323 Project

Joyce Tan, Hein Soe, Pratap Gurung

December 01, 2019

Instr. Carolyn Yao

Problem 1 write up file by Hein Soe.

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

- ⇨ Optimal Substructure is finding the worst case in which given certain days athlete needs to find all possible speed tests by trying all of the speed tests. For each day, athlete will only able to try range of speeds. Athlete will stop at the point He/she will injured at particular speed. Maximum trials will be that maximum speed. With multiple days (max day limit) of speed tests run, athlete will choose the minimum test run days among these days of Speedtest runs to train for actual event since that speed with minimum trial(minimum speed) is the speed required without injuring themselves.

(b) //Refer Code

(c) //Refer Next Page

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

- ⇨ 12 distinct subproblems for given 6 speeds and 2 days. A lot of them overlapped as recursive goes deeper.

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

- ⇨ N * 2

(f) Describe how you would memorize runTrialsRecur.

- ⇨ Memoization of runTrialsRecur will be done by storing the min of of particular speedtest run and day inside 2D arrays. Every next day or next speedtest run will be compared against previous run and minimum test run will be chosen for that particular trial. At last, only the minimum no of speediest trials will become the last number.

(g) //Refer Code

(h) //Refer Code

Trials(speeds, days)

Choose Min from these speed Indexes.

Example:

==> Choosing Max of return from These two + 1

Duplicate Recursive are not shown again.
Ex: T(4,2) , T(3,2) , T(2,2),….. etc.

1
2
3
………….etc
Below
above
T(0,1)
T(4,2)
Min is = 3
T(6,2)
3 + 1 = 4
3 + 1 = 4
2 + 1 = 3
3 + 1 = 4
4 + 1 = 5
5 + 1 = 6
Speed
1
2
3
4
5
6
Rtn=0
Rtn=3
Rtn=1
Rtn=3
Rtn=2
Rtn=2
Rtn=3
Rtn=2
Rtn=4
Rtn=1
Rtn=5
Rtn=0
T(0,1)
max
T(5,2)
T(1,1)
max
T(4,2)
T(2,1)
max
T(3,2)
T(3,1)
max
T(2,2)
T(4,1)
max
T(1,2)
T(5,1)
max
T(0,2)
Min is = 3
Min is = 3
3 + 1 = 4
2 + 1 = 3
2 + 1 = 3
3 + 1 = 4
4 + 1 = 5
Speed
1
2
3
4
5
Rtn=3
Rtn=2
Rtn=2
Rtn=3
Rtn=4
T(0,1)
T(4,2)
T(1,1)
T(3,2)
T(2,1)
T(2,2)
T(3,1)
T(1,2)
T(4,1)
T(0,2)
Duplicate Recursive
Duplicate Recursive
Duplicate Recursive
2 + 1 = 3
2 + 1 = 3
2 + 1 = 3
3 + 1 = 4
Speed
1
2
3
4
Rtn=0
Rtn=2
T(0,1)
T(3,2)
T(1,1)
T(2,2)
T(2,1)
T(1,2)
T(3,1)
T(0,2)
Duplicate Recursive
Duplicate Recursive
Min is = 2
Min is = 2
2 + 1 = 3
1 + 1 = 2
2 + 1 = 3
Speed
1
2
3
Rtn=0
Rtn=2
Rtn=1
Rtn=1
Rtn=2
Rtn=0
T(0,1)
max
T(2,2)
T(1,1)
max
T(1,2)
T(2,1)
max
T(0,2)
Duplicate Recursive
1 + 1 = 2
1 + 1 = 2
Speed
1
2
Rtn=0
Rtn=1
Rtn=1
Rtn=0
T(0,1)
T(1,2)
T(1,1)
T(0,2)

Problem 2 write up file by Pratap Gurung.

(a) Describe the optimal substructure of this problem

- ⇨ Let $S_{i,n}$ be the set of steps that needs to be done. S_i is the starting and S_n is the finishing steps. Consider cook C_m will do some consecutive steps $S_{i,j}$ starting from S_i to $S_{j,}$ then adding optimal solution of remaining steps n-j steps will give optimal solution to our problem.

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

- ⇨ When you are in steps S_i then find the longest consecutive subset of steps starting from S_i to some steps S_j (j <= n) done by a cook C_k and add the steps $S_{i,j}$ done by cook C_k to our optimal solution then repeat the same process for S_{j+1} until you find the optimal solution up to step $S_{n.}$

(c) //Refer Code

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

- ⇨ There are M number of cooks and N number of steps. Since for steps S_i, we iterate over at worst M number of cooks to find longest consecutive steps. At worst, we do this for N times for N number of steps. Hence, our expected runtime complexity of greedy algorithm is O(MN).

(e) Proof of Greedy Algorithm.

- ⇨ Suppose there exists an optimal solution OPT other than our greedy algorithm where we have L (the number of switch) between cooks, and our algo has K such that L < k. Let's consider:

  Alg: $C_1, C_2, C_3, C_4, \ldots\ldots\ldots, C_K$

  OPT: $C'_1, C'_2, C'_3, C'_4, \ldots\ldots\ldots, C'_k$

  Let's say I where Alg and OPT first differ. That means

  $C_1, C_2, C_3, C_4, \ldots\ldots\ldots, C_{i-1} = C'_1, C'_2, C'_3, C'_4, \ldots\ldots\ldots, C'_{i-1}$

  C_i by our algorithm designed, do the longest consecutive subset of steps as it could from steps I. Then

- Cases I: length of subset of $C_i > C'_I$ the you can replace C'_I with $C_{i.}$
- Cases II: length of subset of $C_i < C'_{i.}$ This is impossible based on our algorithm design.

If we swap, the we proceed the same way for C_{i+1} and C'_{i+1}. cut and paste. Until we reach C_L and C'_L. By design, if Alg could have ended at L i.e. L equals to K. Hence, we found contradiction to our original assumption that there exists optimal solution OPT that which does better than our algorithm. Therefore, our greedy algorithm yields an optimal solution to our problem.

Problem 3: League of Patience Write Up By Joyce Tan

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

An algorithm solution to “myFastestPlay” would be the shortest path algorithm. Where instead of just calculating the weight of paths from S(the s^{th} location on the game map) to T(the t^{th} location to the game map) to determine the shortest path, we need to account for the wait time until the next quest time is available. An example from class that I referred to was the knapsack problem. In the knapsack problem, we made a 2D array to store the maximum values we can hold at each weight until we got to the weight capacity by swapping possible values to get the maximum value. In my “myFastestPlay” solution I use an array called times to record the shortest time to each location between the S and T by swapping values to get the minimum time until I get the shortest time from S to T. The method “genericShortest” also use the shortest path algorithm and I used the existing code to help me implement my solution (More details in part d).

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

Time complexity of my proposed solution

T = the t^{th} location on the game (the endpoint of our path)

Number of subproblems = T x T (double for loop when finding shortest path from S to T)

Time/problem = O(1)

Time Complexity = $T^2 \times O(1) = O(T^2)$

(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 method "genericShortest" uses the shortest path algorithm. In "genericShortest", there is an array called times that stores the shortest times to each location on the map from the source. There is also an Boolean array called processed, where if processed[i] is true then the shortest time to the ith location from source is finalized.

In the code, we update time[v] (which represents the shortest time from source to the vth location on the map) only if all the following is true:

1. processed[v] = false (which represents that time[v] is not the finalized shortest time yet)
2. graph[u][v] != 0 (which represents that there is an edge from u to v)
3. times[u] + graph[u][v] < times[v] (which represents that [the shortest time from the source to the uth location + the duration of game play between u and v] < [the stored "shortest" time to get from the source to the vth location])

If all of these conditions are true then times[v] will equal times[u]+graph[u][v] because that time is shorter.

(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.

In "myFastestPlay", I would use the same code as "genericShortest", but I would also have to account for the required wait time for the next quest to be available. I used the helper function "getNextQuestTime" to get the start time for the next quest. Then I used the "minutesBetween" function to get the time (in minutes) between the start time of my current quest to the start of the next quest. So the minutesBetween function returned how many minutes I have to complete the game play between u and v before the next quest start time. So I calculated the wait time by subtracting the durations[u][v] (which is how long the game play takes between u and v) from the minutesBetween the two start times.

Then I used the exisiting code as described in part c, to update the time[v] with a few modifications:

1. Check if the waitingTime is greater than zero

- if the wait time is greater than zero that means that I was able to complete the game play between u and v before the next quest start time begins.

2. Check if times[u] + nextTimeMins < times[v]
   - nextTimeMins = waitingTime + durations[u][v]

- if times[u] + nextTimeMins < times[v] then that means that

[the time from source to the uth position + the duration of the game play from u to v + additional wait time for the next quest to start] < [the stores “shortest” time to get from the source to the vth location]

(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?

Current time complexity of the method “genericShortest”

V = number of vertices

Number of subproblems = V x V (from the double for loop)

Time/problem = O(1)

Time Complexity = V^2 x O(1) = O(V^2)

If the input graph is represented using adjacency list, the time complexity can be reduced to

O(E log V) with the help of binary heap.

(f) //Refer Code

(g) //Refer Code

Summary:

1. how the work was divided up between individuals/who did what in both coding and write-up?
   - ⇨ Our group involved three members, so we decided to do one problem by each member. Each member did their respective problem coding and write-up. However, we did discuss and helped each other to solve our respective problems.
   - ⇨ Running Trials and Tribulations code + write up : Hein Soe
     Holiday Special code + write up : Pratap Gurung

League of Patience code + write up : Joyce Tan

2. the experience of working together.

   I had a positive experience working with my group members. We were able to give our input on each other's work and I was able to learn how to collaborate through GitHub, thanks to my group members. - Joyce Tan