091M4041H - Algorithm Design and Analysis

Assignment 1

Tianyu Cui 201818018670007

October 8, 2018

Contents

1 Problem One **3**
- 1.1 Algorithm Description . . . 3
- 1.2 subproblem reduction graph . . . 4
- 1.3 the correctness of the algorithm . . . 4
- 1.4 the complexity of the algorithm . . . 4

2 Problem Two **5**
- 2.1 Algorithm Description . . . 5
- 2.2 subproblem reduction graph . . . 6
- 2.3 the correctness of the algorithm . . . 6
- 2.4 the complexity of the algorithm . . . 6

3 Problem Three **7**
- 3.1 Algorithm Description . . . 7
- 3.2 subproblem reduction graph . . . 8
- 3.3 the correctness of the algorithm . . . 8
- 3.4 the complexity of the algorithm . . . 8

4 Problem Four **9**
- 4.1 Algorithm Description . . . 9
- 4.2 subproblem reduction graph . . . 10
- 4.3 the correctness of the algorithm . . . 10
- 4.4 the complexity of the algorithm . . . 10

5 Problem Five **11**
- 5.1 Algorithm Description . . . 11
- 5.2 subproblem reduction graph . . . 11
- 5.3 the correctness of the algorithm . . . 11
- 5.4 the complexity of the algorithm . . . 11

6 Problem Six **12**
6.1 Algorithm Description 12
6.2 subproblem reduction graph 12
6.3 the correctness of the algorithm 12
6.4 the complexity of the algorithm 12

1 Problem One

You are interested in analyzing some hard-to-obtain data from two separate databases. Each database contains n numerical values, so there are 2n values total and you may assume that no two values are the same. You'd like to determine the median of this set of 2n values, which we will define here to be the nth smallest value.

However, the only way you can access these values is through queries to the databases. In a single query, you can specify a value k to one of the two databases, and the chosen database will return the kth smallest value that it contains. Since queries are expensive, you would like to compute the median using as few queries as possible.

Give an algorithm that finds the median value using at most O(logn) queries.

1.1 Algorithm Description

To find the midian value of the two databases, the algorithm finds the middle of two databases and compare them. In every operation, find the correct part midian value should exist in. Repeat find the middle of the remain part until the two middles are equal.

Algorithm 1 Find the midian value in two databases

```
1:  function MidianSearch(arrayA, arrayB, n)
2:      leftA = leftB = 0;
3:      rightA = rightB = n;
4:      while true do
5:          midianA = arrayA[⌈(rightA − leftA)/2⌉ − 1];
6:          midianB = arrayB[⌈(rightB − leftB)/2⌉ − 1];
7:          if ⌈(rightA − leftA)/2⌉ == 1 then
8:              return min(arrayA[0], arrayB[0]);
9:          end if
10:         if midianA == midianB then
11:             return midianA;
12:         end if
13:         if midianA > midianB then
14:             rightA = (rightA - leftA)/2;
15:             leftB = (rightB - leftB)/2
16:         else
17:             rightB = (rightB - leftB)/2;
18:             leftA = (rightA - leftA)/2
19:         end if
20:     end while
21: end function
```

1.2 subproblem reduction graph

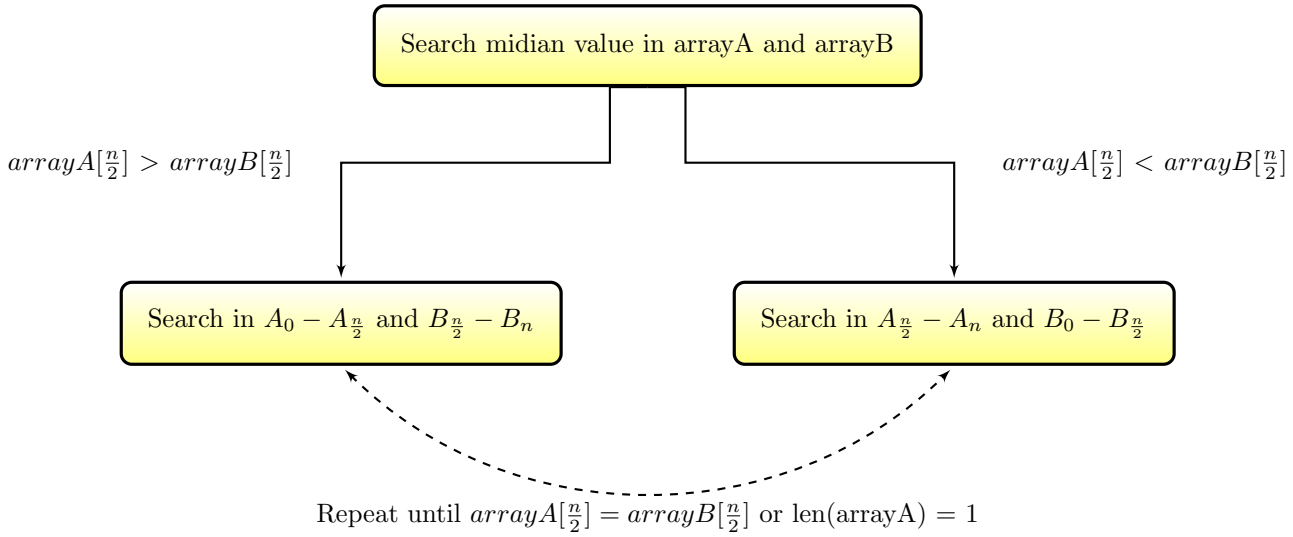

Figure 1: Subproblem reduction graph in problem one

1.3 the correctness of the algorithm

For n = 1, the answer is min(A[0], B[0]).

For n > 1, we choose the midian value of the two databases A[$\frac{n}{2}$] and B[$\frac{n}{2}$]. If A[$\frac{n}{2}$] > $B[\frac{n}{2}]$, because the two databases are already sorted, the midian value must be in the part that the value is smaller than A[$\frac{n}{2}$] and bigger than B[$\frac{n}{2}$]. Also if A[$\frac{n}{2}$] < $B[\frac{n}{2}]$, the midian value must be in the part that the value is bigger than A[$\frac{n}{2}$] and smaller than B[$\frac{n}{2}$]. In the next operation, we can find the remain of the part in two databases. And if A[$\frac{n}{2}$] = B[$\frac{n}{2}$], we can konw that the number the front part of A[$\frac{n}{2}$] and B[$\frac{n}{2}$] is equal to the later part. So the answer is A[$\frac{n}{2}$] or B[$\frac{n}{2}$]. So repeat the operation of finding the midian of the two searchable parts until we find A[$\frac{n}{2}$] = $B[\frac{n}{2}]$ or the number of the remain part is 1. The answer is indeed correct.

1.4 the complexity of the algorithm

Because we only find the middle of two array, we can describe the complexity as T(n) = 2T(n/2) = O(logn).

2 Problem Two

Given a binary tree, suppose that the distance between two adjacent nodes is 1, please give a solution to fnd the maximum distance of any two node in the binary tree.

2.1 Algorithm Description

The algorithm finds the maximum distance in left and right subtree throung the current root node.

Algorithm 2 Find the max distance in a binary tree

```
1: static MaxLen = 0;
2: function FindMaxLen(BTreeroot)
3:     if root == NULL then
4:         return 0;
5:     end if
6:     if root → left == NULL then
7:         root → MaxLeftDistance = 0;
8:     end if
9:     if root → right == NULL then
10:        root → MaxRightDistance = 0;
11:    end if
12:    if root → left! = NULL then
13:        FindMaxLen(root → left);
14:        if root→ left → MaxLeftDistance > root → left → MaxRightDistance then
15:            root → MaxLeftDistance = root → left → MaxLeftDistance + 1
16:        else
17:            root→ MaxLeftDistance = root → left → MaxRightDistance + 1
18:        end if
19:    end if
20:    if root → right! = NULL then
21:        FindMaxLen(root → right);
22:        if root → right → MaxLeftDistance > root → right → MaxRightDistance then
23:            root→ MaxRightDistance = root → right → MaxLeftDistance + 1
24:        else
25:            root→ MaxRightDistance = root → right → MaxRightDistance + 1
26:        end if
27:    end if
28:    if root → MaxRightDistance + root → MaxRightDistance > MaxLen then
29:        MaxLen = root→ MaxRightDistance + root → MaxRightDistance;
30:    end if
31: end function
```

2.2 subproblem reduction graph

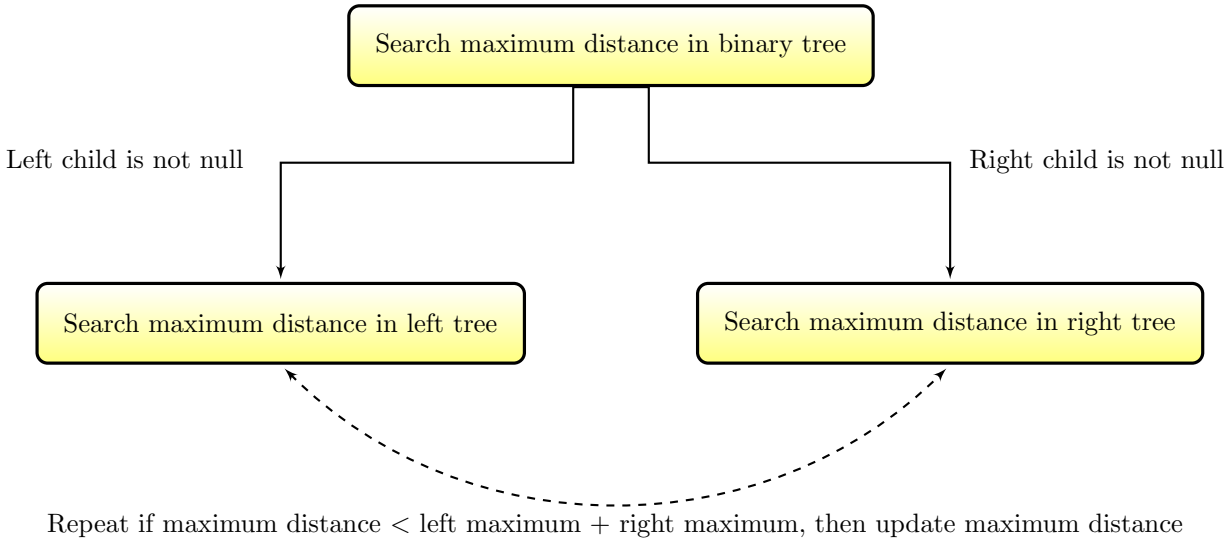

Figure 2: Subproblem reduction graph in problem two

2.3 the correctness of the algorithm

We assumpe the two nodes farthest from the Kth subtree: u_k and v_k, whose distance is defined as d(u_k, v_k), then the node u_k or v_k is the node with the longest distance from the subtree K to the root node R_k. Without loss of generality, we set u_k to be the node with the longest distance R_k from the root node in subtree K, and its distance from the root node is defined as d(u_k, R). Taking the two largest values max_1 and max_2 of d(u_i,R)($1 \geq i \geq k$), then the longest path through the root node R is $max_1 + max_2 + 2$, so the farthest distance in the tree R is The distance between the two points is: $\max(d(u_1, v_1), ..., d(u_k, v_k), max_1 + max_2 + 2)$. So the algorithm is correct.

2.4 the complexity of the algorithm

Beccause the algorithm searches the left and right child in the binary tree, the complexity is equal to the problem of searching in binay tree as O(logn).

3 Problem Three

Consider an n-node complete binary tree T, where n = $2^d - 1$ for some d. Each node v of T is labeled with a real number x_v. You may assume that the real numbers labeling the nodes are all distinct. A node v of T is a local minimum if the label x_v is less than the label x_w for all nodes w that are joined to v by an edge.

You are given such a complete binary tree T, but the labeling is only specifed in the following implicit way: for each node v, you can determine the value x_v by probing the node v. Show how to find a local minimum of T using only O(logn) probes to the nodes of T.

3.1 Algorithm Description

The algorithm searches a local minimum in complete binary tree. In every operation, find the left and right child who is smaller than current nodes.

Algorithm 3 Find a local minimum in complete binary tree

```
1: function FindLocalMin(BTreeroot)
2:     if root == NULL then
3:         return 0;
4:     end if
5:     if root → left == NULL and root → right == NULL then
6:         return root → value;
7:     end if
8:     if root → value < root → left → valueandroot → value < root → right → value then
9:         return root → value;
10:    end if
11:    if root → value ≥ root → left → value then
12:        FindLocalMin(root → left);
13:    end if
14:    if root → value ≥ root → right → value then
15:        FindLocalMin(root → right);
16:    end if
17: end function
```

3.2 subproblem reduction graph

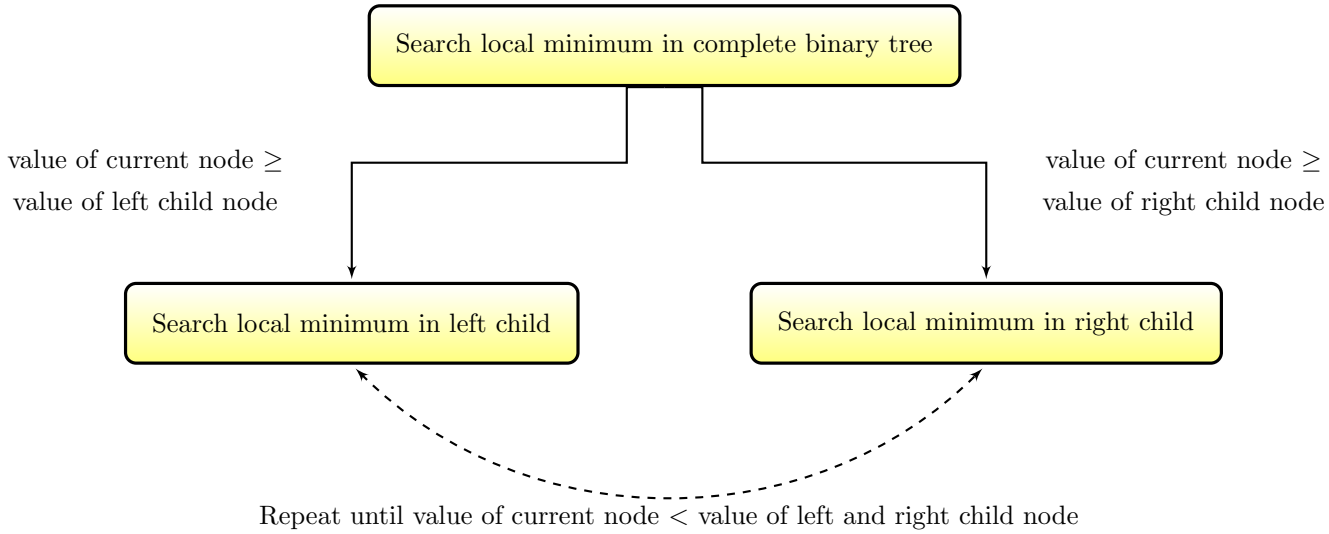

Figure 3: Subproblem reduction graph in problem three

3.3 the correctness of the algorithm

We know that a local minimum is smaller than the other nodes connect with it. From the root of the binary tree, searching in the root's left and right subtree, we must be able to find a set of descending nodes described as $(v_0, v_1, ..., v_n)$ and the n th node is indeed the local minimum we look for. So the algorithm is correct.

3.4 the complexity of the algorithm

Beccause the algorithm searches the left and right child in the binary tree, the complexity is equal to the problem of searching in binay tree as O(logn).

4 Problem Four

Suppose now that you're given an $n \times n$ grid graph G. (An $n \times n$ grid graph is just the adjacency graph of an $n \times n$ chessboard. To be completely precise, it is a graph whose node set is the set of all ordered pairs of natural numbers (i, j), where $1 \leq i \leq n$ and $1 \leq j \leq n$; the nodes (i, j) and (k, l) are joined by an edge if and only if $|i - k| + |j - l| = 1$.)

We use some of the terminology of problem 3. Again, each node v is labeled by a real number x_v; you may assume that all these labels are distinct. Show how to find a local minimum of G using only O(n) probes to the nodes of G. (Note that G has n^2 nodes.)

4.1 Algorithm Description

每一次先寻找竖中轴，寻找中轴上的最小值，找到最小值后试探其左右邻居节点的值是否大于他，如果都大于他，则局部最小就是这个中轴上的最小值，如果邻居小于等于最小值，则继续在小的半区内寻找局部最小，递归直到竖轴线为左边界或右边界。

Algorithm 4 Find a local minimum in an $n \times n$ grid graph G

```
1: function FindLocalMin(Graph[0–n-1][0–n-1])
2:     min = 0, a = 0, b = 0;
3:     for i from 0 to n-1 do
4:         if Graph[i][⌈n/2⌉] > min then
5:             a = i, b = ⌈n/2⌉;
6:         end if
7:     end for
8:     if b == 0 or b == n − 1 then
9:         return Graph[a][b];
10:    end if
11:    if  thenGraph[a][b] ≥ Graph[a][b − 1]
12:        return FindLocalMin(Graph[0 − −n − 1][0 − −b − 1]);
13:    end if
14:    if  thenGraph[a][b] ≥ Graph[a][b + 1]
15:        return FindLocalMin(Graph[0 − −n − 1][b + 1 − −n − 1]);
16:    end if
17:    if  thenGraph[a][b] < Graph[a][b − 1] and Graph[a][b] < Graph[a][b + 1]
18:        return Graph[a][b];
19:    end if
20: end function
```

4.2 subproblem reduction graph

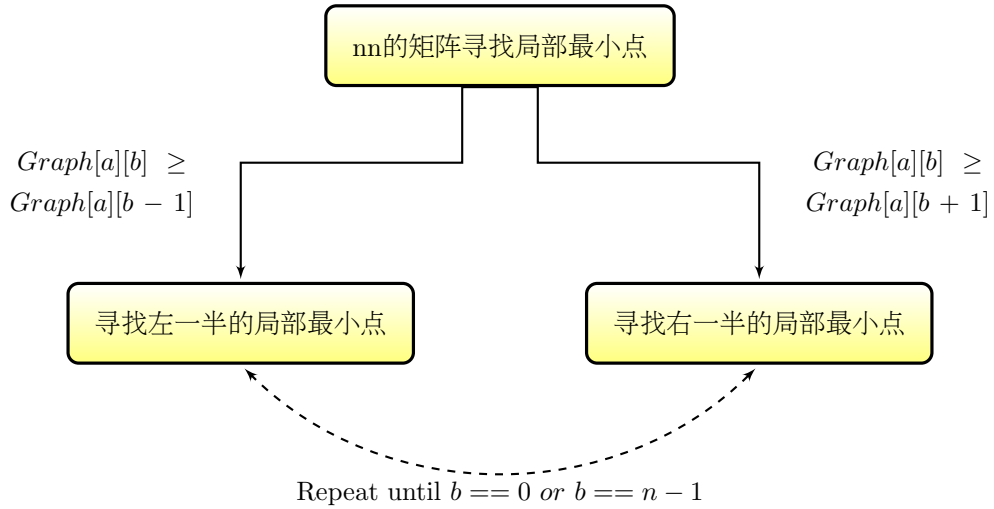

Figure 4: Subproblem reduction graph in problem four

4.3 the correctness of the algorithm

数学归纳法证明n = 1 时，值为Graph[0][0]

$n > 1$时，所有大于1的正方形均能找到其最小值，算法是正确的。

4.4 the complexity of the algorithm

T(n) = T(n/2) + O(n) = O(n)

5 Problem Five

Recall the problem of finding the number of inversions. As in the course, we are given a sequence of n numbers $a_1, ..., a_n$, which we assume are all distinct, and we define an inversion to be a pair $i < j$ such that $a_i > a_j$.

We motivated the problem of counting inversions as a good measure of how different two orderings are. However, one might feel that this measure is too sensitive. Let's call a pair a significant inversion if $i < j$ and $a_i > 3a_j$. Given an O(nlogn) algorithm to count the number of significant inversions between two orderings.

5.1 Algorithm Description

The algorithm searches a local minimum in complete binary tree. In every operation, find the left and right child who is smaller than current nodes.

Algorithm 5 aaa

1: **function** FINDLOCALMIN($BTreeroot$)
2: **end function**

5.2 subproblem reduction graph

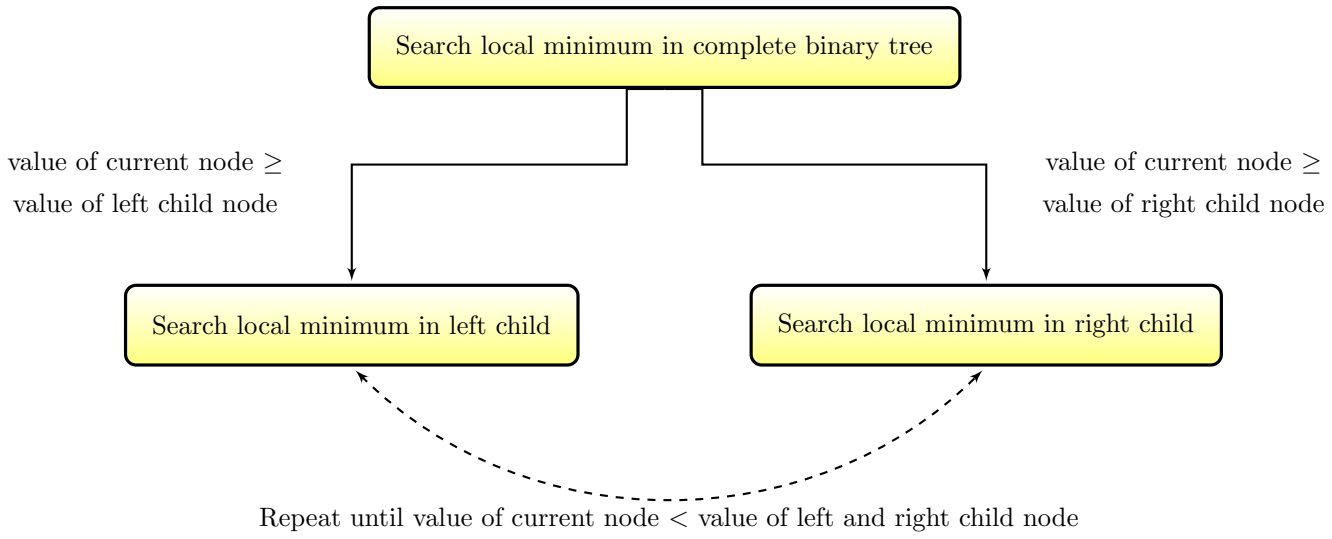

Figure 5: Subproblem reduction graph in problem three

5.3 the correctness of the algorithm

5.4 the complexity of the algorithm

6 Problem Six

Given a table M consisting of $2^n * 2^n$ blocks, we want to fill it with a L-shaped module (consisting of three blocks). The L-shaped module is shown below.

Please give a fill method, so that the last element of the table ($M_{2^n,2^n}$) is empty.

6.1 Algorithm Description

算法每次将平面分成四份，每次考虑4份中的被占的位置在左上，左下，右上，右下，并将L型块放在中间部分的对应位置，保证每份中只有一个位置被占用，如此递归，直到被分成的部分只剩下一个已被占用的点。

6.2 subproblem reduction graph

空点的位置为左上右上左下和右下角4种情况，图中省略了左下和右下两种情况。

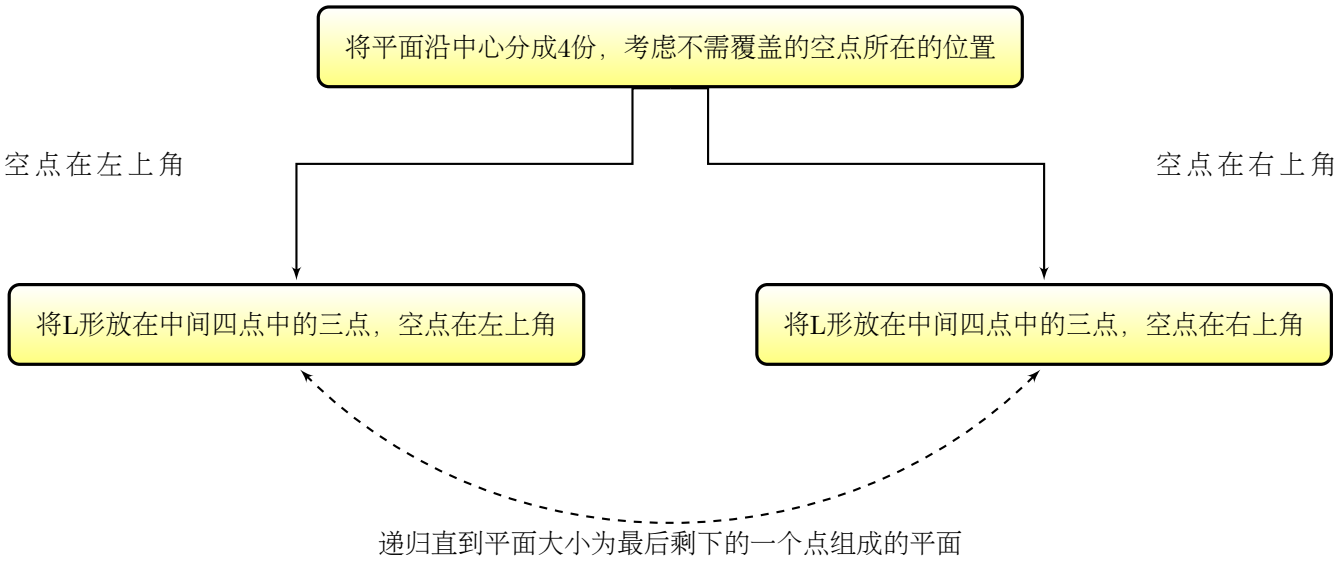

Figure 6: Subproblem reduction graph in problem six

6.3 the correctness of the algorithm

数学归纳法证明

6.4 the complexity of the algorithm

T(n)= T(n-1)+ c = O(n)

Algorithm 6 A fill method with L-shaped module

for i *from* 1 *to* 2^n **do**
  for j *from* 1 *to* 2^n **do**
    $M[i][j] = 0$;
  end for
end for
$i = j = 2^n$;
function FillL-Shaped($M[1-2^n][1-2^n], i, j$)
  if $2^n == 1$ **then**
    return 0;
  end if
  if $i == 1, j == 1$ **then**
    $M[2^n/2][2^n/2+1] = M[2^n/2+1][2^n/2] = M[2^n/2+1][2^n/2+1] = 1$;
    $FillL - Shaped(M[1-2^n/2][1-2^n/2], 1, 1)$;
    $FillL - Shaped(M[1-2^n/2][2^n/2+1-2^n], 2^n/2, 2^n/2+1)$;
    $FillL - Shaped(M[2^n/2+1-2^n][1-2^n/2], 2^n/2+1, 2^n/2)$;
    $FillL - Shaped(M[2^n/2+1-2^n][2^n/2+1-2^n], 2^n/2+1, 2^n/2+1)$;
  end if
  if $i == 2^n, j == 1$ **then**
    $M[2^n/2][2^n/2+1] = M[2^n/2][2^n/2] = M[2^n/2+1][2^n/2+1] = 1$;
    $FillL - Shaped(M[1-2^n/2][0-2^n/2], 2^n/2, 2^n/2)$;
    $FillL - Shaped(M[1-2^n/2][2^n/2+1-2^n], 2^n/2, 2^n/2+1)$;
    $FillL - Shaped(M[2^n/2+1-2^n][1-2^n/2], 2^n, 1)$;
    $FillL - Shaped(M[2^n/2+1-2^n][2^n/2+1-2^n], 2^n/2+1, 2^n/2+1)$;
  end if
  if $i == 1, j == 2^n$ **then**
    $M[2^n/2][2^n/2] = M[2^n/2+1][2^n/2] = M[2^n/2+1][2^n/2+1] = 1$;
    $FillL - Shaped(M[1-2^n/2][0-2^n/2], 2^n/2, 2^n/2)$;
    $FillL - Shaped(M[1-2^n/2][2^n/2+1-2^n], 1, 2^n)$;
    $FillL - Shaped(M[2^n/2+1-2^n][1-2^n/2], 2^n/2+1, 2^n/2)$;
    $FillL - Shaped(M[2^n/2+1-2^n][2^n/2+1-2^n], 2^n/2+1, 2^n/2+1)$;
  end if
  if $i == 2^n, j == 2^n$ **then**
    $M[2^n/2][2^n/2+1] = M[2^n/2+1][2^n/2] = M[2^n/2][2^n/2] = 1$;
    $FillL - Shaped(M[1-2^n/2][1-2^n/2], 2^n/2, 2^n/2)$;
    $FillL - Shaped(M[1-2^n/2][2^n/2+1-2^n], 2^n/2, 2^n/2+1)$;
    $FillL - Shaped(M[2^n/2+1-2^n][1-2^n/2], 2^n/2+1, 2^n/2)$;
    $FillL - Shaped(M[2^n/2+1-2^n][2^n/2+1-2^n], 2^n, 2^n)$;
  end if
end function