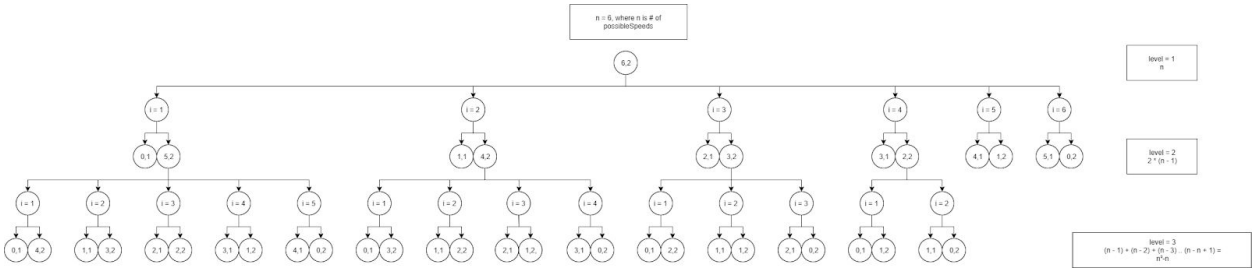
Wynnter Ong & Kyle Tapang
CS 323 - Design & Analysis of Algorithms

Problem 1: Running Trials

---

a) Describe the optimal substructure/recurrence that would lead to a recursive solution.
   i) If an athlete tries possibleSpeed x, one of two things will occur. He/she will either end up injured or not injured. If the athlete is injured, then other possibleSpeeds greater than x will also injure the athlete. Therefore, we only need to test the possibleSpeeds less than x. In this case, the problem is reduced to x - 1 possibleSpeeds and y - 1 days. If the athlete isn't injured, then we only need to test speeds greater than x. In this case, the problem is reduced to n - x speeds (where n is the top possibleSpeed) and y days. Of the two cases, we chose the greater for each speed. Then we choose the speed with the least amount of trials.

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



c) How many distinct subproblems do you end up with given 6 speeds and 2 days?
   i) There are 12 distinct subproblems.

d) How many distinct subproblems for N speeds and M days?
   i) There are n x m distinct subproblems.

e) Describe how you would memoize runTrialsRecur.
   i) I would create a 2D array where I would store solutions to subproblems, such that my algorithm doesn't need re-solve the same subproblems.

Wynnter Ong & Kyle Tapang
CS 323 - Design & Analysis of Algorithms

Problem 2: Holiday Special

---

a) Describe the optimal substructure of this problem
   i) The optimal solution for any given step contains the optimal solution for the previous step.
   ii) A subschedule from an optimal schedule is itself an optimal schedule, assuming that the subschedule starts from the first step.
b) Describe the greedy algorithm in plain words that could find an optimal way to schedule the volunteers for one recipe.
   i) Going through each step in the recipe, if there is a cook selected it will immediately pick that cook for the step, otherwise it will go through each cook, and keep track of the longest streak of steps from there and the cook that has that streak. The cook that has the longest streak will be selected and be kept track for future steps in the recipe.
c) Code
d) What is the runtime complexity of your greedy algorithm?
   i) $O(n) = S * W+1(C*S)$
   Where S = # of Steps, C = # of Cooks, and W = # of switches, +1 is because it has to check all cooks at least once for the first step.
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.
   i) An optimal schedule will contain an optimal subschedule, assuming that the subschedule starts from the first step. The first optimal subschedule would be from the cook with the longest streak of consecutive steps, starting from the first step. The next optimal subschedule to our solution would include the previous subschedule + the cook with the longest consecutive streak starting from where the previous cook left off. This repeats until an optimal schedule is created using all steps.

   For the very first step, k = 0, there is no selected cook with a streak so it has to check all the cooks available and determine which one has the longest streak. Once determined, the streak length and cook selected is stored in variables. After, the selected cook is confirmed for the step, and the streak length is reduced by 1. If the streak is 0, that means the selected cook is not available for the next step and the selected cook variable is reset.

   For the next step, k = 1, there are 2 cases that can happen.

1) There is no selected cook with a streak, so all cooks must be checked to determine which one has the longest streak, which will then be stored in variables. This means a switch from one cook to another has happened.
2) There is a selected cook with a streak, so no cooks need to be checked because the selected cook with the streak will be confirmed for that step. This is optimal because no switch has been made, minimizing the switches needed up to that point. If a switch was made here, that would increase the switch count which is not what the problem wants.

No matter which case, the selected cook is confirmed for the step, and the streak length is reduced by 1. If the

streak is 0, that means the selected cook is not available for the next step and the selected cook variable is

reset.

Wynnter Ong & Kyle Tapang
CS 323 - Design & Analysis of Algorithms

Problem 3: 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.
   i) This adaptation of Dijkstra's algorithm would process one node at a time, asking the API for the time of the quest and using that to calculate the minutes between. After doing so, it would update the adjacent vertices to the time of the processed vertex + the quest time + the minutes between, if the adjacent vertex was not processed yet, not the source, there is an edge between the 2 vertices, and the total weight of the path from the source to the adjacent vertex is smaller than the existing weight of the path from the source to the adjacent vertex.

      Because the goal is specifically the time between the starting location S and the destination T, I will modify the algorithm to immediately stop working once T is reached, as we have reached our destination and any further work is unnecessary.

b) What is the complexity of your proposed solution in (a)?
   i) The complexity of the proposed solution is also the same as genericShortest, which is $O(n) = V^2$

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?

   The algorithm that genericShortest is implementing is 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.
   i) The algorithm would work the same as the one in the genericShortest method, except work needs to be done to calculate the nextQuestTime. Using the nextQuestTime, the minutesBetween would be calculated, which would be also added to the adjacent vertices.

      In order to keep track of the order, a separate order[] array would be created which would hold the nodes that were selected to process in order. At the same time processed[u] is changed to true, order[count] will be changed to u. In order to speed up runtime and to do only the required work, the program will break out of the for loop if u is the destination node because that means we are at the destination.

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?
    i) The complexity of genericShortest is $O(n) = V^2$. It can be made faster with a binary heap implementation in the form of a min heap. By using min heap to store the vertices not confirmed to be in the shortest path, a priority queue can be used to get the minimum distance vertex from the vertices in the min heap.

Wynnter Ong & Kyle Tapang
CS 323 - Design & Analysis of Algorithms

Summary:

The work was divided up between Kyle and Wynnter. In short, Kyle worked on problem 1 and 2, and Wynnter worked on problem 2 and 3. For problem 1, Kyle took care of the code and write-up. For problem 2, Kyle contributed the optimal substructure and pseudocode. Wynnter contributed the code and a majority of the write-up. For problem 3, Wynnter took care of the code and write-up.

Experience:

Kyle: Working with Wynnter has been a pleasant experience. Wynnter was able to meet up with me and we discussed our approach to these problems. During our meeting we were able to knock out problem 2 and from there we decided that I would handle problem 1 and that he would handle problem 3. To my surprise, we were able to complete a majority of the project with plenty of time to spare.

Wynnter: The experience of working together was great, as we were able to figure out how to start the problems together, and work from there. Kyle was able to help fill in the blanks of what I remembered about Dijkstra's algorithm for problem 2, and his pseudocode about basing it on the longest streak helped me a lot while coding Problem 2.