Sangyun Lee

HW3

Q1

Dynamic programming has some common features with divide-and-conquer in terms of solving program recursively. Based on the lecture and reading, both dynamic programming and divide-and-conquer trace down until hit the base case, and then it solves problems by combining solutions to subproblems. However, unlikely, subproblems of Dynamic programming are not unique. It means you could have many of overlapped subproblems. So, Dynamic programming saves those overlapped sub-tasks in table or array, then reuse saved data to solve the sub-problems. Its benefits programmer by resulting in constant running time.

Q2

a. dynamic programming algorithm

- Since the rook can move horizontally and vertically, so there is path h and v. And the board size is 8*8. The path restriction should be 1<= h, v <=8
- And rook can move either horizontal way or vertical way. I can say P[h,1] = P[1,v] = 1 and it is the base case
- If the P[h, v] is the number of shortest path of Rook from board[1,1] to board[h, v], then the next row or column can be researched from previous column or row. Therefore, P[h, v-1], P[h-1,v]
- Put all things together
    - P[h, v] = P[h, v-1] + P[h-1,v]
    - P[h,1] = P[1,v] = 1
    - For 1<= h, v<=8

goal

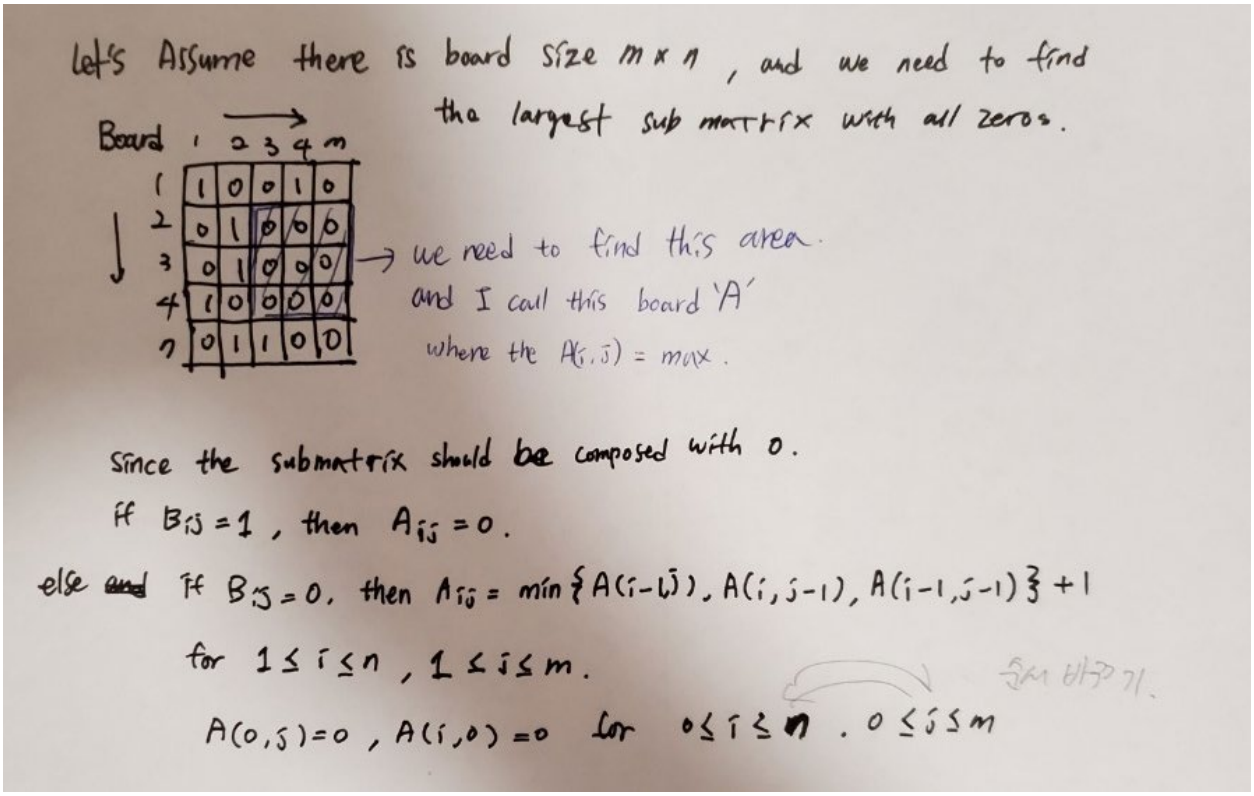
1	8	36	120	330	792	1716	3432
1	7	28	84	210	462	924	1716
1	6	21	56	126	252	462	792
1	5	15	35	70	126	210	330
1	4	10	20	35	56	84	120
1	3	6	10	15	21	28	36
1	2	3	4	5	6	7	8
1	1	1	1	1	1	1	1

Start

b. There are total 14 ways to move the rook: 7 ways to move horizontally and 7 ways to vertically in order to reach to the upper right corner. And either move up or move right, there are 7 distinct ways to be chosen. So, Binomial coefficient/combinations formula, it is C(14,7).

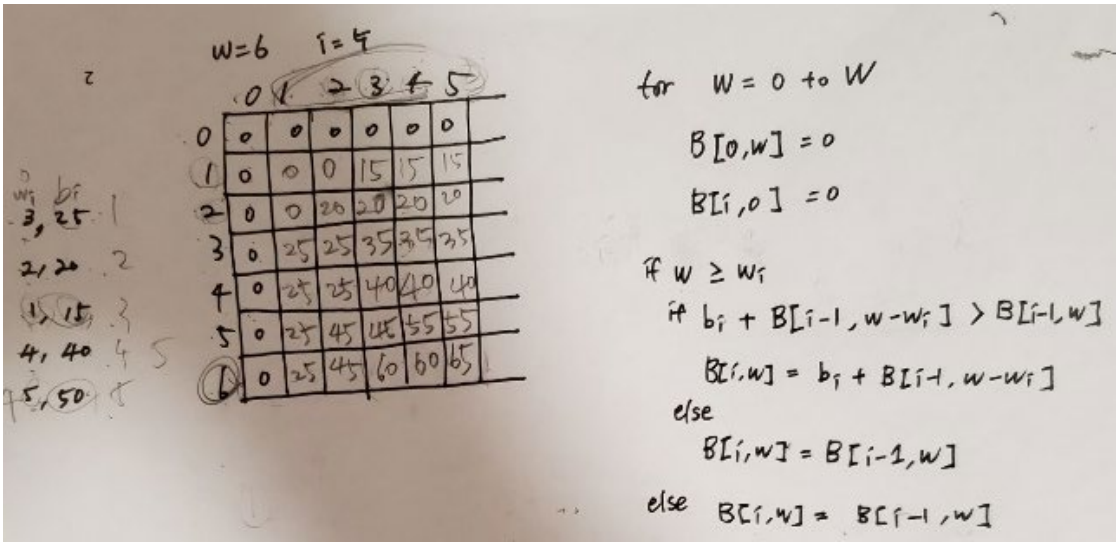
It equal to 14!/7!(14-7)! = 13 * 11 * 3 * 8 = 3432

Q3.



Q4

a.



b. There are one optimal subset because during the optimal process with the given values, I have only one maximal value at B[5,6] which is {item 3 and item 5}

c. If we can find more than one maximal values of a B[I, w], we can say there are more than one optimal subset for the knapsack problem. In order to see whether there is another subset or not, we can compare the B[I, w] with value between B[i-1, w] and [i-1, w-wi].

d.

code will be submitted on TEACH

e.

The bottom up dynamic programming is not the recursive function. And, to get the optimal value of the given data. I have created the table, and the table is 2d array and it is filled by the two for loop. The outer loop is running as much as the size of item + 1(because I need to add one more data cell for 0 values) and the inner loop is running as much as size of Weight + 1 (as like size of item, I need to add one more data cell for 0 values) **Therefore, time complexity is O((n+1)*(W+1)) and it equals to θ(nW)**

Space complexity is the total space taken by the algorithm with respect to the input size. As I mentioned above, I created table to save the computed number into the table. And the 2d array is size of input of Weight + 1 and size of item + 1. Therefore, **space complexity is O((n+1)*(W+1)) and it equals to θ(nW)**

Single for loop to check up the table and find the subset that compute the optimal value. And, since both checking single cell and getting the value takes constant time which is O(1), **therefore it takes O(n) to find the subset.**

Extra Credit

Code will be submitted on Teach