CSE 321 Homework 4 Report

Sema Köse

141044002

Course Assistant : M. Burak Koca

EXPLANATIONS OF THE QUESTIONS

Question 1:

I. Algorithm:

Let C_i be the cost that when we arrived the hotel. If we think we're at the 0 point initially, C_0 will be 0. While respectively calculating the costs of the hotels, for the computation of all C_i's, we compute all possible stops until *i*'th stop. Let a_i be the current stop and a_k be the previous stops; $0 <= k < i$ satisfies.

So; while we computing the cost until reaching the end point, we compute the cost for each i'th stop as:

$$C_i = \min \left(C_k + \left(200 - (a_i - a_k)\right)^2 \right)$$

for all k values that satisfies $0 <= k < i$.

For n hotel, we compute the costs from 1 to n and store them into an array. For assigning the minimum value to C_i we use recursive for computation of temporary cost values.

II. Time Complexity:

For all C_i values, we make the computations in linear time so the algorithm takes $O(n^2)$ time.

$$T(n) \in O(n^2)$$

Question 2:

I. Algorithm:

We have s[1…n] for representing the sentence and d[1…n] for representing the letters in this sentence. For initial state, we label with -1 all of the letters. If s[j…i] is a valid word, we assign 1 to d[j] for represent the start of the valid word and assign 0 to d[k] for all k values which satisfies $j<k<i$. In other words, the first letter of the word is marked as 1 and remainings are marked as 0 including the last letter. For the subset of s[i…n], we call the method recursively and repeat the steps. End of the each call, we return an array which represents the valid word. We merge the arrays and we get the result array which includes 0's and 1's. If we have the -1 value in the array, it means this sentence includes a word or a letter that is not valid and it cannot be reconstructed successfully. If it doesn't have any -1, it can be reconstructed successfully.

II. Time Complexity:

It takes $O(n^2)$ time.

$$T(n) \in O(n^2)$$

Question 3:

I. Algorithm:

For the solution of this problem, we follow the rules exactly what we do in merge sort. We put the list of arrays in pieces. We construct a new array with merging 2 array. And then we repeat it until the end.

For example:

Let S = [

[1,5,7,8],

[2,6,9,11],

[3,10,12,16],

[4,13,14,15]]

be our array of k sorted arrays with n element.

We divide it and start to solve as sub problems.

merge S[1] and S[2] into new array

merge S[3] and S[4] into new array

merge the resulting arrays

repeat it until all arrays has been merged.

II. Time Complexity:

While we turn the k sorted arrays with n element into k/2 sorted arrays with 2n element, we spend O(kn) times. If we think that we follow the merge sort steps, to get an array with kn element we spend O(kn) time O(logk) times. So, the time complexity is O(knlogk).

$$T(n) \in O\,(knlogk)$$

Question 4:

I. Algorithm:

First, the invitees labeled from 1 to n in a list S. Then, we fill a list K representing number of people in S that the ith person know, and a list D representing number of people in S that the ith person doesn't know for invitees which represented as indexes.

For example;

K[2] = 5 => S[2] knows 5 people

D[2] = 7 => S[2] doesn't know 7 people

Then, we check in a loop if there exist invitees that satisfies the condition of D[i] > 5 and K[i] > 5. If the condition is satisfied, we append the people S[i] into new list that represents the best choice of party invitees and return this list.

II. Time Complexity:

We scan all of the list for check the condition of each possible invitee, so there is n iteration. We scan the lists that respresents knowledge situation, again here we have n iterations. So, the runnig time is $O(n^2)$.

$$T(n) \in O(n^2)$$

Question 5:

I. Algorithm:

First, we look at the equality constraints and assign the same values to variables according to this constraints. The variables respresented as a variables list. To each variable in variables list, we assign a list representing the situation between itself and other variables.

For example;

Variables[[inf, 1, inf, 0], // x_1

[inf, inf, 1, inf], // x_2

[inf, inf, inf, 1], // x_3

[inf, inf, inf, inf] // x_4]

x_1 [1] corresponds to relation between x_1 and x_2

x_1 [2] corresponds to relation between x_1 and x_3

x_1 [3] corresponds to relation between x_1 and x_4

If an element of variables list has an element with value of:

inf => the relation not known

1 => $x_i = x_j$

0 => $x_i \neq x_j$

In this example; $x_1 = x_2$, $x_1 \neq x_4$, $x_2 = x_3$, $x_3 = x_4$

We merge the set of the variables that equals to each other. For this, we can use Union-Find structure.

II. Time Complexity:

Union operation repeats n-1 times for n element and it takes O(nlogn) time to calculate. If we assume find operation repeats for each m constraints, it reapeats at most 2m times. So it takes O(m) time. So, the algorithm completes in O(m + nlogn) time.

$$T(n) \in O(m + nlogn)$$