计算机算法设计与分析

第2次作业

主讲老师：卜东波 *Section B*

程逸飞

2021280186700045

第一题 Money Robbing

算法描述

可以将问题抽象成求一个一维列表子集元素和最大的问题，其中的约束条件就是任意两个相邻的元素不能直接取。
问题一：应用自顶向下的思想，最后一个元素取不取，取决于前面的表达式，即如果取最后一个元素，则倒数第二个元素不能去，则最大的和就是最后一个元素的值加上前n-2个元素最大的和；最后一个元素不取，则最大的和是前n-1个元素最大的和。因此得到转移公式即递推公式S(n)=max(S(n-2)+money[n],S(n-1))，使用递归完成该算法（也可以很简单的改成循环）。
问题二：如果是环状的列表，则增加一条约束条件即第一个和最后一个元素不能同时取，于是有了如果取最后一个元素，则直接将整个列表的第一项删除再运行问题一算法；如果不取最后一个元素，则直接求前n-1个元素的最大和即是要求的结果。

伪代码实现

```
ROBBING(n):# n represents length of current money list
1: if n == 1 then
2:    return money list[0];
3: elif n == 2 then
4:    return max(money list[0], money list[1]);
5: else
6:    return max(ROBBING(n-2)+money list[n-1], ROBBING(n-1));
7: end if
ROBBING_CIRCLE(n): # n represents length of current money list
1: Last_unactived = ROBBING(n-1);
2: Remove the first elements of money list;
3: Last_actived = ROBBING(n);
4: return max(Last_unactived, Last_actived);
```

正确性分析

根据递归表达式的推论，首先可以明确最后一个商店偷与不偷是基于偷与不偷获得利益进行比较得来的，那么可以分两种情况讨论，第一种为偷最后一家店，则根据规则倒数第二家店就不能偷，所以此时获得的利益是最后一家店的财产加上前面n-2家店最大能获得的财产；第二种为不偷最后一家店，则舍弃最后一家店，此时能获得的利益是前面n-1家店最大能获得的财产。最后选取两种情况中利益更大的情况即是满足要求的结果。其中前n-2和前n-1家店最大能获得的财产又是基于同样的规则去与可迭代对象加和并取最大值。又规定当n=1时只返回当前值，n=2时返回两个值中更大的值。由此递归转移关系可以证明该算法是正确的。
第二问只需要增加约束条件即第一家店和最后一家店不能同时偷，换个说法就是如果偷最后一家店，则第一家店就不予考虑，于是将第一家店直接从大街上（列表）删除再使用第一问的算法即可；如果不偷最后一家店，则就是最后一家店不予考虑，于是将最后一家店直接从大街上（列表）删除再使用第一问的算法即可。最后取两种结果的最大值即使能偷取的最大财产。由第一问算法的正确性加上问题转化的简单逻辑可以证明第二问算法也是正确的。

复杂度分析

将迭代形式改写成循环形式，即考虑自底向上的思想，商家列表长度为1时，能攫取的最大利益就是该商家价值；商家列表长度为2时，能攫取的最大利益是这两个商家价值的最大值；商家列表长度大于2时，能攫取的最大利益是max{商家列表最后一个的价值+前n-2个商家能攫取的价值，前n-1个商家能攫取的价值}。
所以只需要一个for循环，就可以自底向上找到最终的结果，因此问题一的时间复杂度是O(n)；第二问只需要重复两次第一问算法，因此时间复杂度也是O(n)。

第二题 Largest Divisible Subset

算法描述

首先明确，题目要求是对于一个正整数集合，要求找到一个子集合，约束条件是该子集合中枚举所有的对，这些对都是可互相整除的，求这样的子集合的最大长度。首先想到的是对该集合进行排序，这样对于判断两个数能否互相整除的问题就转化成了，较大的数是否能整除较小的数。定义一个一维数组max_length，其中max_length[i]表示在正整数集合中的第i个数所能形成的最大可除子集合长度。使用parent[i]更新i的前面能整除的数字。通过自底向上的思想，将当前的正整数与前面的正整数进行整除比较，可以整除则保存一下中间值，对前面的所有正整数都比较之后，选择当前候选集里最长的子集合，用其更新当前节点的max_length。最后通过简单的比较判断得到最大可整除子集合的长度。

伪代码实现

```
Largest Divisible Subset(nums):
 1: Sort nums;
 2: max_length = [0] * length of nums;
 3: parent = [0] * length of nums;
 4: max = 0;
 5: max_index = -1;
 6: for i = 0 to length of nums do
 7:   for j = i - 1 to -1 do
 8:     if nums[i] % nums[j] = 0 and max_length[i] < max_length[j] + 1 then
 9:       max_length[i] = max_length[j] + 1;
10:       parent[i] = j;
11:       Update max and max_index if max_length[i] > max;
12:     end if
13:   end for
14: end for
15: return max;
```

正确性分析

该算法对于使用动态规划的思想，对于每一步搜索能和当前节点组成最大可整除子集合的目标集合，并通过筛选所有的候选集合，找到最大长度的可整除子集合，因此每一步都能找到当前元素可组成的最大可整除子集合。因此从最开始的一个节点一直使用该动态转移方程则可以找到整个集合的最大可整除子集合。

复杂度分析

排序使用归并排序，时间复杂度是O(nlogn)，后面的动态规划算法通过两层循环完成整个判断步骤，其时间复杂度为O(n^2)，因此该算法总体的时间复杂度为O(n^2)

第三题 Unique Binary Search Trees

算法描述

该题目需要计算给定个数为n时，可构成的n个节点的所有结构的二叉搜索树的个数。采用自底向上的思想，首先n为0时，为一个空树只有一个节点，返回数值1；n为1时，可以看成是其左子树个数乘以右子树个数，左右都是空子树所以返回数值1；n为2时，由于1和2都可作为根因此使用动态规划的思想，就可以分别计算，即dp[2] = dp[0]*dp[1]* + *dp[1]*dp[0]。因此自底向上可以通过两层循环计算目标数值。

伪代码实现

```
UBST(n):
1: dp = [0] * (n+1);
2: dp[0] = dp[1] = 0;
3: for i = 2 to n + 1 do
4:   for j = 0 to I do
5:     dp[i] = dp[j] * dp[i-j-1];
6:   end for
7: end for
8: return dp[n];
```

正确性分析

该算法基于卡塔兰数的正确性，题目要求是卡塔兰数的一个例子。可知卡塔兰数的递推公式为C_0=1 and C*(n+1)*=∑(i=0)^n⟦C_i C_(n-i) ⟧,for n≥0。因此上述算法是正确的。

复杂度分析

该算法通过两层循环实现动态规划求解，其时间复杂度为O(n^2)。