Write Up Solutions

Authors: Divya Samaroo (1:40PM Section), Choun H. Lee, Kenneth Hill, Mohammed Rahat

Problem 1:

a) In order to achieve an optimal substructure using a recursive solution, there are two possible outcomes for each trial.

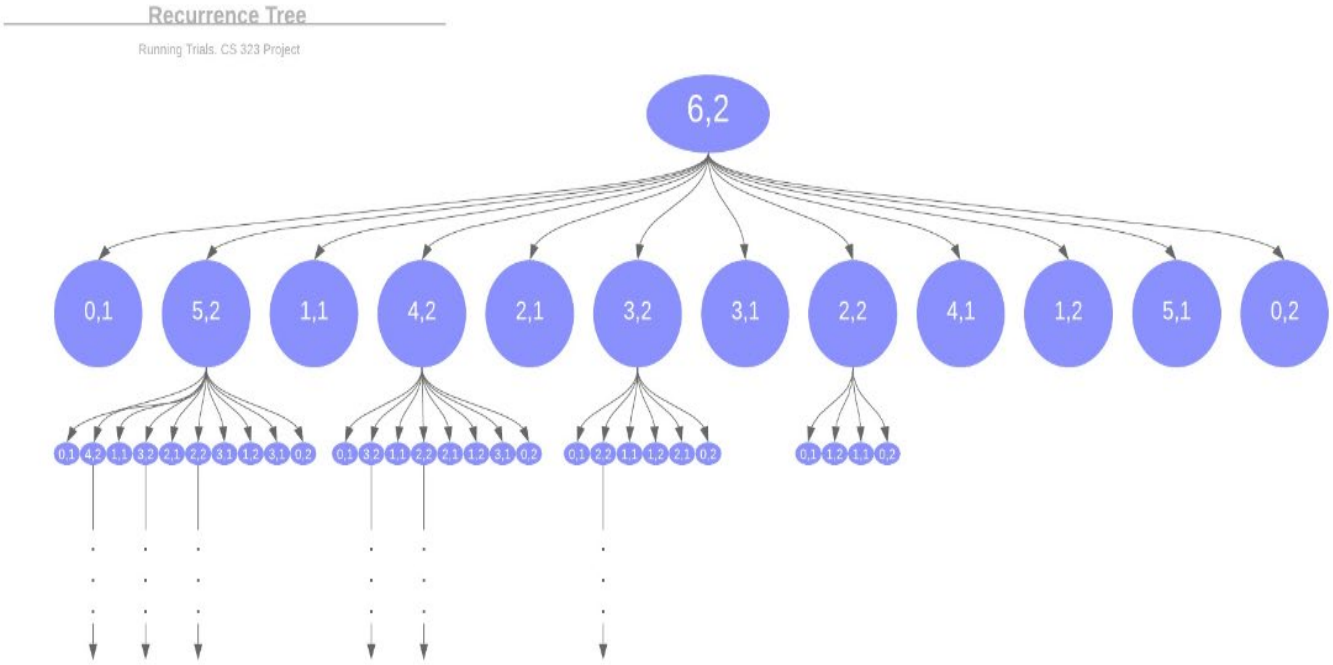
Outcome 1: The athlete gets injured and we know the maximum speed possible which is the number of tests.

Outcome 2: The athlete doesn't get injured and can continue training, and uses up remaining day.

In order to minimize the number of trials, we take the maximum of two cases. We consider the max of above two cases for every athlete and choose the athlete which yields minimum number of trials.

b) Coded

c)



d) With 6 speeds and 2 days, there are 12 distinct sub problems.

e) There are N * M distinct sub problems for N speeds and M days.

f) To memorize the code, we could attempt to put the speeds and times in a dimensional array with N rows and M columns where N is the speeds and M is the days. The results of each sub problems would be stored in their respective row and column (5 speeds and 1 day would be stored in the index [5][1]. The only issue would be the wasted space given that the indexes with 0 need to be initialized to 0. You'd end up with a whole column and row of all the 0's.

g) Coded

Problem 2:

a) The optimal substructure of this problem is to take the chef with the longest consecutive steps. Then take the next chef with the longest consecutive steps that the previous chef did not execute.

b) The greedy algorithm requires that you take all the chefs and find out who has the longest consecutive step. Then assign that chef to the recipe and among the remaining chefs, check for the longest consecutive step that the previous chef did not execute and assign that chef. Repeat this process continuously until there are no more steps remaining. However, in case of two chefs having identical length of longest consecutive steps, choose the chef that will leave the least number of consecutive steps remaining in the recipe.

c) coded

d)O(2n) + O(n^3) The first O(n) creates an empty array filled with zeros, to store the data of the steps that need to be completed. Our algorithm has a while loop containing 3 for loops, the while loop is not iteratively ran, but it can be at worst case, which is when chefs are able to do only 1 step each, which then makes it n time. Our last for loop will run n times only no matter how many times the outer loop runs, thus we are resulted with 2 separate loops running n times independently to give us O(2n) + O(n^3) for the nested loops

e) Proof by contradiction:

Assume there is an optimal algorithm (OPT)

L = Longest available consecutive step

ALGO = L1 + L2 +... Li ...Ln

OPT = L1` + L2` +...Li`...Ln`

<L1, L2...Li> and **<L1`, L2`....Li`>** where 1 to the i will represent the smallest number of distinct longest available consecutive steps until no more steps are remaining.

Claim: **i (1 <= i <= n) is the smallest index where n-i! = n`-i`**

Keep modifying the optimal solution using the claim that we cut and paste:

ALG = L1` + L2` +... Li`....Ln` ...Ln

OPT= L1` + L2` +... Li` ...Ln`

If there exists an optimal solution, then n` should be less than n. If that is true, then if we cut the primes into our algorithm there should be remaining L's from n` to n. Since n` = n, opt perfectly cut and pastes to our algorithm without any remainder, thus creating a contradiction. Therefore, our greedy algorithm will give us an optimal solution.

Problem 3:

a) Dijkstra's shortest path algorithm compares all the edges that are connected to the current vertex and then chooses the smaller edge. Then we visit the shorter edge and compare all the adjacent vertices that are yet to be visited. Every time you visit a vertex update the shortest path tree set. The set is the shortest path you are visiting. Pick the shortest path again, visit the shorter of those adjacent vertices that has yet to be visited until there are no more vertices left to visit.

b) Given r is the number of rows in matrix, e is the position of the n vertex.

T(r) + T(r) * e = O (r * e)

c)The generic shortest method is implementing Dijkstra's algorithm to find the shortest path of a graph.

d) Our goal is to find the shortest path and genericShortest finds the shortest path. The only missing part is to calculate what time the next quest that allows you to advance from u to v takes place. Using the helper method find getNextQuestTime, we are able to find the time it will take for the next quest to be available. Adding the time to our given play time graph we are able to find the total time it will require to move to the next quest.

e) The time complexity for the generic shortest method is:

T(V) + T(V^2) = O(V^2)

Where V is # of vertices.

f) See code

During the project we all worked collectively and helped each other whenever someone was struggling to understand a question. The coding was done in pair programming via TeamViewer. We all met in the library to work on the code as well. We forked the code from the main person's profile and pushed any changes /comments to that profile. Using google drive we created a google doc to produce our write up questions. This allowed for everyone to edit the file and see the progress made so far. Each person was assigned specific questions in the write up. However, we all still worked together to help someone if they did not understand the question they were given. Our group made sure no one struggled and we made time to meet up. We tried to devote as much time as we could to meet up together as a group and express our thoughts and progress so far on the project. Google drive helped us to keep track of the questions and made a big difference in communication. We could each see when someone was working on the doc and exactly what they were editing.

Working together as a group showed us that by brainstorming ideas together, we could build off of each other's thoughts and share our knowledge. Divya and Choun worked on write up problem number two. Kenneth and Mohammed worked together on problem number one. We all worked on problem three together as well. Together we still reviewed each problem, discussed the solutions and how someone came up with a certain solution and if there was anything we could add. Working together as a team had a very positive impact because we provided support and encouragement to each other. This was a very creative way to work on a project in order to achieve our goals as a team.