CSE 321
Introduction to Algorithm Design

Homework #5
Report

Oğuzhon Agkuş
161044003

① I've designed a dynamic programming algorithm to find best plan with minumum cost.

My algorithm creates a cost table according to operation costs and moving cost.

Example:

	1	2	3	4
SF	1	3	20	30
NY	50	30	2	4

operation costs for 4 month

moving cost = 10

Cost Table

	1	2	3	4
SF	1	4 (63)	24 (71)	34 (34)
NY	50	41 (60)	16 (43)	20 (20)

→

	1	2	3	4
SF	1	4	24	34
NY	50	41	16	20

- Each item represents the minimum total cost until the month.

SF row = min (cost of staying in SF, cost of moving from NY)

NY row = min (cost of staying in NY, cost of moving from SF)

Example: SF in month 2 → 1+3 = 4

50+3+10 = 63

After creating the cost table we can see the best location for end. Now we can trace back one by one. Finally we will find the path.

Example (continued):

	1	2	3	4
SF	1	4	24	34
NY	50	41	16	20

↓ Minimum end

20 - 16 = 4 → stay in NY

20 - 24 - 10 = -14

16 - 4 - 10 = 2 → move to SF

16 - 41 = -25

4 - 1 = 3 → stay in SF

4 - 50 - 10 = -56

Find maximum difference between i and i-1 th months. → Path: NY, NY, SF, SF

Reverse

SF, SF, NY, NY

Time complexity is O(n), where n is month count. Algorithm has 3 loops. Each of them traverse all months.

$$O(n) + O(n) + O(n) = O(n)$$

② For this problem I defined two classes: Session and Symposium. Session objects have ID, start and end time data.

Symposium objects have a list of Session objects and some basic methods. My approach is firstly ordering the session in increasing order according to end times. I used quick sort algorithm that I've implemented in HW#3. I just modified it to sort Session objects.

Defaultly it selects the first session. Then if the next session starts just after the current session, select it. Traverse all sessions like that.

Example:

ID	Start	End
1	1	3
2	5	9
3	11	12
4	0	7
5	3	4
6	8	10
7	2	5

Sort →

ID	Start	End *
1	1	3
5	3	4
7	2	5
4	0	7
2	5	9
6	8	10
3	11	12

Optimal sessions → 1, 5, 2, 3

Quick sort have O(nlogn) average case and $O(n^2)$ worst case complexity.

Traversing all sessions to select have O(n) complexity.

If sessions are ordered already we don't need to sort them. If it is ordered, algorithm complexity will be O(n).

If sessions are not ordered, we have to sort sessions. So algorithm complexity will be $O(n^2) + O(n) = O(n^2)$

↓ quick sort worst case complexity

(3)

I designed a dynamic programming algorithm to find subsets whose elements sum equal to zero. Also this algorithm find the subset count satisfy the rule, I have a global list to store the subsets. If the algorithm finds a subset except the empty set, print the first subset it find.

```
def find_subsets(array, n, index = 0, sum = 0,  subset = [], dp = []):
    if not(len(dp)):
        dp = [[0 for i in range(200)] for j in range(n)]

    if(index == n):
        if(sum == 0):
            subsets.append(subset)
            return 1

        else:
            return 0

    s_1 = subset.copy()
    s_2 = subset.copy()
    s_1.append(array[index])

    temp_1 = find_subsets(array, n, index+1, sum + array[index], s_1, dp)
    temp_2 = find_subsets(array, n, index+1, sum, s_2, dp)

    dp[index][sum + n] = temp_1 + temp_2
    subset_count = dp[index][sum + n]

    return subset_count
```

create dynamic programming table

if sum of subsets elements equals to zero return 1 else return 0

copy the previous subset add next element to the first one.

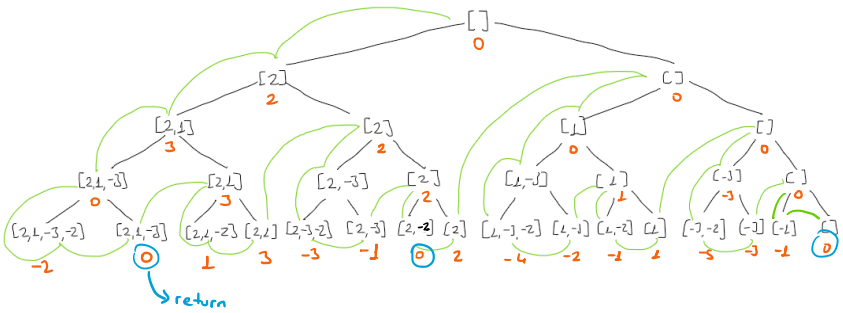
recursive calls

subset count and update the dp table

Time complexity = O(n.S) where n is array size
S is sum of all elements.

Example:

array → [2, 1, -3, -2]



(4)

I designed a dynamic programming algorithm but I need do some small changes on the question. The problem is like that: You want to minumum penalty but mismatch and gap have negative points. So we can fill with not matchig characters. I multiplied the penalty points with -1. Now we can find the minumum penalty. But at the end I re-calculate the minumum penalty with respect to values you give. Also I suppose that strings should be have some length. Firstly I build a table.

Example:

	-	A	G	G	C	A
-	0	2	4	6	8	10
A	2	-2	-1	0	1	6
G	4	-1	-4	-3	-2	-1
G	6	0	-3	-6	-5	-4
G	8	1	-2	-5	-4	-3
C	10	2	-1	-4	-3	-6
T	12	3	0	-3	-6	-5

minimum penalty

A - G G C A
A G G G C T

4x2 - 2 - 1 = 5
(match, mismath, gap)

-5 x -1 = 5
(Shortly recalc.)

Time complexity: O(m.n) where m is length of string-1
n is length of string-2

→ $O((\max(m,n))^2)$

(5) I designed a greedy algorithm. I observed that firstly we should add smallest numbers. So I need to sort my array. I can use quick sort algorithm but I have used it in previous question, also I modified it. Instead of quick sort I used built-in sort method of Python list class. It is an implementation of a sorting algorithm called Timsort. I derived merge and insertion sort. It has O(nlogn) complexity at worst case. Complexity of my algorithm is $O(n\log n) + O(n)$, ★
↓ Sorting ↓ Traversing array once

Example: Array = [1,5,7,3]

1,3,5,7 → 1+3 = 4, 4+5 = 9, 9+7 = 16 ⟩ 4+9+16 = 29

The algorithm does not find best solution always because it's a greedy algorithm.

Example: Array = [3,18,27,30,15]

3,15,18,27,30 → 3+15 = 18, 18+18 = 36, 36+27 = 63, 63+30 = 93 ⟩ 18+36+63+93 = 210

★ 3+15 = 18, 18+27 = 45, 18+30 = 48, 48+45 = 93 ⟩ 18+45+48+93 = 204 ★ Smaller ★