Homework 3

Dynamic Programming and Graph Search

Matt Demusz, Frank Porco, Joyce Wang

Part A (35 points)

Problem 1: You are participating in the design of a new stepping stones challenge for the remake of the cult Japanese TV show “Takeshi’s Castle”.

The challenge involves a team of two people tied with a rope that need to walk over a sequence of stepping stones. The first teammate is allowed to go over the stepping stones that are painted red $\{r_0, ..., r_n\}$, while the second teammate is allowed to go over the stepping stones that are painted blue $\{b_0, ..., b_m\}$. For the team to win, the two players have to walk over all the stepping stones in the corresponding sequence while they are connected with the rope. The teammates are not allowed to backtrack. At each point in time, either one of the players can jump from her current stone to the next one and the other one stays at his current stone, or both of them jump simultaneously from their current stones to the next ones. Such a jump is obviously feasible only if the distances between the two players before and after the jump are less than the length of the rope connecting them.

The TV show producer has already built the two sequences of red and blue stepping stones. Given the coordinates of the stepping stones, your task is to select the minimum length of the rope for which it is possible for the two players to win the game. This is what will make the show entertaining for the audience!

Provide an algorithm for this computation and argue its running time.

Part B (35 points)

Problem 2:

A. You have a collection of n distinct chopsticks of length $l_1, ..., l_n$. Any two of them can be paired for use if the length of them differ at most k. How can you easily pair as many of the chopsticks as possible? Describe a greedy algorithm of time complexity $O(n \log n)$ to solve this problem and prove the correctness of your algorithm.

First, we can sort the chopsticks from shortest to longest using mergesort. The runtime of mergesort is $nlogn$. After we sort them, we can start from the shortest chopsticks and grab them in pairs. In the pair that was grabbed, we can take the longer chopstick length and subtract that by the shorter chopstick length. If the result of the subtraction is less than or equal to k, then we can put those chopsticks aside as a pair. If the result of the subtraction is greater than k, then we can discard the smaller chopstick in the pair and grab the next chopstick in the list of sorted chopsticks to pair with the "greater" chopstick if their difference in length is less than or equal to k. Traverse through the list of chopsticks by always looking at the smallest two chopsticks, and repeat the subtraction process above until no chopsticks are left to pair. Traversing through the list of chopsticks requires one iteration only, giving us a running time of n, so we have $O(nlogn + n)$, which is equal to $O(nlogn)$.

B. Consider now a variant of the above problem. You can still only pair chopsticks that differ at most k in length. But now a value w_i is also associated with each individual chopstick. You want to maximize the sum of the values of the chopsticks that have been paired.

For example, suppose you have 7 chopsticks of length $5, 2, 3, 11, 9, 12, 16$ and corresponding values $1, 1, 2, 5, 3, 3, 10$. You are allowed to pair chopsticks that differ by at most 3 units in length. Then one of the optimal solutions here is $\{(2, 3), (9, 11)\}$ of optimal value $1+2+3+5 = 11$.

How can you pair the chopsticks so as to maximize the value? Describe a dynamic programming algorithm of time complexity $O(n^2)$ to solve this problem. Can you do better than $O(n^2)$?

Part C (20 points)

Problem 3: In the robotics lab the new robot has just arrived. The robot has the ability to construct a topological map of the environment, such as the graph shown in Figure **??**. The robot is allowed to move only forward along the directions of the edges on the topological map. Moreover, the graph is being constructed in such a way that will prevent the robot to execute loops, i.e., the robot is not able to visit a node that it has already visited.

A. Given a start location for your robot and a target location, provide an efficient algorithm that will return all the possible paths from the start to the target. What is the running time for your algorithm?

B. You want to check if the topological map provides enough information for your robot to be able to visit all the rooms so as to clean them. Provide an efficient algorithm that will be able to check if there is a path for the robot on the graph that can visit all the rooms (i.e., nodes on the graph).

Part D (20 points)

Problem 4: You are preparing a banquet where the guests are government officials from many different countries. In order to avoid unnecessary troubles, you are asked to check the list of international conflicts in the last ten years. Then, you will assign the guests to two tables, such that in each table, any two guests are not from countries that had conflicts in the last ten years.

Provide an efficient algorithm that determines whether it is possible to make such an assignment. If it is possible to do so, the algorithm should return the assignment of these two tables. What is the running time?

Given a list of international conflicts of size n and a list of guests from each country of size m:

1. Iterate through the list of guests and create a hash map where the key is the contry and the value is the guest's name. ($O(m)$)
2. Create a queue that will be used to store the first node of each disconnected graph.
3. Traverse the list of international conflicts. ($O(n)$) For each country1 and country2 that has a conflict:
    - Use the hash map to check if country1 has a person ($O(1)$). If country1 has a person and is not already on the graph, then add that country's person as a node to the graph ($O(1)$). If country1 has person and is already on the graph or if country1 does not have person, then don't add anything to the graph.
    - Use the hash map to check if country2 has a person ($O(1)$). If country2 has a person and is not already on the graph, then add that country's person as a node to the graph ($O(1)$). If country2 has person and is already on the graph or if country2 does not have person, then don't add anything to the graph.
    - If both nodes are on the graph, then a strongly connected relationship should be denoted between these two nodes.

- If a disconnected graph has been created, then add one of the nodes from that graph to the queue so that we can keep track of all of the disconnected graphs ($O(1)$).

4. Create two arrays: one for table1 and another for table2.

5. Dequeue from the queue ($O(1)$), and starting from that node, and using BFS, traverse through one of the graphs. At the first node of that graph, add it to table1. For all the neighbors of that node, add them to table2 if they do not already exist in table2 since being neighbors denote a conflict, which means that two nodes that are neighbors cannot be in the same table, or array.

6. Continue to traverse using BFS until all nodes have been visited, and everytime we're on a new node, make sure that it does not already exist in a table and that it is not added to the same table as its neighbors that have already been visited. If we find that we cannot add a node to either table because it already has visited neighbors in both tables, then we stop and return false. Otherwise, we continue to dequeue and repeat steps repeat steps 5 and 6 until the queue is empty, and we return the two arrays of tables.

Without taking into consideration the running time of BFS, we have $O(m+n)$ so far. For BFS, let E be the count of all edges in the graph, and there are m nodes, or vertices. The total running time is of BFS is $O(|E|+|m|)$. Therefore, we have $O(m+n+E+m) = O(m+n+E)$.