CS 170 Homework 4

Due **2/20/2024, 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 Arbitrage

Shortest-path algorithms can also be applied to currency trading. Suppose we have n currencies $C = \{c_1, c_2, \ldots, c_n\}$: e.g., dollars, Euros, bitcoins, dogecoins, etc. For any pair of currencies c_i, c_j, there is an exchange rate $r_{i,j}$: you can buy $r_{i,j}$ units of currency c_j at the price of one unit of currency c_i. Assume that $r_{i,i} = 1$ and $r_{i,j} \geq 0$ for all i, j.

The Foreign Exchange Market Organization (FEMO) has hired Oski, a CS170 alumnus, to make sure that it is not possible to generate a profit through a cycle of exchanges; that is, for any currency $i \in C$, it is not possible to start with one unit of currency i, perform a series of exchanges, and end with more than one unit of currency i. (That is called *arbitrage*.)

More precisely, arbitrage is possible when there is a sequence of currencies $c_{i_1}, \ldots, c_{i_k}$ such that $r_{i_1,i_2} \cdot r_{i_2,i_3} \cdot \cdots \cdot r_{i_{k-1},i_k} \cdot r_{i_k,i_1} > 1$. This means that by starting with one unit of currency c_{i_1} and then successively converting it to currencies $c_{i_2}, c_{i_3}, \ldots, c_{i_k}$ and finally back to c_{i_1}, you would end up with more than one unit of currency c_{i_1}. Such anomalies last only a fraction of a minute on the currency exchange, but they provide an opportunity for profit.

We say that a set of exchange rates is arbitrage-free when there is no such sequence, i.e. it is not possible to profit by a series of exchanges.

(a) Give an efficient algorithm for the following problem: given a set of exchange rates $(r_{i,j})_{i,j \in n}$ which is *arbitrage-free*, and two specific currencies a, b, find the most profitable sequence of currency exchanges for converting currency a into currency b. That is, if you have a fixed amount of currency a, output a sequence of exchanges that gets you the maximum amount of currency b.

Hint 1: represent the currencies and rates by a graph whose edge weights are real numbers.

Hint 2: $\log(xy) = \log(x) + \log(y)$

(b) Oski is fed up of manually checking exchange rates, and has asked you for help to write a computer program to do his job for him. Give an efficient algorithm for detecting the possibility of arbitrage.

For both parts (a) and (b), give a three-part solution.

Solution:

(a) **Main Idea:**
We represent the currencies as the vertex set V of a complete directed graph G and the exchange rates as the edges E in the graph. Finding the best exchange rate from a to b corresponds to finding the path with the largest product of exchange rates. To turn this into a shortest path problem, we weigh the edges with the negative log of each exchange rate, i.e. set $w_{ij} = -\log r_{ij}$. Since edges can be negative, we use Bellman-Ford to help us find this shortest path.

Proof of Correctness:
To find the most advantageous ways to converts c_a into c_b, you need to find the path $c_{i_1}, c_{i_2}, \cdots, c_{i_k}$ maximizing the product $r_{i_1,i_2} r_{i_2,i_3} \cdot \cdots \cdot r_{i_{k-1},i_k}$. This is equivalent to minimizing the sum $\sum_{j=1}^{k-1}(-\log r_{i_j,i_{j+1}})$. Hence, it is sufficient to find a shortest path in the graph G with weights $w_{ij} = -\log r_{ij}$. The correctness of the entire algorithm follows from the proof of correctness of Bellman-Ford.

Runtime:
Same as runtime of Bellman-Ford, $O(|V|^3)$ since the graph is complete.

(b) **Main Idea:**
Just iterate the updating procedure once more after $|V|$ rounds. If any distance is updated, a negative cycle is guaranteed to exist, i.e. a cycle with $\sum_{j=1}^{k-1}(-\log r_{i_j,i_{j+1}}) < 0$, which implies $\prod_{j=1}^{k-1} r_{i_j,i_{j+1}} > 1$, as required.

Proof of Correctness:
Same as the proof for the modification of Bellman-Ford to find negative edges.

Runtime:
Same as Bellman-Ford, $O(|V|^3)$.

Note:
Both questions can be also solved with a variation of Bellman-Ford's algorithm that works for multiplication and maximizing instead of addition and minimizing.

3 Not So Crazy Delivery

PNPenguins are running a new food delivery business where they wish to deliver ice-cakes to all their fellow penguins in the shortest amount of time. The penguins community is a network consisting of m households total and r two-way/bidirectional roads. The time it takes to travel between two households h_1,h_2 is given by $c(h_1, h_2)$. Due to limited budget, PNPenguins can only afford to have a maximum of k delivery centers within the community. PNPenguins have found a subset of n households ($m >> n > k$) within the community who are willing to sell their home to PNPenguins and become a delivery center. For this problem, assume that households always get ice-cakes delivered by the closest delivery center to them.

a) Design an algorithm that helps PNPenguins to find the most optimal locations of delivery centers; that is, a set of delivery centers that minimizes the maximum delivery time needed to all households. This algorithm need not run in polynomial time with respects to all variables. **Please provide a 3-part solution.**
Hint: First write an algorithm which computes the maximum delivery time for some fixed set of delivery centers. Then, use this algorithm to check the maximum delivery time for each possible set of delivery centers.

Solution:
Algorithm: For each possible subset of size k out of the n total locations, we construct the graph G such that each household is represented by a vertex v_i and each road is represented by an edge with weight $c(h_i, h_j)$ between vertex v_i and v_j. We then connect the households who are chosen to be delivery centers to a dummy node with edges of weight 0. Finally, we run Dijkstra on the dummy node whose delivery time is initialized to 0. We can keep track of the maximum delivery time we have seen so far as we run Dijkstra or do a linear scan at the end of Dijkstra. After we finish iterating through all possible subsets, we output the one that minimizes the maximum delivery time.

Proof of Correctness: Since we connected delivery centers to dummy node with edges of weight 0, all delivery centers will have an initial delivery time of 0. Then running Dijkstra is guaranteed to find the shortest path between any household to its nearest delivery centers. (Note in this case we have multiple source nodes as opposed to a single source node in regular Dijkstra)

Runtime Analysis: For each iteration over a possible subset, we need $O(m + r)$ to construct the graph and $O((m + r)logm)$ to run Dijkstra. Since we have a total of n choose k possible subsets, the final overall runtime will be O($\binom{n}{k}$ * (m+r)logm).

PNPenguins have now decided to use the locations you chose from part a). But right before they start the delivery service, Pesla (a highly innovative iceship company headquartered in PNP-Borderland) releases a new magical iceship model that allows passengers to skip over the t most time-consuming roads along any of their journeys. PNPenguins all agree to integrate this new technology into their delivery business.

b) Design an algorithm that finds the updated/new maximum delivery time for each household.
Please provide a 3-part solution.
Hint: create a graph such that edges and nodes encode information about whether an edge is skipped when traversed by a graph algorithm.

Solution:
Algorithm: Let graph G be the same graph as in part a). For this part we will construct a new graph G' that contains t copies of graph G (but dummy node is not copied since we only need one dummy node for the entire graph G'). Specifically, we start with the set of vertices v_1,..., v_t which are copies of vertex v at levels 1 through t, and a set of edges (u_l, v_l) corresponding to (u,v) in the original graph for each level $l \in \{1, ..., t\}$. Then, for every edge (u, v) in the original graph, we will add a directed edge going from u_l to v_{l+1} and from v_l to u_{l+1} with weight of 0. We also add a directed

edge going from v_l to v_{l+1} with weight 0. These edges connect each level of our graph with the next level. Finally we run Dijkstra on G' and the delivery times we get for vertices at level k will be the new delivery time for each household.

Proof of Correctness: Each directed edge represents a chance to skip the current edge since the directed edge has a weight of 0. And because the edges between two levels are directed, we cannot return to previous levels and thus it's not possible to skip more than k edges. Once we have reached level t, that means we have taken exactly t skips over the entire journey. For households that are less than t edges away from the nearest delivery center, we added in the zero-weight directed edge between copies of the same vertex to make sure their copies at level t will have the correct value.

Runtime Analysis: Constructing graph G' will take $O(t * (m + r))$ since we have t copies of the original graph G. Running Dijkstra on G' will take $O((tm + tr)log(tm))$ time. So the overall runtime will be $O((tm + tr)log(tm))$.

4 Three-Legged Race

You are a teacher for two classes, each of n students. You are organizing a three-legged race, where students are paired with students in the opposing class. Each student must be paired up, and each student will only be paired with one other student. In an ideal three-legged race, you and your partner are evenly matched in terms of stride. Luckily, you have data on the stride lengths of all your students. Design an algorithm to minimize the total difference in stride length between pairs.

Please provide a 3-part solution.

Solution: Main Idea: We will use a greedy algorithm, described as follows. Sort the students in each class (A and B) separately by stride length. Pair the student with the shortest stride length in Class A with the student with the shortest stride length in Class B. Continue pairing Class A's ith-stride length student with Class B's ith-stride length student, until all n students have been paired.

Proof of Correctness: We use an exchange argument. Let S be the set of pairings that our algorithm provides, and let O be the set of optimal pairings to minimize total difference in stride length. Represent student i from Class A's stride length as A_i, and from Class B as B_i. Let S and O both be sorted by A_i.

Consider the first pairing that differs between S and O. Call this pairing in S the ith pairing, with student i from Class A and student i from Class B. Then, in O, student i must have been paired with a student j in Class B, where $i < j$, since this is the first difference between S and O. Similarly, student i in Class B must have been paired with a student k in Class A, where $i < k$.

The different cases of ordering A_i, A_k, B_i, B_j are listed below. Consider exchanging the pairs in O so that Class A and Class B student is are paired together, and Class A student k is paired with Class B student j.

1. A_i, A_k, B_i, B_j. The stride length contribution from these pairings before the exchange is $(B_j - A_i) + (B_i - A_k)$, which is the same after.
2. A_i, B_i, A_k, B_j. The stride length contribution from these pairings before the exchange is $(B_j - A_i) + (A_k - B_i) \geq (B_i - A_i) + (B_j - A_k)$ after the exchange.
3. A_i, B_i, B_j, A_k. Similarly, the stride length before is $\geq$ the stride length after.

The remaining scenarios (B_i, A_k, A_i, B_j; B_i, A_i, A_k, B_j; B_i, A_i, B_j, A_k) follow similarly. Since the exchange leads to a pairing that has a total stride length of at most the optimal solution, and the exchange agrees with our greedy algorithm at each (swapping) step, our greedy algorithm is optimal.

Runtime: Sorting takes $O(n \log n)$ time, and pairing them up takes $O(n)$ time. The algorithm in total takes $O(n \log n)$ time.

5 Cars vs. Bikes

PNPenguin is an avid cyclist, and has traveled to Penguinland to go on an epic bike ride. Unfortunately for PNPenguin, however, car traffic in Penguinland makes it dangerous to ride on certain roads.

More specifically, Penguinland consists of N buildings and M unidirectional roads each connecting two buildings, with each road having a nonnegative length l_i. One of the buildings is Penguincorp, a large local company, and all the other $N-1$ buildings are houses of Penguinland residents, all of whom work at Penguincorp. Every morning, each resident of Penguinland will drive from their house to Penguincorp on the shortest possible route. Each resident has a unique shortest route.

To avoid being hit by any cars on his bike ride, PNPenguin wants to find which roads definitely won't be traveled by any Penguinland drivers on their morning commutes. Write an efficient algorithm to find this list of roads in $O((N+M)\log(N))$.

Please provide a 3-part solution.

Solution:

Algorithm: Represent the buildings and roads as a weighted directed graph. Reverse the edges in the graph, and run Dijkstra's Algorithm with the source node being whichever node Penguincorp corresponds to. Record all roads traversed by Dijkstra's Algorithm, and return a list of all roads in the graph that aren't in this list.

Proof of Correctness: By reversing the edges in the graph, running Dijkstra's Algorithm will efficiently find the shortest path from each of the $N-1$ houses to Penguincorp in the original graph. Each of these paths will represent one resident's commute, so the entire set of edges traversed by this call to Dijkstra's Algorithm will therefore represent all of the unsafe roads in Penguinland.

Runtime: Dijkstra's Algorithm takes $O((N+M)\log(N))$ to run, and checking which roads haven't been traversed by Dijkstra's Algorithm can be accomplished using a hash table, making this step of the algorithm run in $O(M)$. Overall, the runtime of the algorithm will be dominated by Dijkstra's Algorithm, making the entire algorithm run in $O((N+M)\log(N))$

6 [Coding] Shortest Paths and Huffman Encoding

For this week's coding questions, we'll implement some shortest paths algorithms, as well as huffman encoding. There are two ways that you can access the notebook and complete the problems:

1. **On Datahub**: click here and navigate to the `hw04` 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 `hw04` 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 4 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.