主題 | 日期 | 星期

Algorithm Analysis and Design HW-5

5.3. First it's clear that if there exists a majority equivalence class in the set of size n, then at least one of the two halves has a majority equivalence class.

Then suppose S is the set of all bank cards and isSame(i,j) stands for whether i and j belongs to the same owner (so far)

We can define function f like this:

Function $f(S)$:

  If $|S| = 1$ return true

  Else let $S = S_1 \cup S_2$, $S_1 \cap S_2 = \phi$, $|S_1| = \lfloor \frac{|S|}{2} \rfloor$

    If $f(S_1) = \text{false}$ and $f(S_2) = \text{false}$:

      return false

    If $f(S_1) = \text{true}$:

      let a be a card included by the majority equivalence class

      Test the cards in S_2 with a and get a new equivalence class in S, name it T_1

主題

日期

星期

If $f(S_2) = true$.

let a be a card included by the majority equivalence class

Test the cards in S_2 with a and get a new equivalence class in S, name it T_2

End if

End if

The time cost by function f is $O(n\log n)$.

J.4. (Not solved yet)

主題

日期
星期

5.5. First we need to define a new structure called "group", each group contains several lines and some of their crossing points.
If a group g contains n lines, then it has at most $(n-1)$ points, each divides different areas apart.

At first we have n independent groups, each with one line only.
The first step, we combine them into pairs, and we get $[\frac{n}{2}]$ new groups. In each group there are one or two lines. And for a group with 2 lines l_1, l_2 and (maybe) a crossing point (x_0, y_0), we have $l_1 > l_2$ when $x < x_0$ and $l_1 < l_2$ when $x > x_0$.
Next we continue such process and get $[\frac{n}{4}]$ groups.
...
By this means we finally finish merging all groups into one in $[\log_2 n]$ rounds. And the total time cost is $O(n\log n)$.

This algorithm can calculate faster than normal because with the help of the structure "group", we don't need to consider all crossing points.

主題

日期
星期

5.6 First we introduce the algorithm:

```
Function f(depth, node):
    If depth = d-1:
        return node
    Else if the father of the node and node's
     children are all larger than node:
        return node
    Else:
        Return f(depth+1, node's larger child)
```

It's clear this algorithm has time complexity $O(\log n)$
So we only need to prove it outputs the correct answer

Case 1: f returns a node who's smaller than it's father and two children. ✓

Case 2: f returns a leaf node, which means the leaf node is larger than it's father. ✓

In conclusion, this algorithm fits the requirements