Gebze Technical University
Computer Engineering

CSE 321
Introduction to Algorithm
2020-2021 Fall

HOMEWORK 5 REPORT

Yusuf Can Kan
161044007

1 Question 1

In this question we need to implement a algorithm that finds subsets with has sum of 0. While doing this we need to use dynamic programming.

I implemented my algorithm such a way that, it stores every calculated subsets in an list and also it stores the sum of this subset array. In whenever it needs to use any subset sum it takes if from subset memory.

My algorithm first starts to iterate every element of the array one by one and starts to calculate subsets.

```
def printsubsets(l):
    subset_memory=[] #stores previous calculated subsets
    print("Subsets which has 0 sum are;")
    for i in l:
        findZero(i,subset_memory)
```

In order to create subset I created a differen function called **findZero**.

```
def findZero(elem,l): #memory of subsets
    for i in range(0,len(l)):
        subset=[(l[i][0])+elem]
        subset.extend(l[i][1:])
        subset.extend([elem])
        if subset[0] == 0:
            print(subset)
        l.extend([subset])
    if(elem==0):
        print([elem])
    l.extend([[elem,elem]])
```

This subzero function takes one element and memory_list. It iterates all the subsets in memory list(in this way it does not calculate the same subsets again) and adds creates copy of this subsets with adding element every one of it. In this way it calculates the subsets with adding itself and stores it to the dp memory. While making this process it does not calculates the sum of subset elements because memory already has this sum. It just uses this.

My subset memory array has this form;

[[sumofsubset1, subset1], [sumofsubset2, subset2]... [sumofsubsetn, subsetn]]

As you can see it stores every calculated subsets in this array and whenever it needs to create new subsets for new element it use this array.

For example;
Lets say we have list [1,2,-2]

- In first iteration Findzero function first takes 1 and empty memory list (**subset_memory**).
  Since memory list is empty it creates first sublist with input. [1,1]
  First one is sum of all the elements in this sublist and second 1 is subset element.
  So our memory list is [[1,1]].

- In second iteration we feed findZero function with 2 and [[1,1]].
  It copies every element of sublist and adds 2 to them. So memory list is; [[1,1].[3,1,2],[2,2]]

- In third iteration it adds the -2 to the sublist with using previos calculated sublists;
  [[1,1].[3,1,2],[2,2],[-1,1,-2],[1,1,2,-2],[0,2,-2],[-2,-2]]]
  In this stage also while its calculating the [0,2,-2] it prints the screen.

Since in every case this algorithm iterates all the array and creates all the subsets (program does this because in homework pdf, all the subsets requested.) worst, best and average case is all the same.

In first function it iterates all the array one by one: n
In findZero function, every time function came it iterates all the subsets. Lets say m

So the complexity of this algorithm is O(n*m).

Tests;

```
Input List;
[2, 3, -5, -8, 6, -1]
The subsets that has 0 sum are;
[2, 3, -5]
[2, -8, 6]
[-5, 6, -1]
[3, -8, 6, -1]
```

```
Input List;
[1, 2, 3, -1, -2, -3]
The subsets that has 0 sum are;
[1, -1]
[2, -2]
[1, 2, -1, -2]
[3, -1, -2]
[1, 2, -3]
[3, -3]
[1, 3, -1, -3]
[2, 3, -2, -3]
[1, 2, 3, -1, -2, -3]
```

2 Question 2

In this part of the homework we need to find a way that finding the minimum sum way in the triangle.

In this question I follow a bottom to top aproach. While goint top, I used a memorization array (paths) which stores the previous minimum roads in it. For example;

```
  1
 2 3
4 5 6
```

First, array fills with bottom part. (4,5,6)

Then algorith takes the one upper row of the triangle which is 2 and 3.

For 2, we can use the path from 4 and 5. We choose 4 becase it is smaller than 5. So 4+2=6.
For 3, we can use the path from 5 and 6. We choose 5. 3+5=8

So our current results is;

```
 1
6 8
```

Since this 6 and 8 stored in an array, program does not calculates this paths again. With this sum values, memory array also stors the paths which it passed. So for 6 array also stores [4,2] and for 8 program also stores [3,5].

In final program comapres 6 and 8 and choses 6. It adds 1 to this path and obtains final path.

Path cost(SUM): 7
Path: 1 2 4

Complexity Analysis;

In this question the basic operation is comparison. All out path choices depends on comparisons between 2 path costs.

In every line of triangle we make x-1 (x represents the line number) comparison. If we assume we have total n element in triangle, in total we will make n – count of lines comparisons. Since count of lines is a constant number, our complexity with respect to comparison operation is O(n).

Tests:

```
Input Array = [
        [         1 ],
        [       5, 1 ],
        [      5, 3, 4 ],
        [    5, 4 ,1, 8],
        [  1, 4, 7, 1, 11 ],
```

[12, 8, 6, 9, 1, 5]]

Output

```
yck@ubuntu:~/Desktop/ALGO/161044072 (9)$ python3 question2.py
Sum Path:8
The Path is:[1, 1, 3, 1, 1, 1]
yck@ubuntu:~/Desktop/ALGO/161044072 (9)$
```

:

Input Array = [
[2],
[5, 4],
[1, 4, 7],
[8, 6, 9 ,6]]

Output:

```
yck@ubuntu:~/Desktop/ALGO/161044072 (9)$ python3 question2.py
Sum Path:14
The Path is:[2, 5, 1, 6]
```

3 Question 3

In this question we tried to solve knapsack problem with using dynamic programming. In my program I basically use normal solution with usin 2d array. 2d array has size of i.W which i is the input element count and W is the capacity.

In every iteration program stores the current result in memory array. In this way if some of the calculations will be needed in the future it can be basically taken from this array.

For this at the beginning of the program my function controls the array instances for if the calculations has been done before and assigned to the memory.

```
def knapsackProblem(i,W,weights,values,memory):
    if memory[i][W] != 0:
        return memory[i][W]
```

If it is not this means that it will calculated. So for this function checks 3 situation. If there is any element left for calculations (since this is recursive function this is the base case) or is capacity is full. If capatiy is full we don't need to calculate any other calculations.

```
    if i==-1 or W==0: #if capacity is full or there is no element left
        returnValue=[0,[]]
```

After this function checks if current elements weight is bigger than capacity. If this is the case we will ignore this element and calculate rest of it recursively.

```
    elif weights[i]>W: #if weight of single element is bigger than current
                       # capacity, ignore it
        returnValue=knapsackProblem(i-1,W,weights,values,memory)
```

In the end we have 2 recursive call. First is take into consideration current element and other is not. We will calculate both recursively and compare results.

At the end we will update the memory for future calculations and return the result.

```
else:
    withoutCurrent=knapsackProblem(i-1,W,weights,values,memory)

    withCurrent=knapsackProblem(i-1,W-weight[i],weights,values,memory)
    withCurrent[1].append([weights[i],values[i],i])
    withCurrent[0]=withCurrent[0]+values[i]
    if withoutCurrent[0] > withCurrent[0]:
        returnValue=withoutCurrent
    else:
        returnValue=withCurrent

memory[i][W]=returnValue
return returnValue
```

My function starts to calculate the array from end to beginning. So first last element will be calculated.

Complexity Analysis;

The function makes calculation i.W(i is the index count and W is the capacity) times in worst case. After i.W times the memory array will be full and it does not need to be any calculations anymore.

In else case of the function it makes 2 function calls. So we can see that we have O(2*i*w) complexiy.

So the final complexity is O(i*W) which i is input count(items count), W is capacity.

```
yck@ubuntu:~/Desktop/ALGO/161044072 (9)$ python3 question3.py
The input is:
Values: [10, 4, 3]
Weights: [5, 4, 2]
Capacity: 9

The output is:
Total value: 14
Element 0 Value is: 10 weight is: 5
Element 1 Value is: 4 weight is: 4
yck@ubuntu:~/Desktop/ALGO/161044072 (9)$
```