithm for her program; time taken for this precomputation is not considered in the query run time.

(c) Bob has n different boxes. He wants to send the famous "Blue Roses' Unicorn" figurine from his glass menagerie to his crush. To protect it, he will put it in a sequence of boxes. Each box has a weight w and size s; with advances in technology, some boxes have negative weight. A box a inside a box b cannot be more than 15% smaller than the size of box b; otherwise, the box will move, and the precious figurine will shatter. The figurine needs to be placed in the smallest box x of Bob's box collection.

Bob (and Bob's computer) can ask his digital home assistant Falexa to give him a list of all boxes less than 15% smaller than a given box c (i.e. all boxes that have size between 0.85 to 1 times that of c). Bob will need to pay postage for each unit of weight (for negative weights, the post office will pay Bob!). Find an algorithm that will find the lightest sequence of boxes that can fit in each other in linear time (with respect to the number of edges in the graph).

Hint: How can we create a graph knowing that no larger box can fit into a smaller box, and what property does this graph have?

6 [Coding] DFS & SCCs

For this week's homework, you'll implement implement DFS and the SCC-finding algorithm covered in lecture. There are two ways that you can access the notebook and complete the problems:

1. **On Datahub**: click here and navigate to the `hw03` folder.
2. **On Local Machine**: `git clone` (or if you already cloned it, `git pull`) from the coding homework repo,

   `https://github.com/Berkeley-CS170/cs170-sp24-coding`

   and navigate to the `hw03` folder. Refer to the `README.md` for local setup instructions.

Notes:

- *Submission Instructions:* Please download your completed submission `.zip` file and submit it to the Gradescope assignment titled "Homework 3 Coding Portion".
- *Getting Help:* Conceptual questions are always welcome on Edstem and office hours; *note that support for debugging help during OH will be limited.* If you need debugging help first try asking on the public Edstem threads. To ensure others can help you, make sure to:

  1. Describe the steps you've taken to debug the issue prior to posting on Ed.
  2. Describe the specific error you're running into.
  3. Include a few small but nontrivial test cases, alongside both the output you expected to receive and your function's actual output.

  If staff tells you to make a private Ed post, make sure to include *all of the above items* plus your full function implementation. If you don't provide them, we will ask you to provide them.
- *Academic Honesty Guideline:* We realize that code for some of the algorithms we ask you to implement may be readily available online, but we strongly encourage you to not directly copy code from these sources. Instead, try to refer to the resources mentioned in the notebook and come up with code yourself. That being said, we **do acknowledge** that there may not be many different ways to code up particular algorithms and that your solution may be similar to other solutions available online.