171044010
Dilara Karataş

CSE 321 - HW5

QUESTION 1:

A better approach will be using Dynamic Programming in polynomial time complexity. We create a boolean 2D table subset [] [] and fill it in bottom up manner. The value of subset [i] [j] will be true if there is a subset of set [0...j-1] with sum equal to i, otherwise false. Finally, we return subset [0] [0]. Let's suppose sum of all the elements we have selected upto index "i-1" is "S". So, starting from index "i", we have to find a subset with sum closest to "-S".

Now, we can include "i" in the current sum or leave it. Thus we, have two possible paths to take. If we include "i" current sum will be updated as S+arr[i] and we will solve for index "i+1" i.e. dp[i+1] [S+arr[i]] else we will solve for index "i+1" directly. Thus, the required recurrence relation will be.

$$dp[i][s] = RetClose(arr[i] + dp[i][s+arr[i]], dp[i+1][s], -s);$$

$O(N * S)$, where N is the number of elements in the array and S is the sum of all the numbers in the array.

QUESTION 2:

Let's consider the conditions for using DP to find an efficient solution:

- **Overlapping Sub-problems:** Yes, we can see with our memorization algorithm that there are overlapping sub-problems.
- **Optimal Substructure:** Yes. At each node in the call-tree, we are making the decision to go down the left or right path. The decision is based on the path sums for each sub-tree for which we always select the smaller one.

The DP Table

We need to store the sum-path for a given sub-tree as the value in the table. Since the inputs to path sum function $f(m,n) = dp[m,n]$ takes the row number and index, they make good candidates for the table row and column. We will define function $v(m,n)$ to be the value for the number at row m index n.

CamScanner ile tarandı

Using this information we get following DP table.

m/n	0	1	2	3
0	14	-	-	-
1	12	14	-	-
2	7	10	13	-
3	8	6	9	6

The bottom row(4) is the base case where path sums for leaf nodes are the values themselves. We'll work our way from the bottom row (which represents the bottom or triangle) to the top.

Fill Row 3 (1, 4, 7)

dp [2][0] : is min (dp [3] [0], dp [3] [1]) + v (2,0) = min (8,6) + 1 = 7,

dp [2] [1] : is min (dp [3][1], dp [3][2]) + v (2,1) = min (6,9) + 4 = 10,

dp [2] [2] : is min (dp [3][2], dp [3][3]) + v (2,2) = min (9,6) + 7 = 13,

Fill Row 2 (5,4)

dp [1][0] : is min (dp [2][0], dp [2][1]) + v (1,0) = min (7,10) + 5 = 12

dp [1][1] : is min (dp [2][1], dp [2] [2]) + v (1,1) = min (10,13) + 4 = 14

Fill Row 1 (2)

dp [0][0] : is min (dp [1,0], dp [1] [1]) + v (0,0) = min (12,14) + 2 = 14

The Complexity

The DP solution runs O(n) where n is the number of elements. The table consusmes O(n*m) but that can be reduced to just m if you use linked list. This understanding of linked list to reduce space complexity is important.

CamScanner ile tarandı

QUESTION 3:

In the Dynamic Programming we will work considering this cases: The item is included in the optimal subset and the item is not included in the optimal set.

In a DP[][] table let's consider all the possible weights from "1" to "W" as the columns and weights that can be kept as the rows.

The state DP[i][j] will denote max value "j-weight" considering all values from "1 to ith". So if we consider "wi" (weight in ith) we can fill it in all columns which have "weight values > wi". Now two possibilities can take place:

- fill "wi" in the given column
- Do not fill "wi" in the given column.

Now we have two take a max of these possibilities, formally if we do not fill "ith" weight in "jth" column then DP[i][j] state will be same as DP[i-1][j] but if we fill the weight, DP[i][j] will be equal to the value of "wi"+ value of the column weighting "j-wi" in the previous row. So we take the max of these two possibilities to fill the current state.

v	w		0	1	2	3	4	5	6	7	8	9
		0	0	0	0	0	0	0	0	0	0	0
3	2	1	0	0	2	2	2	2	2	2	2	2
4	4	2	0	0	2	2	4	4	6	6	6	6
10	5	3	0	0	2	2	4	5	6	7	7	9

Complexity: $O(N*W)$, where 'N' is the number of weight element and 'W' is capacity. As for every weight element we traverse through all weight capacities $1 \le w \le W$.

CamScanner ile tarandı