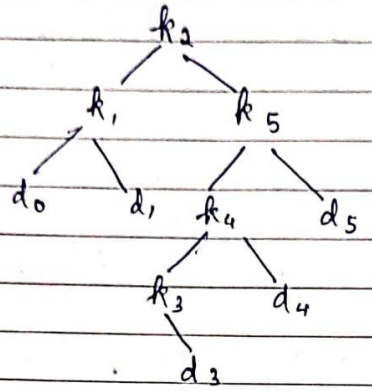
ASSIGNMENT-5

1. For Optimal Binary search tree example, give the pseudo code to constructing the optimal binary search tree from the root table (see text book page 403 exercise 15.5-1 for output format.

→



Pseudo-code:

```
optimal_binary_search_tree(root, i, j, last=0)
if i < j:
    return
if last == 0:
    print root[i,j] + "is the root"
else if j < last:
    print root[i,j] + "is the left child of" + last
else
    print root[i,j] + "is the right child of" + last
optimal_binary_search_tree(root, i, root[i,j] - 1, root[i,j])
optimal_binary_search_tree(root, root[i,j] + 1, j, root[i,j])
```

2.1 (Leetcode question 322. Coin Change)

a. Analyze subproblem structure and give the recursive relationship between problem and its subproblems.

→ We need to find the solution for its subproblems and then we can use it for finding solution of the problem.

eg. for recursive relation, if we have $y = [1, 2, 5]$, amount = 11, then we have to choose the smallest : $1 + \min(10), 1 + \min(7)$. Here we are considering sum of $5+5$ and $5+2$. So we can use it for finding the solution of the problem.

For recursive relationship, the optimal solution will be $A[j] = 1 + \min_{1 \le i \le k} (A[y^\circ - b_i^\circ])$ assuming $y^\circ \ge 1$ where $A[j]$ is the minimum numbers of coins and y° is the value the we need to make, b_i° will be the coin value.

b. Does the problem have overlapping subproblems ? Explain.

→ Yes. The same subproblems are involved multiple times for finding the solution. Let's say that we picked a coin with value x and x will be the solution. A_{n-x} will be the value to be changed and it will be the min. number of coins. So, A_n will be $1 + A_{n-x}$ here, the value of x is unknown. So we have to try it for every value of x that is possible so that we can select the min. value out of all those. So, we can say that there are overlapping subproblems.

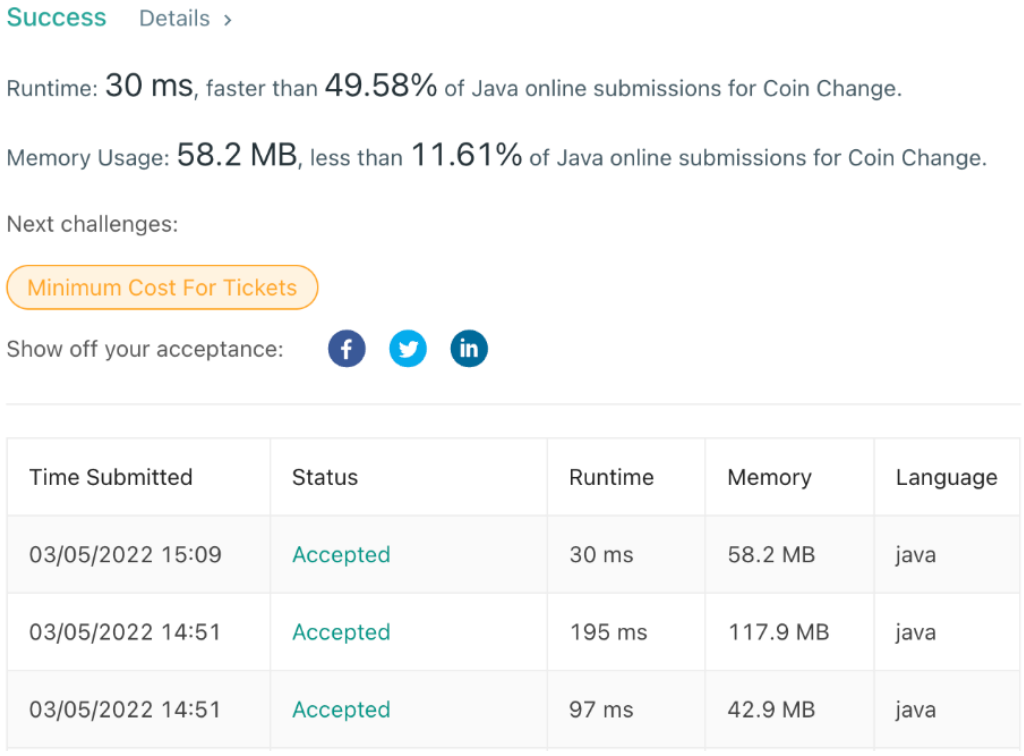
c. Design and implement a DP algorithm and create a successful submission in leetcode.com.

→ In document ⟶

Solution:

```
class Solution {
    public int coinChange(int[] coins, int amount) {
        int n = coins.length;
        int[][] dp = new int [n+1][amount+1];

        for(int i = 0; i <= n; i++){
            dp[i][0] = 0;
        }
        for(int i = 0; i <= amount ; i++){
            dp[0][i] = Integer.MAX_VALUE - 1;
        }

        for(int i = 1; i <= n; i++){
            for(int j = 1; j <= amount; j++){
                if(coins[i-1] <= j){
                    dp[i][j] = Math.min(dp[i-1][j], 1+dp[i][j - coins[i-1]]);
                }
                else
                    dp[i][j] = dp[i-1][j];
            }
        }
        return dp[n][amount] != Integer.MAX_VALUE - 1 ? dp[n][amount] : -1;
    }
}
```

Screenshot:



2.1 d) If you are using bottom up algorithm, then give an example to fill a bottom up tabular form.

→ eg. coins = [1, 2, 5] and amount = 11

	0	1	2	3	4	5	6	7	8	9	10	11
1	0	1	2	3	4	5	6	7	8	9	10	11
2	0	1	1	2	2	3	3	4	4	5	5	6
5	0	1	1	2	2	1	2	2	3	3	2	3

We will require 3 coins to make up the amount.

2.2 (Leetcode question 64. Minimum Path Sum)

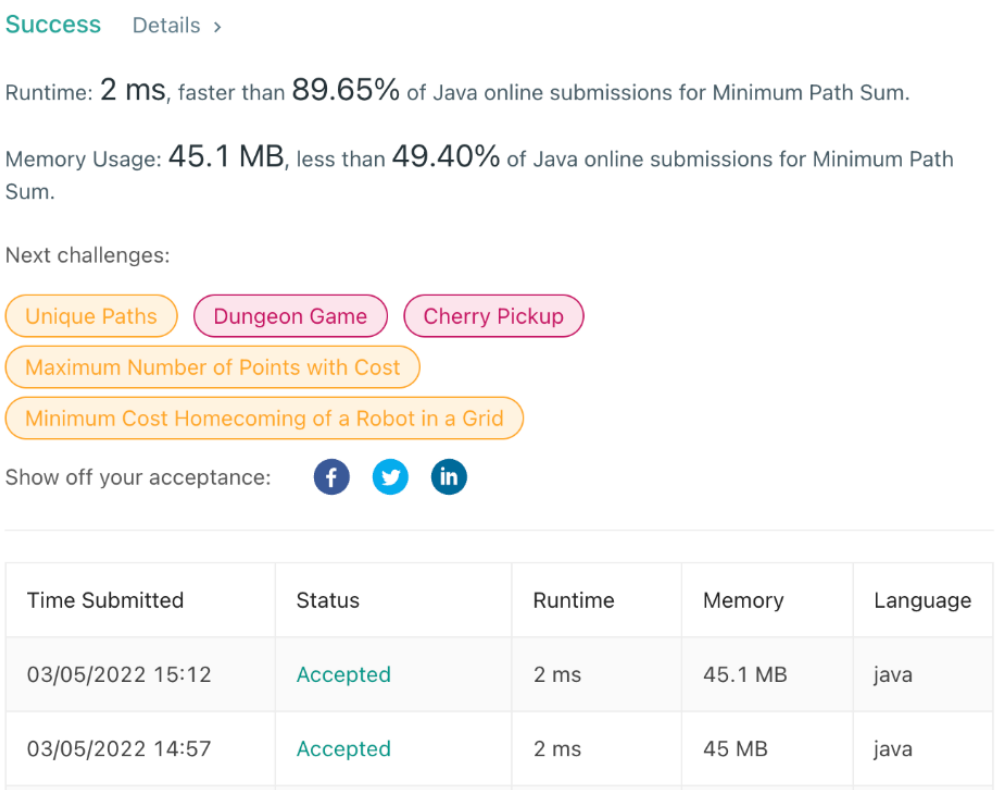
a) The subproblems here are to find the sum which comes from left to right or top to bottom as per the given constraints. For the m*n grid, the first row only moves to the right and the first column only moves downwards. For each grid blocks, the summation is done. The min. value sum form top and left in addition with the current cost goes to the table. So, the recursion takes place here. For each grid positions we have the problems and subproblems.

b) Yes. Here, the same subproblems are occuring multiple times as we have to calculate their sums continuously to find the min. sum path. The sum which have been calculated previously, have been used for multiple times to get the solution. So we can say that this problem has the overlapping problems.

Solution:

```
class Solution {
    public int minPathSum(int[][] grid) {
        int m = grid.length; int n = grid[0].length;

        int[][] dp = new int[m][n];
        dp[0][0] = grid[0][0];

        for (int i = 1; i < m; i++) {
            dp[i][0] = grid[i][0] + dp[i-1][0];
        }

        for (int i = 1; i < n; i++) {
            dp[0][i] = grid[0][i] + dp[0][i-1];
        }

        for (int i = 1; i < m; i++) {
            for (int j = 1; j < n; j++) {
                dp[i][j] = grid[i][j] + Math.min(dp[i-1][j], dp[i][j-1]);
            }
        }
        return dp[m-1][n-1];
    }
}
```

Screenshot:



Time Submitted	Status	Runtime	Memory	Language
03/05/2022 15:12	Accepted	2 ms	45.1 MB	java
03/05/2022 14:57	Accepted	2 ms	45 MB	java

2.2 d) eg. 3*3 grid is given below:

1	3	1
1	5	1
4	2	1

Then, the bottom up tabular form will be:

1	4	5
2	7	6
7	9	7

∴ The min. path sum will be 7.

2.3 (Leetcode question 416. Partition Equal Subset Sum)

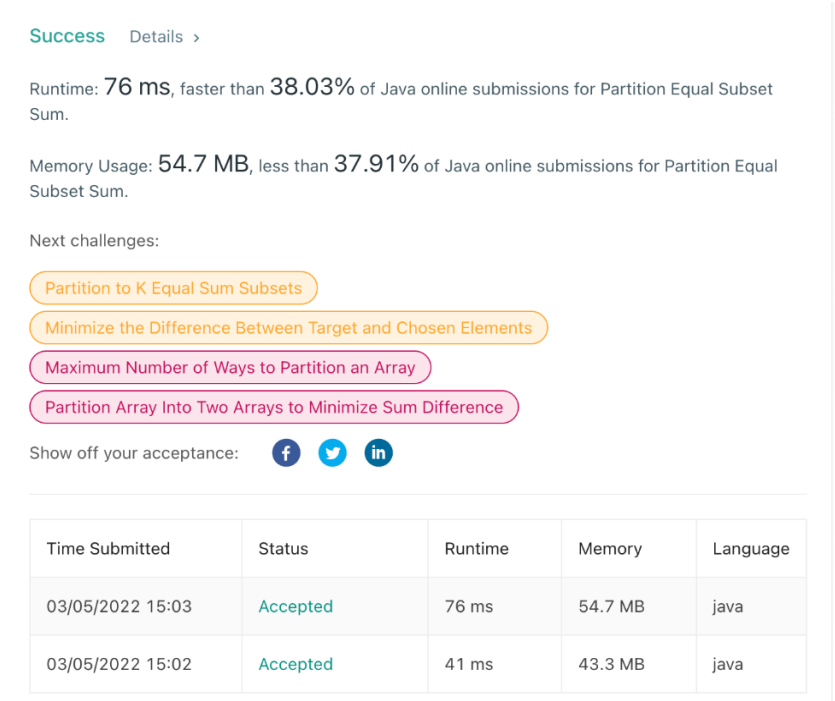
a) If sum of the array is odd, then there cannot be an array of two subsets who are having equal sum. so this will be false. If the sum of the array elements are even, then we need to find the sum divided by 2 and find the subset of that array having the sum which will be equal to the sum divided by 2. The subproblems there will be to find if the subsets are having the equal sum. so, the recursive relationship would be the sum considering the last element and reducing it by 1. Also, we ~~can~~ have to check without considering last element sum. If true, then we will have a solution.

b) Yes, Here we are finding the sum of the subarrays which we will use later for the finding the sum including the next element from the array. so, we can say the problem has overlapping subproblems.

Solution:

```
class Solution {
    public boolean canPartition(int[] nums) {
        int sum = 0;
        int n = nums.length;
        for(int i : nums)
            sum+=i;
        if(sum%2!=0)
            return false;
        sum /= 2;
        boolean dp[][] = new boolean[n+1][sum+1];
        for(int i=0;i<=n;i++){
            for(int j=0;j<=sum;j++){
                if(i==0 || j==0)
                    dp[i][j] = false;
                else if(nums[i-1] > j)
                    dp[i][j] = dp[i-1][j];
                else if(nums[i-1]==j)
                    dp[i][j] = true;
                else
                    dp[i][j] = dp[i-1][j] || dp[i-1][j-nums[i-1]];
            }
        }
        return dp[n][sum];
    }
}
```

Screenshot:



2-3 d) eg. it is given that array A = [3, 1, 1, 2, 2, 1]

	empty array	{3}	{3,1}	{3,1,1}	{3,1,1,2}	{3,1,1,2,2}	{3,1,1,2,2,1}
0	T	T	T	T	T	T	T
1	F	F	T	T	T	T	T
2	F	F	F	T	T	T	T
3	F	T	T	T	T	T	T
4	F	F	T	T	T	T	T
5	F	F	F	T	T	T	T

2-4 (Leetcode question 55. Jump game)

a) Here, we have an array of non negative integers. For the bottom up approach we need to consider the first approachable index or left most index. Here, we update the good index when the sum of the current index and number is greater than or equal to the index. The subproblems here are to make a jump from starting index to value of that index value.

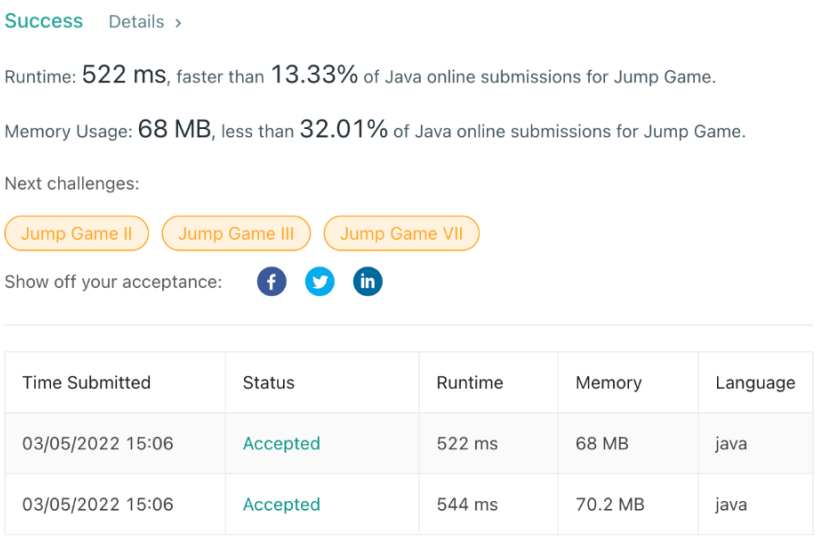
b) Yes. Here, the subproblems are needed to be looked upon for finding the solution to the jump game problem. The overlapping subproblems would help find if the jumps that are made are reaching to the last index of the array. So we can say that this problem has overlapping subproblems.

d) eg. we have an array [3, 2, 1, 0, 4]
The starting index 0 has the value 3 so that jump would be made to value 0, which is at index 3. It has the value 0 so it will not make a jump. So it will be false.

Solution:

```
class Solution {
    public boolean canJump(int[] nums) {
        int[] dp = new int[nums.length];
        Arrays.fill(dp, -1);
        return canJump(nums, 0, dp);
    }

    public boolean canJump(int[] nums, int index, int[] dp) {
        if (index >= nums.length-1) {
            return true;
        }
        if (nums[index] == 0) {
            dp[index] = 0;
            return false;
        }
        if (dp[index] != -1){
            return dp[index] == 1;
        }
        int jumps = nums[index];
        for (int i = 1; i <= jumps; i++) {
            if (canJump(nums, index+i, dp)) {
                dp[index] = 1;
                return true;
            }
        }
        dp[index] = 0;
        return false;
    }
}
```

Screenshot:



2.5 Given an integer representing money amount, one problem is to use minimum number of coins (with given values) to make up this value (assuming there is unlimited number of coins eg. given $14, and coin system of $1, 5, 10, 15, 20 $ it can be changed into $10, 1, 1, 1, 1$ (one 10 and four 1s). Do you think if this problem can be solved by greedy algorithms? If not, give a counter example.

→ Yes, it can be solved by greedy algorithm.

It goes as follows

① Sort array of coins in descending order.
② Initialize result as 0.
③ Find the largest coin that is smaller than
④ Add that coin to the result and subtract value of coin found from the amount
⑤ If amount = 0 then print the result
⑥ Else repeat steps 3 and 4 for the new value of the total sum of the amount.

In the example given

① Sorted array = {20, 15, 10, 5, 1}
② Initialize result = 0
③ Largest coin is 10 which is smaller than 14. So add 10 to result. Subtract 10 from 14. amount is 4.
④ Largest coin is 1 which is smaller than 4. So add 1 to result. Subtract 1 from 4. amount is 3.
⑤ Largest coin is 1 which is smaller than 3. So add 1 to result. Subtract 1 from 3. amount is 2.
⑥ Largest coin is 1 which is smaller than 2. So add 1 to result. Subtract 1 from 2. amount is 1.
⑦ Largest coin is 1 which is smaller to 1. So add 1 to result. Subtract 1 from 1. amount is 0.
⑧ Hence amount becomes 0. Print the result

So, the output will be $10, 1, 1, 1, 1$

2.6 Given a time table of railway trains, design an algorithm to find the min. number of platforms so that all the trains can be accomodated.

eg.

Train	Arrival	Departure
A	9am	9:30am
B	9:15am	1.0pm
C	10:30 am	11:00pm
D	10:45am	11:45am

→
```
int find_no_platform (int arr[], int depart[], int n)
    sort (arr, arr + n)
    sort (depart, depart + n)
    int need_platform = 1, result_platform = 1;
    int i = 1, j = 0;
    while (i < n && j < n)
        if (arr[i] ≤ depart[j])
            need_platform = need_platform + 1;
            i++;
        // update result_platform if needed_platform
        is greater than result_platform
        if (needed_platform > result_platform)
            result_platform = needed_platform
    else
        need_platform = need_platform - 1;
        j++;
    return result_platform;
```

In the example :-

Sorted order of arr [] and depart []

arr []	9:00	9:15	10:30	10:45
dep. art []	9:30	11:00	11:45	13:00

Initially, needs platform = 1, result_platform = 1

① For i at arr [1] = 9:15 and j at depart [0] = 9:30, j > i, increment i to 10:30 and increment result_platform which gives result_platform = 1+1 = 2

* need_platform < result platform so increment need_platform by 1 which gives 1+1 = 2

② For i at arr [2] = 10:30 and j at depart [0] = 9:30, depart [0] < arr [2] so 1 train will depart. Increment depart [0] by 1 and decrement result_platform by 1 which gives result_platform = 2 - 1 = 1

need_platform > result_platform, so need_platform remains unchanged which is need_platform = 2

③ For i at arr [2] = 10:30 and j at depart [1] = 11:00, arr [2] < depart [1] Increment arr [2] by 1. Increment result_platform by 1 which gives result_platform = 1+1 = 2.

need_platform = result_platform., so need_platform remains unchanged which is need_platform = 2

④ For i at arr [3] = 10:45 and j at depart [1] = 11:00. arr [3] < depart [1]. Increment result_platform by 1 which gives 2+1 = 3. need_platform < result_platform so increment need_platform by 1 which gives 2+1 = 3

All the arr [] have been traversed.

The algorithm returns result_platform which is 3.