Team Member:

Chuhui Chen session 1:45

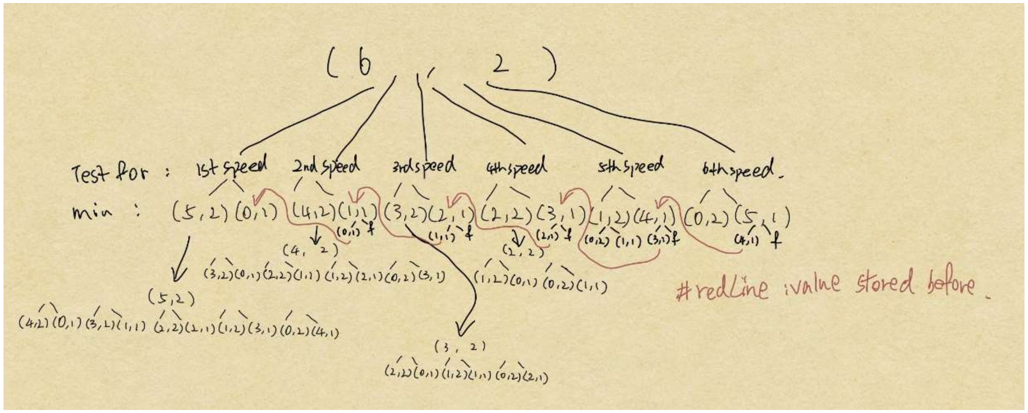
Zhiwen Jiang session 3:10

Yubo Sun session 3:10

Question 1: Running Trials:

a. Optimal structure: for testing each speed, we do the recursion to find the values of when it fails and success. After we get the values, we find the max of each pair of fail and success. After that we find the min of those max values to be the minimal test numbers we need to do if we test that speed. By finding the value for minimal test at each step, we can find the minimal test for the total number of speeds.

c.


(6, 2)
Test for : 1st speed 2nd speed 3rd speed 4th speed 5th speed 6th speed.
min : (5,2) (0,1) (4,2) (1,1) (3,2) (2,1) (2,2) (3,1) (1,2) (4,1) (0,2) (5,1)
(0,1) f (1,1) f (2,1) f (0,2) (1,1) (3,1) f (4,1) f
(4, 2)
(3,2) (0,1) (2,2) (1,1) (1,2) (2,1) (0,2) (3,1)
(2, 2)
(1,2) (0,1) (0,2) (1,1)
(5,2)
(4,2) (0,1) (3,2) (1,1) (2,2) (2,1) (1,2) (3,1) (0,2) (4,1)
(3, 2)
(2,2) (0,1) (1,2) (1,1) (0,2) (2,1)
#redLine :value stored before.

As we can see, many part of the recurrence tree can be used.

d. 11

e. M * (N – 1) + 1

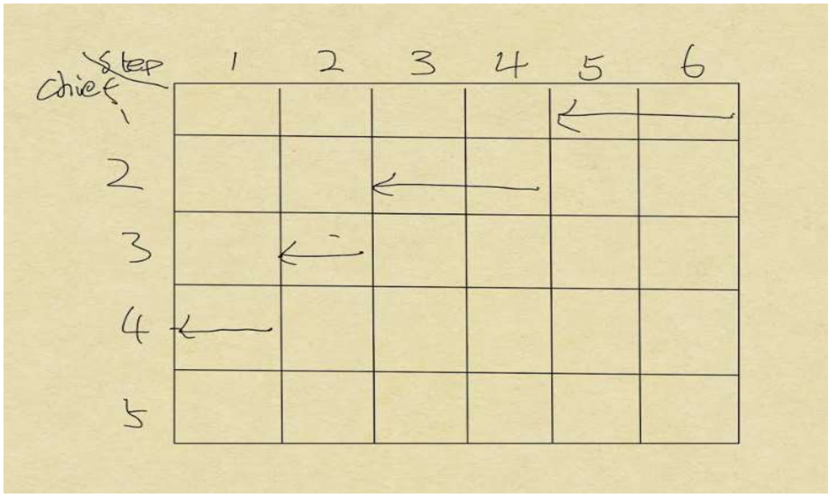
f. First we setup a 2-D array. Set row = speeds and column = days. Then we store the value when day = 1 into the array. When we run the loop, if the value is already stored in the 2-D array we use value in the array. Otherwise, we calculate the number of trails and store it into the array.

Question 2: Holiday Special

a) optimal substructure: For each step we need to find a new chief, if we pick the one that can do the most consecutive steps, we will end up the least switch for the whole cooking procedure.

b) Start from the last step. Search all chief that can do that step. For each assignable chief, we are going to calculate the longest consecutive steps he can do from back to the beginning. We will assign the chief with the longest consecutive steps among all assignable chief. Then repeat this procedure from step that picked chief couldn't do. Until we reach 1 step.

d) It looks like a O(M*N*M) at the first glance because there are M steps need to search. At each steps we need to search N chiefs and for each chief we need to search M steps. However, if we take a closer look at the nested loops, we will realize it is actually filing a N*M matrix where N is number of chief and M is number of steps. The result is something like this:



We will at most check the whole 2-D array. So the run-time complexity is actually (N*M)

e) suppose there is a better solution that provide less switches for the arrangement called OPT. Let's call our algorithm ALG. In order for OPT has less switch than ALG, there is at least two consecutive steps we assign to different chiefs in ALG can be done with one chief in OPT. However, by design of ALG, each chief contains the max consecutive steps at that time, thus it is not possible to use one chief to do the same work without switch. If it could be done, that chief will be picked by ALG already. That means OPT must have same number of switches as ALG. We have a contradiction; thus our algorithm is the optimal.

Q3: League of Patience

a) This is the classic shortest path problem. We want to know the shortest path from point S to point E giving an adjacency matrix. We can use Dijkstra's algorithm to solve this problem. We create an array to store the actual cost where index indicate stage number. We initialize value in the array to be infinity. We are going to use the max value of integer in this case since there is no such number as infinity in Java. We are going to set the index represent the starting point to be 0 since there is no cost. Then we are going to do following operation. Pick the lowest cost index, we say that vertex with its current cost must be in part of our shortest path called X. Update the cost for each vertex from the X. Replace the value if the cost is less than the value stored in the array. Repeat the operation N time until all the vertexes are picked.

b) V*V where V is the number of vertex.

c) Dijkstra's algorithm

d) Since the cost is fixed from point a to point b within the same day, I can update the matrix with cost plus waiting time using the getNextQuestTime and minutesBetween. Then I can run the genericShortest function to find the shortest path. Because I am also doing the extra credit, instead of calling genericShorest, I am going to use same code but add some line to update the path matrix while I relaxing the vertex.

e) Currently, it is V*V. There are three ways to speed up our algorithm.

   1) One way we can optimal the algorithm is stop when we relaxing our destination. So instead of completing the shortest path for all vertex, we stop when we find our target.

   2) Another way to optimize it is to a bidirectional Dijkstra algorithm instead. We need another array and priority queue but search backward. Index T(target) will be set at 0 (relax first) and rest of them are going to be infinite (max value of integer). In this additional part, we are going to reverse what we do in the generic shortest path, we are going to looking for the cost lead to the cost that all the vertex can get to the destination. we are going to relax and update the cost use the similar logic in generic shortest path. Let the operation we done in generic-shortest path called forward

search. We are going to do a forward search followed by a backward search. We repeat this cycle until we find a common Vertex W relaxed by both forward search and backward search. Then we are going to find the minimal sum of the cost from forward search and backward search where we know that point must be included in the shortest path. Let’s call that vertex x. We are going to take the union by sum forward[x] and backward[x] minus the original cost[x] to get the shortest path from S to T.

3) Yet, the above two algorithms only guaranteed to run no slower than generic shortest path algorithm. The best way to improve the algorithm is to implement a heap instead of an array to store the costs because we can find the minimal cost in O (log v) time in a heap. Combining with adjacency matrix, update time take O(log v). The runtime become O(v log v + e log v) which is better than v*v.