算法设计与分析-work4

yu wang

April 2024

1 问题一

我们假设该算法的时间复杂度为 T（n），该数组形成的最大堆的高度为 h，于是我们可以得到 $2^{h-1} \leq n \leq 2^h - 1$。我们以满二叉树为例子，于是我们可以得到，每次调用函数的数，如果是第一层，节点数为 2^0, 需要调整的层数为 h-1 层，如果是第二层，节点数为 2^1，需要调整的层数为 h-2 层..... 以此类推我们可以得到最后一层的节点数为 2^{h-2}，需要调整的层数为 1。于是我们可以得到 T(n).

$T(n) = 2^0 * (h-1) + 2^1 * (h-2) + 2^2 * (h-3) + ... + 2^{h-2} * 1(1)$

我们利用高中的错位相减知识来进行求解

$2T(n) = 2^1 * (h-1) + 2^2 * (h-2) + 2^3 * (h-3) + ... + 2^{h-1} * 1(2)$

将（2）式减去（1）式可得

$T(n) = 2^0 + 2^1 + 2^2 + ... + 2^{h-2} - h + 1$

接下来根据等比公式可得

$T(n) = 2^h - 1 - h$ 又因为满二叉树,$n2^h - 1$,所以 $T(n) = 2^h - 1 - h = n - h$, 同时我们可以得到 h 为对数阶。在之前的学习中，我们已经知道对数阶是小于多项式阶，所以原始可以继续变为如下

$T(n) = 2^h - 1 - h = n - h \leq n$

所以对于满二叉树而言，调整为最大堆的时间复杂度为 O（n）。而对于非满二叉树而言，每层节点需要调整的层数只可能更少，总的时间复杂度不会超出 n，同样可以推出为 O（n）. 最终我们得到调整为最大堆的时间复杂度为 O（n）

2 问题二

A 是数字 sanjiaoxing,n 代表行数

MAX-PATH(A,n)

1: if n==0 then

2: return 0

3: if n==1

4: return A[1][1]

5: for i=1 to 2^{n-1}

6: dp[n][i]=A[n][i]

4: for i=n-1 to 1

5: for j=1 to 2^{i-1}

6: dp[i][j]=A[i][j]+max(dp[i+1][2j],dp[i+1][2j-1])

8: Find-Path(dp,n)

9: return dp[1][1]

-

Find-Path(dp,n)

1: k=1

2: for i=1 to n

3: x=0,node=a_{i1}

4: for j=2k-1 to 2k

5: if dp[i]][j]>x

6: x=dp[i][j]

7: node=a_{ij}

8: k=j

9: print node

在这个问题中，我们可以将数字三角形不断划分为左右的子三角形，每个子三角形的顶部都是一个数字，即对于每个顶部数字，我们可以选择向下走到左侧相邻的数字或者向下走到右侧相邻的数字，以获得更大的路径和。

假设从顶部数字开始，我们知道左右子三角形的最优路径和，那么从顶部数字开始的最优路径和就可以通过比较两个子三角形的最优路径和来确定。因此，我们可以确定这个问题的最优子结构。

3 问题三

我们假设我们已经得到了小偷前 n 家商店投钱财的最优解。对于小偷，我们可以得知小偷偷前 n 家店的最优解是要么偷了第 n 家商店的情况要么偷了第 n-1 家商店，那么最优的情况就是比较这两种情况的最大值。接下来如何找到这两个情况的最优子结构。该问题的最优子结构是在所有商家中，对于前 i 个商家，能够找到偷取的最大价值。这个最多的偷取钱的最优解，可以由之前的子问题的最优子解得出，即对于每个商店 i，最优解是由之前的商店（i-1 和 i-2）的最优解推导而来的，对于 i-2 个商店可以继续偷取第 n 家商店，i-1 个商店和前面偷取钱财的最大金额就为前 i 个商店偷取的最大钱财。

shopv 存储的是所有商家的钱财，n 为商家的个数

```
maxrobmoney(n,shopv)
1: Let prices[1..n] be a new array, and set every element to 0.
2: if n==0
3:     return 0
4: prices[1]=shopv[1]
5: if n==1
6:     return prices[1]
7: prices[2]=max(shopv[1],shopv[2])
8: if n==2
9:     return prices[2]
10: for i=3 to n
11:     prices[i]=prices[i-1]
12:     temp=prices[i-2]+shopv[i]
13:     if temp>prices[i]
14:         prices[i]=temp
15: return prices[n]
```

接下来我们应用强归纳公理验证算法的正确性

首先，确定谓词 P（n）：

.P(n)：该算法能够求解出个数为 n 的商家中你能够偷取的最大钱财

第二步是证明基本情况 P（0）和 P（1）：当数组为空时，没有商家偷取钱财，返回 0，当只有一个商家的时候，偷取商店的最大钱财就是这个商家的钱财。

第三步是证明一般情况 $\forall n \in N(P(0) \wedge P(1) \wedge \cdots \wedge P(n) \Rightarrow P(n+1))$, 假设对于任意个数为 0,1,2，...,n 的商家，该算法都能够求解出小偷能够偷取的最大钱财。则对于任意一个个数为 n+1 的商家，算法会去考虑是否偷取第 n+1 的商家，如果偷取第 n+1 的商家，则最大钱财由 n+1 商家的钱财和前 n-1 家商家偷取的最大钱财的和确定；如果不偷取第 n+1 家商店，则最大钱财是偷取钱家商家的最大钱财。则最终的最大钱财由两种情况的最大钱财决定。命题得证，我们可以证明该算法能够求解出个数为 n 的商家中你能够偷取的最大钱财。