Algorithms: Assignment 1 Write-up

Arhum Khan

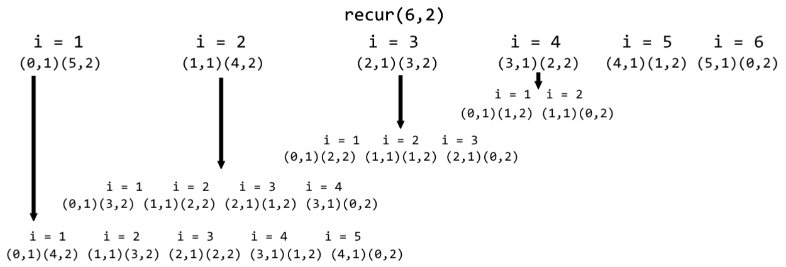
December 1, 2019

1. Running Trials and Tribulations

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

The recurrence here is that each possible speed will will either injure the athlete or not injure the athlete, which result in a call for either: 1 less day and all the possible speeds below the one injured at, or for the next possible speed with the same amount of days.

(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?

12. This can be confirmed as we use a matrix to keep track of every possible combination of speed and days left, which are each a distinct subproblem.

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

There are N x M subproblems.

(f) Describe how you would memoize runTrialsRecur.

I would use and array of possibleSpeeds by days to keep track of each subproblem such that each potential subproblem's solution is stored in the array. This way if an answer was already computed it could be found via array lookup, which is much faster than recomputing the solution.

2. *Holiday Special - Putting Shifts Together*

(a) Describe the optimal substructure of the problem.

The optimal substructure of the problem is to, first, find the most consecutive steps that can be done by a chef, take those steps out of the recipe, and again find the most consecutive steps that can be done by the next chef, and so on.

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

The first part is to figure out which chef of all the chefs is able to do the most consecutive steps, then to take all those steps out of the recipe and take the chef out of potential chefs to do the rest of the recipe, and from there figure out the chef that can do the most steps from there. This is greedy in that we are taking the chef that can do the most consecutive steps every time.

(d) What is the runtime complexity of your greedy algorithm?

O(N^3)

(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.

:(

3. *League of Patience*

(a) Describe an algorithm solution to this problem.
I settled on using Dijkstra's Algorithm since we covered it in class, but since this is already used in the method genericShortest, I decided to add some optimizations using existing helper methods that should help very minorly.

(b) What is the complexity of your proposed solution in (a)?
That would be O(V^2).

(c) Which algorithm is genericShortest method implementing?
The method genericShortest is implementing Dijkstra's Algorithm.

(d) In the file LeagueOfPatience.java, how would you use the existing code to help you implement your algorithm?
I utilized the getNextQuestTime method to add its "minutes" to our current minutes, which allowed me to easily compare if our "shortest" was actually shorter (such that I could replace it faster).

(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?
The current complexity of genericShortest is O(V^2), as it is an implementation of Dijkstra's Algorithm. To make this algorithm faster, we could use the same algorithm but with a binary heap implementation. That will alter our complexity to be O((E + V) log V).