CSE 321
INTRODUCTION TO ALGORITHM DESIGN

HOMEWORK 05 REPORT

Ömer ÇEVİK

161044004

1 Question's Solution

In this part, there are two difference city New York (NY) and San Francisco (SF). The main idea of that problem is to visiting these cities on each month using dynamic programming but the month costs can be distinct. The cost of moving is given as M.

To finding the minimum cost of changing location on NY and SF cities is the our problem.

Algorithm is working on keeping two list costs of NY and SF initializing the first element as zero because of the summing the costs. Then creating a loop on length of SF or NY to find the minimum cost summing the costs of each cities. Then the each probability of minimum total cost kept in the last element of NY and SF cost lists.

The minimum total cost of location is between the NY and SF's last elements. Totally time complexity is O(n).

So $T_W(n)\ \epsilon\ \underline{O(n)}$

2 Question's Solution

In this part, there is an 2D array which represents the sessions start and end times. The sessions times can be unsorted. To apply the greedy algorithm it must be sorted. So before to starting greedy algorithm, sorting the sessions increasing order by end time. To sort that 2D array, **Bubble Sort** algorithm is implemented.

That sorted array is ready to find the optimal sessions. The first index of array must be in our sessions list. Because finish time is so close. Then starting the next index of that sessions 2D array checking the start time of 2D arrays elements is greater than previous indexed element's end time. If it is true, then we can say that that session is available to go and previous index is re-initialized to start time of current element. That job goes in a loop of sessions.

To evaluate worst case scenario, if the sessions array is unsorted then the bubble sort will be execute in worst way as n^2. Then the sessions loop goes O(n).

So $T_W(n)\ \epsilon\ \underline{O(n^2)}$

3 Question's Solution

In this part, there is a given list to function and the main idea is to finding all sub-lists of that given list using dynamic programming which is sum of elements equals to zero it stops.

In algorithm design, there is a loop which moves on given list and there is another loop which takes the each elements and appends the checked element to checked array. That checked array uses the list's elements only once for efficiency. Mapping the values to check already used to sum and finally if a sub-list's sum equals to zero then terminates the each loops and returns the sub-list which elements sum is zero.

Checking the combinations of elements in list with map gets O(2^n) time complexity in worst way. The sum of sub array costs O(logn). If not found the zero sum is the worst scenario.

So $T_W(n)$ ϵ $\underline{O(2^n logn)}$

4 Question's Solution

In this part, there are given two sequences as A and B. And checking the match of characters in these sequences. If match happens then evaluates. Otherwise unmatched evaluates. If at least one matching happened then the gap score is evaluated.

In that algorithm, for dynamic programming used a 2D list of python. 2D list is initialized using numpy library by zero. That 2D array is going to evaluate the index of matched and unmatched characters of sequences using similarly Smith-Waterman algorithm. The matched characters' index are saved in a list and the length of these list shows up the matched characters size.

Each row has its own values using maximum values of and matching characters. If the row is zero based then that character is mismatched and that is countered. To evaluate the gap, the indexes of matched characters are searched ordered by increasing one. If there is a space of increasing order by one then it will be counted as gap. Finally all the score show up to screen. Creating and evaluating these 2D table costs O(mn) and counting the gap costs O(logn). Consequently, mn + logn costs and it equals to O(mn).

So $T_W(n)$ ϵ $\underline{O(mn)}$

5 Question's Solution

In this part, user gives the array and greedy algorithm calculates the sum of that array. But firstly sorting the array to apply greedy algorithm. Then the maximum element of that array is the border of the loop. The loop is created to sum each value of array. While summing the array elements there is a counter to sum of operation size.

Sorting the array in Python3 is O(nlogn). The loop is created until maximum element is O(n). The complexity of programme is O(nlogn + n).

So $T_W(n)$ ϵ $\underline{O(nlogn)}$