Đại học Quốc gia Thành phố Hồ Chí Minh

Trường Đại học Công nghệ Thông tin

Bài tập nhóm tuần 2

Đánh giá độ phức tạp của thuật toán không đệ quy

CS112 - Phân tích và Thiết kế Thuật toán

Nhóm: 05
Lớp: CS112.N21.KHTN
MSSV: 21520029, 21521845

Ngày 27 tháng 3 năm 2023

Bài tập 1

The range of a finite nonempty 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. Indicate the time efficiency classes of these algorithms using the most appropriate notation ($\mathcal{O}$, Θ, or Ω).

a. An unsorted array
b. A sorted array
c. A sorted singly linked list
d. A binary search tree

Solution

For an unsored array, the process of finding minimum and maximum value take $\mathcal{O}(n)$ by traverse all elements. Algorithm C++ code is as following:

```
max_value = a[0];
min_value = a[0];
for(int i = 1; i < a.size(); i++)
{
   max_value = max(max_value,a[i]);
   min_value = min(min_value,a[i]);
}
```

For a sorted array, we already have the minimum value is at the beginning of the array, and maximum one is the end, so it only takes $\mathcal{O}(1)$ to acess these elements.

For a sorted singly linked list, the index of extremas does not change, taking the minimum value element should take $\mathcal{O}(n)$. However, due to the fact that the linked list only has one head, so getting the last element of this list costs $\mathcal{O}(n)$.

For a binary search tree, the minimum value is located at the left-most node of the tree and the maximum value is vice versa. To do this, we should visit the leaf nodes which take $\mathcal{O}(n)$ and $\Theta(\log n)$. Bellow is C++ source code for implementation.

```
int find_min(Node *pos)
{
   if(pos->left == NULL) return pos->value;
   else return find_min(pos->left);
}
int find_max(Node *pos)
{
   if(pos->right == NULL) return pos->value;
   else return find_max(pos->right);
}
```

Bài tập 2

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.

Solution

Consider $n = 3k + p$ where $p \in \{0, 1, 2\}$
Split the coins in to three parts A (k coins), B(k coins), C(k coins) and D(p coins).
First, let us use the scale to compare A and B.

- If weight of A is as same as B:

  Then we sould compare A and C.

  If weight of C is still the same, then the fake coin is in D.

  Reuse this algorithm again with only part D.

  This case takes maximum 4 comparisions.

- If A is heaiver than B:

  Then we sould compare A and C.

  If A weight is the same with C, then the fake coin is in B and it is lighter.

  If A is heavier than C, then the fake coin is in A and it is heavier.

  The case A is lighter than C should not happen because there is only one fake coin.

  This case take 2 comparisions.

- If A is lighter than B:

  Slove like the previous case and change the position of A and B.

In general, we only need to use the scale for not more than 4 times, so the algorithm is $\Theta(1)$.

Bài tập 3

ALGORITHM $GE(A[0..n-1, 0..n])$

//Input: An $n \times (n+1)$ matrix $A[0..n-1, 0..n]$ of real numbers

for $i \leftarrow 0$ **to** $n-2$ **do**

  for $j \leftarrow i+1$ **to** $n-1$ **do**

    for $k \leftarrow i$ **to** n **do**

      $A[j,k] \leftarrow A[j,k] - A[i,k] * A[j,i] \,/\, A[i,i]$

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?

Solution

a.

It is clear that each loop of the algorithm take $\mathcal{O}(n)$, so the total complexity is $\mathcal{O}(n^3)$.

b.

Let cosider the most inside loop where k take value from i to n.

When $k = i$, we have:

$A[j,k] = A[j,k] - A[i,k] * A[j,i]/A[i,i]$

$\Rightarrow A[j,i] = A[j,i] - A[i,j] * A[j,i]/A[i,j]$

$\Rightarrow A[j,i] = A[j,i] - A[j,i] = 0$

When $k > i$, we have:

$A[j,k] = A[j,k] - A[i,k] * A[j,i]/A[i,i]$

$\Rightarrow A[j,k] = A[j,k] - A[i,k] * 0/A[i,j]$ ($A[j,i] = 0$ from case $k = i$)

$\Rightarrow A[j,k] = A[j,k]$

So the algorithm just change the value of $A[j,i]$ to 0 and does not take any affect to the others.

Rewrite the algorithm in C++ language:

```
for (int i = 0; i <= n-2; i++)
{
    for(int j = i+1; j <= n-1; j++)
    {
        A[j][i] = 0;
    }
}
```

The number of loops reduce to 2, so this algorithm should take $\mathcal{O}(n^2)$.