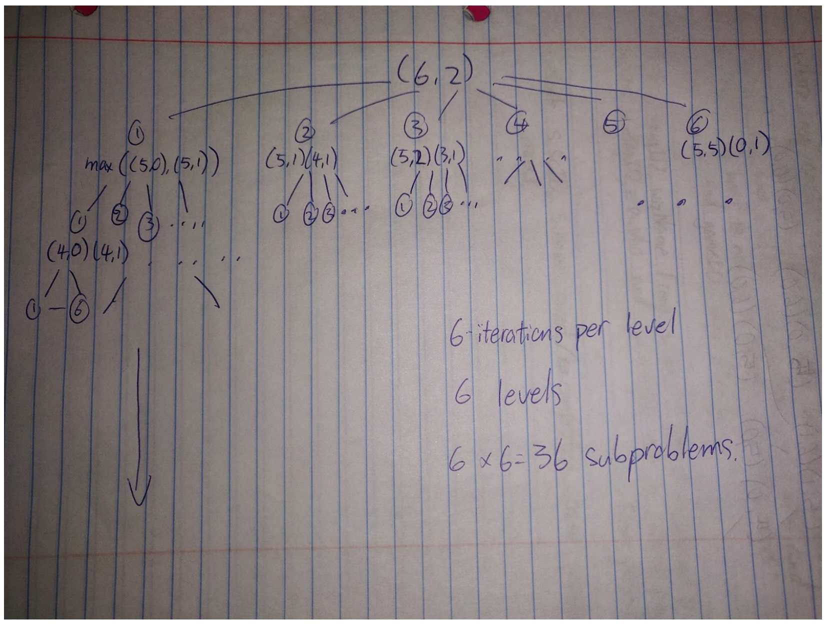
Group: Mohammed Chowdhury, Asad Malik, Nj Lin, Kareem El Sayed

<u>**Running Trials and Tribulations**</u>

(a) Describe the optimal substructure/recurrence that would lead to a recursive solution

We need to check if the athlete is injured or not, if they are, check the previous speed from the day before, if they are not, check the remaining speeds.

(c) Draw recurrence tree for given (# speeds = 6, # days = 2)



(d) How many distinct subproblems do you end up with given 6 speeds and 2 days?

There are 36 distinct subproblems

(e) How many distinct subproblems for N speeds and M days?

There will be N^2 subproblems

(f) Describe how you would memoize runTrialsRecur.

Create a table that stores the results of each sub problem. At each iteration of runTrialsRecur, it will search the table to see if it already contains the result, if the result already exists, it simply reuses the result. This ensures that the function will run at a constant speed and doesn't waste time on overlapping subproblems.

Holiday Special - Putting Shifts Together

(a) Describe the optimal substructure of this problem.

At each stage, we pick the volunteer that can perform the most consecutive steps in order to make sure that we have the least switching between the cooks.

(b) Describe the greedy algorithm in plain words that could find an optimal way to schedule the volunteers for one recipe.

Pick a volunteer that can perform the most consecutive steps starting with step 1, and when that volunteer come across a step that they cannot perform, find the next volunteer that can perform the most consecutive steps.

(d) What is the runtime complexity of your greedy algorithm? Again, you don't need to factor in the setup of the signup table, just your scheduling algorithm.

O(nlogn)

(e) In your write-up file, based on your answer to part b, give a full proof that your greedy algorithm returns an optimal solution.

Suppose there was a more optimal solution, and at the ith iteration of the optimal method differs from our greedy method. This must mean that the optimal solution has assigned a volunteer that can perform the same number of steps as the volunteer from our greedy algorithm. If this is the case, we can swap the volunteers and if would not make change the minimum number of switches. The volunteer of the optimal solution must not be able to perform more consecutive steps than the volunteer from our greedy algorithm, because by design, our greedy algorithm picks the volunteer that can perform the most consecutive steps. We can keep checking until the end, resulting in the optimal algorithm = greedy algorithm.

League of Patience

(a) Describe an algorithm solution to this problem. Feel free to talk about how you would adapt an algorithm we covered in class.

Since the game map is can be represented as a connected graph, we can use the shortest path algorithm to solve this problem. One algorithm that comes to mind is Dijkstra's algorithm, we can assign gameplay time as the weight for the edges between two locations on the graph.

(b) What is the complexity of your proposed solution in (a)?

O(ELogV)

(c) See the file LeagueOfPatience.java, the method "genericShortest". Note you can run the LeagueOfPatience.java file and the method will output the solution from that method. Which algorithm is this genericShortest method implementing?

Dijkstra’s algorithm

(d) In the file LeagueOfPatience.java, how would you use the existing code to help you implement your algorithm? The existing code only handles one piece of data per edge, so describe some modifications. Note the helper methods available to you, including one that simulates the game's API that returns the next quest time.

The genericShortest method performs similarly the function, but it doesn’t account for the time difference between the current time and the next quest start time. We need to modify the genericShortest by using existing methods, getNextQuestTime and minutesBetween to add the in between into the comparison.

(e) What's the current complexity of "genericShortest" given V vertices and E edges? How would you make the "genericShortest" implementation faster? Describe any algorithm changes or data structure changes. What's the complexity of the optimal implementation?

O(V+E)

Summary

All four members of our group had an exam on Wednesday so before then we weren't able to make much progress on the assignment. We had all looked at it individually and thought about possible approaches but it all came together when we meet up after our exam. We all sat together and approached each part of the assignment together. We all shared our ideas and put it together to do the projects. We did as much as we could for each project and when we were stuck, we noted the issue and moved forward as much as we could. We meet up again on Friday and Saturday and went thru our codes and fixed errors and made improvements. The code and the writeup went hand in hand in most parts so we noted what we did as we wrote the code. After the code was done we had most of the answers already. The rest we discussed as a group and found the answers for. We felt this was the best way because the code seemed complicated and we figured that if we did it as a group we would be able to work through it quicker and better.

Working together on this project was a new experience for all of us. None of our previous programming classes has us working together on one code before. In those situations we were all alone and when stuck on a problem, we didn't have a different approach to it so it took a long time to find a solution. When doing this project we had four minds brainstorming at the same time on every issue. So when we ran into an error we had 4 different perspectives on how to solve it. Even when our code didn't have errors, we were able to improve on each others ideas. A few of us are also enrolled in the Internet and Web Technologies class in which the final assignment is a group project. Working on this together provided each of us with insight on how well it works and also how to work together more efficiently in our individual groups.