Algorithm Design and Analysis - Assignment 2

2020K8009907032

唐嘉良

2022 年 11 月 3 日

目录

1 Problem One

A robber is planning to rob houses along a street. Each house has a certain amount of money stashed, the only constraint stopping you from robbing each of them is that adjacent houses have security system connected and it will automatically contact the police if two adjacent houses were broken into on the same night.

1. Given a list of non-negative integers representing the amount of money of each house, determine the maximum amount of money you can rob tonight without alerting the police.

2. What if all houses are arranged in a circle?

1.1 Algorithm Description

采用动态规划的方法。设dp[n]为从第1个房屋开始打劫到第n个房屋所获得的总收益，circle[n-1]为从第2个房屋开始打劫到第n个房屋所获得的总收益。对于问题1，分为是否抢劫第n个房屋给出dp的状态转移式。对于问题二，仍然分为是否抢劫第n个房屋给出状态转移式，但是必须同时给出dp和circle的状态转移式，并利用circle来求取dp[n-1]。

Algorithm 1 Money Robbing

```
1:  function MaxMoney(money, n)
2:      dp[n],circle[n];
3:      dp[0] = money[0];
4:      dp[1] = max(money[0],money[1]);
5:      circle[0] = 0;
6:      circle[1] = money[1];
7:      circle[2] = max(money[1],money[2]);
8:      if n == 1 then
9:          return money[0];
10:     end if
11:     if n == 2 then
12:         return max(money[0],money[1]);
13:     end if
14:     //If houses are in a circle, do this 'for' to compute circle[]
15:     for i = 3 to n do
16:         circle[i] = max(circle[i-1],circle[i-2]+money[i]);
17:     end for
18:     for i = 2 to n − 1 do
19:         dp[i] = max(dp[i-1],dp[i-2]+money[i]);
20:         //if in a circle, do this sentence instead of the front one
21:         dp[i] = max(dp[i-1],circle[i-2]+money[i]);
22:     end for
23:     return dp[n-1];
24: end function
```

1.2 the optimal substructure and DP equation

最优子结构是偷取前n-1个房屋和偷取前n-2个房屋的最大收益。因为偷n个房屋的最大收益可以分为两种情况：偷第n个，那么最优子结构是偷前n-2个；不偷第n个，那么最优子结构是偷前n-1个。动态规划的状态转移方程为dp[i] = max(dp[i-1],dp[i-2]+money[i]);

如果房屋呈环形，那么先利用同样的状态转移方程计算从第二个房屋开始偷的最大收益，再利用计算出的最大收益计算总最大收益，状态转移方程为dp[i] = max(dp[i-1],circle[i-2]+money[i]);

1.3 the correctness of the algorithm

正确性的说明是简单的。对于只有一个或两个房子的情况，显然无论是否环形都是偷最有钱的那一家。对于大于等于3个房子的情况，考虑最后一个房子是否偷，如果偷，那么倒数第二个房子不能被偷，而前n-2个房子则是自由偷，是最优子结构，用最后一个房子的收益加上前n-2个房子的最大收益即可。

对于环形的情况，同样考虑最后一个房子是否被偷，与线性排列唯一的不同点是如果最后一个房子被偷，那么第一个房子不能被偷，第二个到第n-2个房子是自由偷取的。于是先计算一个排除第一个房子之后的最大收益数组circle，替换线性情况的dp[n-2]即可。

1.4 the complexity of the algorithm

由于线性的计算方式，只需要扫一遍数组即可得到答案，无论环形还是线性排列，时间复杂度都为 T(n) = O(n).

最多只需要两个长度为n的数组，空间复杂度为 S(n) = O(n).

2 Problem Two

An ugly number is a positive integer whose prime factors are limited to 2, 3, and 5.

Given an integer n, return the nth ugly number.

2.1 Algorithm Description

本题思路是：第n个数肯定是前n-1个数中某个数的2倍或3倍或5倍，维护三个指针搜寻前n-1个数中自身2/3/5倍恰好大于第n-1个数的数即可。且下一次搜寻的时候接着上次搜寻的指针位置进行搜寻即可，减免时间开销。

Algorithm 2 Ugly Number

```
1: function THE NTH UGLY NUMBER(n)
2:     uglynum[n];
3:     point2=0,point3=0,point5=0;
4:     uglynum[0] = 1;
5:     uglynum[n];
6:     if n == 1 then
7:         return 1;
8:     end if
9:     for i = 1 to n − 1 do
10:        while uglynum[point2]*2 ¡= uglynum[i-1] do
11:            point2++;
12:        end while
13:        while uglynum[point3]*3 ¡= uglynum[i-1] do
14:            point3++;
15:        end while
16:        while uglynum[point5]*5 ¡= uglynum[i-1] do
17:            point5++;
18:        end while
19:        uglynum[i] = min(uglynum[point2],uglynum[point3],uglynum[point5]);
20:    end for
21: end function
```

2.2 the optimal substructure and DP equation

最优子结构是前n-1个丑数，因为第n个丑数一定是前n-1个丑数的某一个的2或3或5倍。动态规划的状态转移方程为uglynum[i] = min(uglynum[point2],uglynum[point3],uglynum[point5])。

2.3 the correctness of the algorithm

正确性是容易说明的。首先，第n个丑数由于只能是2、3、5的倍数，他要么是2、3、5，要么除以2或3或5之后仍然是只是2、3、5的倍数，这个数仍然是丑数，将1视作第一个丑数，我们得到：第n个丑数一定是前n-1个丑数的某一个的2或3或5倍。

因为第n个丑数肯定比第n-1个丑数大，而且按序寻找，那么三大指针目的在于寻找之前丑数的2、3、5倍的数中比第n-1个丑数大的最小的那个，只有这三个有可能是第n个丑数。

2.4 the complexity of the algorithm

三大指针均最多需要扫一遍长度为n的数组，因此时间复杂度为T(n) = O(n). 只需要开一个数组存储计算出来的丑数，空间复杂度为S(n) = O(n).

3 Problem Three

Given n, how many structurally unique BST’s (binary search trees) that store values 1...n?

3.1 Algorithm Description

考虑根节点的左右子树，两个二叉树同构当且仅当左右子树均同构，所以计算不同构的二叉树，仅需按节点数量分类，分别计算左右子树不同构的二叉树数目，相乘并按不同节点数量的情况进行累加即可。

Algorithm 3 Unique Binery Search Trees

```
1:  function UniqueBTNum(n)
2:      num[n];
3:      num[0] = 1, num[1] = 1;
4:      if n == 0||n == 1 then
5:          return 1;
6:      end if
7:      for i = 2 to n do
8:          for j = 0 to i − 1 do
9:              num[i] += num[j]*num[i-1-j];
10:         end for
11:     end for
12:     return num[n];
13: end function
```

3.2 the optimal substructure and DP equation

最优子结构是前面0到n-1个节点的不同构二叉树的数目，不同构的二叉树，仅需按节点数量分类，分别计算左右子树不同构的二叉树数目，相乘并按不同节点数量的情况进行累加即可。这个计算方法得到的答案正是卡特兰数!

3.3 the correctness of the algorithm

对于0个或1个结点的情况，很显然只有1种独特二叉搜索树。

2个或以上节点情况下，两个二叉搜索树不同构等价于根节点左右子树之一不同构，所以对于左右子树根节点数目分类，计算不同结点数下不同构的左右子树的数量，根据组合数学的原理，对其相乘即为该节点数目情况下不同构二叉搜索树的数目，最后依节点数目情况累加即可。算法正确。

3.4 the complexity of the algorithm

需要双重循环计算num[n]，时间复杂度为T(n) = O(n^2). 只需要一个数组存储计算结果，空间复杂度为S(n) = O(n).

进一步地，在发现该问题结果为卡特兰数之后，根据组合数学的知识，我们可以化简出卡特兰数的通项公式，以此代入n以进行不超过O(n)的计算（因为阶乘的计算只需要O(n)复杂度的乘除法），甚至无需写一个上面这样子的O(n^2)的算法。在允许这种优化的情况下，T(n) 不超过 O(n)。