CSE 321 HOMEWORK 5
GOKHAN HAS – 161044067

Question 1:

The optimal plan either ends in NY or in SF If it ends in NY, it will pay N plus one of the following two quantities:
• The cost of the optimal plan on n - 1 months, ending in NY, or
• The cost of the optimal plan on n - 1 months, ending in SF plus a moving cost of M

An analogous observation holds if the optimal plan ends in SF. Thus, if OPTn (j) denotes the minimum cost of a plan on months 1...j ending in NY, and OPTs (j) denotes the minimum cost of a plan on months 1….j ending in SF, then

OPTn (n) = Nn + min(OPTn(n – 1), M + OPTs(n – 1))
OPTs (n) = Sn + min(OPTs(n - 1), M + OPTn(n- 1))

This can be translated directly into an algorithm:

OPT (0) - OPT(0) = 0
For = 1, …., n
    OPTn(i) = Ni + min(OPTx(i-1), M + OPTs (i - 1))
    OPTs(i) = Si + min(OPTs(i-1), M + OPTn(i - 1))
End Return the smaller of OPTn(n) and OPTs (n)

The algorithm has n iterations, and each takes constant time. Thus the running time is O(n)).

Question 2:

In the simultaneous sessions problem, we are given n simultaneous sessions numbered 0 to n – 1. Each activity i has a start time s_arr(i) and a finish time f_arr(i). Two simultaneous sessions i and j are mutually compatible if s_arr(i) >= f_arr(j) or s_arr(j) >= f_arr(i). The problem is to find a largest subset of simultaneous sessions that are mutually compatible.

1. Interval timing function is defined.

2. The two lists take stimulation and f_arr as arguments.

3. Warnings [i] is the start time of i activity.

4. f_arr [i] is the end time of activity i.

5. The algorithm first operates by ordering the simultaneous sessions according to the earliest completion times.

6. The activity with the earliest finish time is then included in the solution set.

7. All simultaneous sessions incompatible with the newly added activity will be removed.

8. The above two steps are repeated until there is no more activity left.

9. A set of solutions is returned, which is a maximum dimensional subset of mutually compatible simultaneous sessions.

In sort functions, it is in the nlogn the worst case, and n the for loop. Thus it is O(nlogn) in worst case.

Question 3:

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.

Let's define dp[i][S] first. It means sum of the subset of the subarray{i, N-1} of array 'arr' 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 4:

Given as an input two strings, str1 and str2 output the alignment of the strings, character by character, so that the net penalty is minimised. The penalties calculates as :

1. A penalty of gap occurs if a gap is inserted between the string.
2. A penalty of miss occurs for mis-matching the characters of str1 and str2.

We can use dynamic programming to solve this problem. The feasible solution is to introduce gaps into the strings, so as to equalise the lengths. Since it can be easily proved that the addition of extra gaps after equalising the lengths will only lead to increment of penalty.

Optimal Substructure

It can be observed from an optimal solution, for example from the given sample input, that the optimal solution narrows down to only three candidates.

1. str1 and str2.
2. str1 and gap.
3. gap and str2.

Proof of Optimal Substructure.

We can easily prove by contradiction. Let str1 – str1[i] be str1' and str2 – str2[i] be str2' . Suppose that the induced alignment of str1' , str2' has some penalty miss, and a competitor alignment has a penalty miss* , with miss* - miss.

Now, appending str1[i] and str2[i], we get an alignment with penalty* + miss < penalty + miss penalty. This contradicts the optimality of the original alignment of str1 and str2.

Hence, proved.

Let dp[i][j] be the penalty of the optimal alignment of str1[i] and str2[i]. Then, from the optimal substructure,

dp[i,j] = min (dp[i-1,j-1] + miss, dp[i-1,j] + miss, dp[i,j-1] + gap.

The total minimum penalty is thus, dp[m][n].

Reconstructing the solution To Reconstruct,

1. Trace back through the filled table, starting dp[m,n].

2. When (I,j)

    2a. if it was filled using case 1, go to (i-1,j-1).

    2b. if it was filled using case 2, go to (i-1,j).

    2c. if it was filled using case 3, go to (i,j-1).

3. if either i = 0 or j = 0, match the remaining substring with gaps.

    And the worst-case time complexity is O(n * m), n is the str1 length's and the m is the str2 length's.

Question 5:

In this question I get the array as a parameter. I break this array and assign it to another array. This is called newArr. Then I have to get the smallest element from the array. Because both the greedy algorithm is requested and the least number of operations is required. I'm deleting this element from the array. I then write a loop that continues until the size of this new array is zero. I continue to do the operations I just described in this loop. Adding these minimum values to the variable _sum. The important point is that the variable bu_sum is not the variable _sum that holds the sum of the elements of the array. I would return this value last.

Fx: is the input array is [1,2,3,4,5]

    1 + 2 = 3

    3 + 3 = 6

    6 + 4 = 10

    10 + 5 = 15

And the result is 15 + 10 + 6 + 3 = 34.

The worst-case complexity is O(n). Where n is array's length. I assume that the get of the min element of array is constant time.

Gökhan Has - 161044067