Hua Yang
Alexio Mota
Erik Kamp

Homework 3

Dynamic Programming and Graph Search

Deadline: April 4, 11:59pm.
Available points: 110. Perfect score: 100.

Homework Instructions:

Teams: Homeworks should be completed by teams of students - three at most. No additional credit will be given for students that complete a homework individually. Please inform Athanasios Krontiris **if your team has changed from the previous assignments.** (email: ak979 AT cs.rutgers.edu).

Submission Rules: Submit your solutions electronically as a PDF document through `sakai.rutgers.edu`. Do not submit Word documents, raw text, or hardcopies etc. Make sure to generate and submit a PDF instead. Each team of students should submit only a single copy of their solutions and indicate all team members on their submission. Failure to follow these rules will result in lower grade in the assignment.

Late Submissions: No late submission is allowed. If you don't submit a homework on time, you get 0 points for that homework.

Extra Credit for LaTeX: You will receive 10% extra credit points if you submit your answers as a typeset PDF (using LaTeX, in which case you should also submit electronically your source code). Resources on how to use LaTeX are available on the course's website. There will be a 5% bonus for electronically prepared answers (e.g., on MS Word, etc.) that are not typeset.

25% Rule: For any homework problem (same will hold for exam questions), you can either attempt to answer the question, in which case you will receive between 0 and 100% credit for that question (i.e., you can get partial credit), or you can write "I don't know", in which case you receive 25% credit for that question. Leaving the question blank is the same as writing "I don't know." You may get less than 25% credit for a question that you answer erroneously.

Handwritten Reports: If you want to submit a handwritten report, scan it and submit a PDF via Sakai. We will not accept hardcopies. If you choose to submit handwritten answers and we are not able to read them, you will not be awarded any points for the part of the solution that is unreadable.

Precision: Try to be precise. Have in mind that you are trying to convince a very skeptical reader (and computer scientists are the worst kind...) that your answers are correct.

Collusion, Plagiarism, etc.: Each team of students must prepare its solutions independently from other teams, i.e., without using common notes or worksheets with other students or trying to solve problems in collaboration with other teams. You must indicate any external sources you have used in the preparation of your solution. Do not plagiarize online sources and in general make sure you do not violate any of the academic standards of the course, the department or of the university (the standards are available through the course's website). Failure to follow these rules may result in failure in the course.

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 $\{r_0, \ldots, r_n\}$, while the second teammate is allowed to go over the stepping stones that are painted blue $\{b_0, \ldots, b_m\}$. 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.

Answer

- **Variables:**
  - red_array of (x, y)
  - blue_array of (x, y)
  - variable to track current min-max length between two points
- **Objective:**
  - Find max length of distance between two points in red_array and blue_array while minimizing it.
    - int red_index = 0;
    - int blue_index = 0;
    - int rope_length = 0;
  - //stop when one player reaches the end, remaining player can move however many steps he has remaining while(red_index < red_array.length && blue_index < blue_array.length) {
    - //distance if we move up red array one spot
    - int dist1 = distanceBetweenPoints(red_array[red_index+1], blue_array[blue_index])
    - //distance if we move up blue array one spot
    - int dist2 = distanceBetweenPoints(red_array[red_index], blue_array[blue_index+1])
    - //distance if we move up blue & red array one spot
    - int dist3 = distanceBetweenPoints(red_array[red_index+1], blue_array[blue_index+1])
    - //Next determine shortest distance that can be traveled and update rope_length if it's greater than current rope length
    - if(dist1 < dist2 && dis1 < dist3) {
      - red_index++;
      - rope_length = updateRopeLength(rope_length, d1)
      - continue;

        * }
        * if(dis2 < dist3) {
            · blue_index++;
            · rope_length = updateRopeLength(rope_length, d2)
            · continue;
        * }
        * red_index++;
        * blue_index++;
        * rope_length = updateRopeLength(rope_length, d3)
    - }
    - function updateRopeLength(current_len, candidate_len) {
        * if(candidate_len > current_len) {
            · return current_len;
        * }
        * return candidate_len;
    - }

Part B (35 points)

Problem 2:

A. You have a collection of n distinct chopsticks of length $l_1, \ldots, l_n$. 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.

Answer

- Sort the collection of n distinct chopsticks, use merge sort. Running time is would be $nlogn$.
- Now go through the sorted list sequentially and pair items as soon as they meet the condition of having a length difference at at most k.
- **Pseudocode:**
    - chopstick_list
    - sort(chopstick_list) //$nlogn$
    - //loop through list, takes at most n
    - while($i + 1$ < list_length) {
        * if((chopstick_list[$i + 1$] − chopstick_list[i].length) <= k){
            · //pair both chopsticks
            · $i+ = 2$
        * }
        * else{
            · //if the difference between the chopstick[i] and choptiskc[$i + 1$] is greater than k, then you ignore the chopstick at chopstick[i] because the chopstick_list[$i + 2$] will have a length greater than the chopstick[i] since the list is sorted thus wonâĂŹt be able to pair with any other chopstick in the list
            · $i++$
        * }
    - }

- **Running time:** $nlogn + n -> O(nlogn)$

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 $O(n^2)$?

Answer

- **Properties:**
  - S = collection of n distinct chopsticks
  - Each chopstick has a length of i and a value of v
  - Each chopstick has a value per length of $t = w/i$
  - A pair of chopsticks can differ at most k in length
- Use the Fractional Knapsack method to find the optimal value. Set the limit for the Fractional Knapsack to k.
- For each iteration, perform the following:
  1. Make the greedy choice of picking and removing the chopstick with the highest value currently available in S.
  2. Make the greedy choice of picking the next highest value currently available in S.
  3. Check the length of first ($i1$) and second ($i2$) chopstick:
     - If $|i1 - i2| <= k$, remove the second chopstick from S, add both first and second chopstick to the Knapsack, set $k = |i1 - i2|$, GOTO step 4.
     - if $|i1 - i2| > k$, GOTO step 2.
     - if $|i1 - i2| > k$ and have checked all available chopsticks in S, GOTO step 4.
  4. Check the following:
     - if k = 0, print what is in the Knapsack and end the program.
     - if there are 1 or no chopstick left in S, print what is in the Knapsack and end the program.
     - if there are more than 1 chopstick in S, GOTO step 1.
- The removal of the chopstick from S takes $\Omega(logn)$. There are n removals, where n = number of chopsticks in S. The iteration takes $O(n)$ times. The total time is $O(nlogn)$, which is 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 1. 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

Figure 1: An example of a directed graph that the robot build for this map.

algorithm?

Answer

- Should use Dijkstras to find all possible paths from a start to the target. Dijkstras will traverse the graph starting at the start point and exploring each possible path. The runtime will occur when all edges are checked for an optimal path to the target which will take $O(|E+|V|Log|V||)$ where E represents the number of edges in the graph and V represents the number of vertices in the graph.
- The robot would execute Dijkstras the following way :
  - **do{**
    - Visit start/current location and add all neighbors location to memory.
    - Then visit each direct neighbor marking the current path in memory noting the distance and setting start/current node as exhausted.
  - **}while(there are unexhausted neighbors, or in other words when a node is not in the robots mem)**

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).

Answer

- Unlike the previous problem in this problem we just want to find if there exists a path between two nodes on the graph and not exactly know the shortest or know all the paths from one node to the other. Therefore, if we just want to find a path we can use either Breadth First Search, which will examine all nodes one away, then all nodes two away and so on until the target is reached, or can use Depth First Search which will continue to search the graph until the end is reached and then recurse backwards looking at subtrees to find the target. Both of these extensive searching algorithms will take O(|E|+|V|) time to run where e is the number of edges and v is the number of vertices or nodes.
- The robot would execute DFS the following way:
  - **do{**
    - Visit start location and mark as visited
    - Then continue to follow edges or directions to nodes marking them as visited until the target is reached
    - If target not reached and sink encountered, backtrack to beginning and check unexplored different path.
  - **} while(subtree or start contains unexplored node frontier)**
- The robot would execute BFS the following way:
  - **do{**
    - Visit start/current location and mark as visited
    - Then visit all neighboring nodes marking them as visited
    - Set current to one of the neighbors that are not explored marking as explored and visit its neighbors.
    - If current visiting set contains all explored set current visiting set equal to one of the exhausted neighbors visited set

– } while(subtree or start contains unexplored node frontier/neighbor visited set)

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?

Answer

- In order to solve this problem one should traverse the guest list given coloring the guest one color and their enemies another color. In other words when a guest is encountered in the list assign the guest to one table and their enemies to another table. Continue to traverse the list of guests assigning the guest to one table and their enemies to another table until the guest list is exhausted. Each time when visiting a guest make sure that the guest and their enemies are assigned a different table/different color. This might not be possible if there are color or table constraints that cannot be avoided such as two enemies also being enemies of each other. This graph coloring is termed vertex coloring, where each guest holds an edge with each one of their enemies. In vertex coloring no two connected vertices by an edge share the same color.
- Given that there are n guests at the banquet this will take an $O(n)$ time to traverse the list of guests and we know that a guest cannot be enemies with n other guests because one cannot be an enemy of themselves, therefore this will take $O(n) * O(c)$ where c is some constant where $c <= n - 1$. Due to the fact that $n - 1 < n$ we can drop the constant c and say the runtime is $O(n)$.