Kalyan Ghosh(kghosh) Mohit Satarkar(mmsatar2)

CSC 505 Design and Analysis of Algorithms

Homework 1

Programming Problem: Longest Contiguous Subsequence of Unique Values

A. Worst case analysis for unique1 (Naïve Approach):

Sol:

* In the 1st naive approach to solve this problem, we apply a brute force method, where we are running two loops (one outer loop & one inner loop).

* In the outer loop, we are looping through each of the elements in the list

and in the inner loop, we are checking that what is the longest subsequence we can explore starting from that index of 'i'. Here also we check, that is the current element is present in the current subsequence which takes time proportional to n.

* So, in this case we can do an asymptotic time analysis as follows:

1. T(n) can be expressed as:

T(n) will be bounded by O(n^2), because:

In worst case T(n) will have to run for mentioned iterations:

T(n) = (n-1)*n + (n-2)*n + ...+ (n-(n-1))*n ----> (1)

In equation 1, the outer loop will run n-1 times for the (n-1) elements except the last element and the inner loop will also accordingly run for (n-1), (n-2).....,1 times and for each term, we check/compare if the current element is present in the current subsequence which takes time proportional to n

T(n) in worst case will lead to:

T(n) = [(n-1) + (n-2) + ...+ (1)]*n

= [n(n-1)/2]*n

= [(n^2-n)/2]*n

Therefore,

T(n) = O(n^3)

2. Now, we do time analysis on actual size of data and record the times:

Time taken for size 80000(input-6.txt file) = 6.35seconds

Note: For smaller values of n i.e. [10,20,100,1000], the time to run the code is insignificant

****

B. Worst case analysis for unique2 (Time Optimized Approach):

Sol:

* In the 1st naive approach to solve this problem, we apply a more optimized way to solve the problem by using additional data structure.

* This approach would require extra space but will give a better asymptotic time complexity.

* In this approach, we use the TreeSet data structure in Java to store the found nonduplicate elements for each iteration of the outer loop.

* Using the TreeSet, we see that for each element in in the outer loop, how many non-duplicate contiguous elements can be explored before finding a duplicate and we store the non-duplicate elements in the Treeset which is its property.

* Once we find a repeating number, we clear the TreeSet for that iteration, check if the length of the current subsequence is greater than the globally stored largest subsequence and continue with the next iteration.

* Now we do an analysis on how data is stored in TreeSet:

TreeSet provides guaranteed **log(n)** time cost for the basic operations (add, remove and contains).

* Now, we do Time Complexity Analysis & Space analysis of our improved version of the code

1. Time Analysis:

In this approach, we make a single pass through the list starting from the 1st element to (n-1) th element

Then at each iteration we, explore the longest possible contiguous subsequence without duplicates and keep adding elements in to the TreeSet till we find a duplicate

Note: A TreeSet is guaranteed to hold only unique elements.

So, each outer loop, we store values in TreeSet

Therefore,

T(n) in worst case will lead to:

T(n) = (logn) + (logn) +.....+(logn) --->1

[where each term is for the outer loop for n-1 elements and logn is the time to check if that elements exists in the TreeSet]

T(n) = O(n*logn)

2. Space Analysis:

In this approach, we are using extra space using a TreeSet

So, for each exploration of contiguous subsequence, we store the elements in a TreeSet and hence in worst case the entire list may be unique

So, we require space which is proportional to the size of the list in worst case.

Therefore,

S(n) = O(n)

3. Now, we do some time analysis on some actual sizes of data and record the times:

Time taken for size 80000(input-6.txt file) = 9.91 seconds

Note: For smaller values of n i.e [10,20,100,1000], the time to run the code is insignificant

From the two approaches, we see that in the second approach, the runtime is considerably faster for same size of input