Group 2

Trần Hoàng Bảo Ly – 21521109

Lê Thu Hà – 21520800

Question 10

The *range* of a finite non empty set of n real numbers S is defined as the difference between the largest and smallest elements of S. For each representation of S given below, describe in English an algorithm to compute the range. Indi- cate the time efficiency classes of these algorithms using the most appropriate notation (O, big Theta, or big Omega).

a. Anunsortedarray

b. A sorted array

c. A sorted singly linked list

d. A binary search tree

Solution

Answer:

Range, in statistics, is the difference between the highest and the lowest value in the set.

Thus, our task is to identify the highest and lowest values in the given type of sets and then calculate the range.

a) An unsorted array

take the first number in list to initialize two variables "high" and "low".

Now we start iterating through a set of operations. These operations include

check if the next number in the list is greater than that of variable "high", if so, update the variable "high" otherwise

check if the next number in the list is smaller than that of variable "low", if so, update the variable "low".

This would be repeated for all the numbers in the list. at the end of list we could get the least value in the list and also the highest value in the list. Then, we can find the range easily by calculating their difference.

Time complexity for doing this would be Ω(n), where n is the length of the list or simply number of elements in the list.

Space complexity would be O(n)

b) A sorted array

In this, if we assume that the list is sorted in asscending order, we can find that the first element in the list is the least value and the last element in the list is the highest value. Calculating the difference between them would give the range.

Time complexity for accessing those values would be Ω(2) , since we are accessing two values in the list.

Space complexity would be O(n)

c) A sorted singly linked list

Same as a sorted array, we assume that the list is sorted in asscending order. Then we could access the first element from the first node and last element from the last node. Difference between these two node values would be our range.

Time complexity for accessing the first node value is Ω (1)

Time complexity for accessing the last node value is Ω (n)

space complexity for the singly list consisting of n values is O(n)

d) Binary search tree

in this, we have to search for two values: one which is highest and one which is the least. Then we can calculate the range by simply finding the difference between them.

Time complexity for searching each value in the binary search tree is Ω (nlog(n))

Question 11

Lighter or heavier? You have n > 2 identical-looking coins and a two-pan balance scale with no weights. One of the coins is a fake, but you do not know whether it is lighter or heavier than the genuine coins, which all weigh the same. Design a $\theta(1)$ algorithm to determine whether the fake coin is lighter or heavier than the others.

Answer

Algorithm To determine fake coin is lighter or heavier.

Step 1: Divide n coins into 3 equal parts.

Step 2: Take the first 2 parts and put them on the two-pan balance scale. There are 2 possible cases:

- Two-pan scale balance, remaining part contain fake coin. Go to step 3
- Two-pan scale unbalance on of the two part contain fake coin . Go to step 4

Step 3: Replace one of the two parts with the remaining part, if two pan scale skew to the side of remaining part, the fake coin is heavier, otherwise the fake coin is lighter.

Step 4: Replace the heavier parts with the remaining part. If two pan scale still balance, it mean that the part has just been place contains the fake coin and it heavier. Otherwise the part left contains the fake coin and it lighter.

Note: We only need 2 weight. So the time complexity for this algorithm is $\theta(1)$.

Question 12

Consider the following version of an important algorithm that we will study

later in the book.

ALGORITHM GE(A[0..n − 1, 0..n])

//Input: An n × (n + 1) matrix A[0..n − 1, 0..n] of real numbers

for i ← 0 to n − 2 do

    for j ← i + 1 to n − 1 do

        for k ← i to n do

            A[j, k]← A[j, k] − A[i, k] ∗ A[j, i] / A[i, i]

a. Find the time efficiency class of this algorithm.

b. What glaring inefficiency does this pseudocode contain and how can it be

eliminated to speed the algorithm up?

Answer

a. Find the time efficiency class of this algorithm.

$$C(n) = \sum_{i=0}^{n-2} \sum_{j=i+1}^{n-1} \sum_{k=j+1}^{n} 1$$

$$= \sum_{i=0}^{n-2} \sum_{j=i+1}^{n-1} n - j = n \sum_{i=0}^{n-2} \sum_{j=i+1}^{n-1} 1 - \sum_{i=0}^{n-2} \sum_{j=i+1}^{n-1} j = n \sum_{i=0}^{n-2} n - 1 - i - \sum_{i=0}^{n-2} \sum_{j=i+1}^{n-1} j$$

$$We\ have \sum_{i=0}^{n-2} n - 1 - i = (n-1) + (n-2) + \cdots + 1 = \frac{n(n-1)}{2}$$

$$So\ C(n) = n.\frac{n(n-1)}{2} - \sum_{i=0}^{n-2} \sum_{j=i+1}^{n-1} j \approx \frac{n^3}{2} - \sum_{i=0}^{n-2} \sum_{j=i+1}^{n-1} j$$

$$Remove\ lowerbound\ we\ have\ \ C(n) \in O(n^3)$$

b. What glaring inefficiency does this pseudocode contain and how can it be eliminated to speed the algorithm up?

We can compute A[j, i] / A[i, i] in A[j, k]← A[j, k] − A[i, k] ∗ A[j, i] / A[i, i] outside third for-loop, Now the pseudocode can be rewritten as follows:

```
for i ← 0 to n − 2 do
    for j ← i + 1 to n − 1 do
        tmp = A[j, i] / A[i, i]
        for k ← i to n do
            A[j, k]← A[j, k] − A[i, k] ∗ tmp
```

For old version each loop we need to calculate 3 operator -,*,/ but know we just need calculate 2 operater -,* the / operater was calcutate outside of third loop this will speed the algorithm up.