求解“钻石金字塔”问题

——动态规划与分治法的使用&区别

一、任务要求

1. 问题描述

给定一个n行的三角形金字塔，你现在所在的位置为（0，0）。你的移动方式被限定为只能往下走和往右下方走，即从（x，y）->（x + 1, y）+（x + 1, y + 1）。对于每一个点有一个价值value[i][j]表示你走到这里能够获得多少金币。请你找到一条从第0层到第n-1层的路线，使得能够获得最大数目的金币。

2. 样例输入

```
4
5
1 3
4 8 1
6 2 3 4
```

3. 样例输出

```
19
The best way is :(1,1)->(2,2)->(3,2)->(4,3)
Program ended with exit code: 0
```

二、算法设计

0. 性质分析

在设计算法之前，我们首先要分析一下问题所具有的特性。考虑我们现在的位置为(x, y)，很明显我们只能够从(x-1, y)或(x-1, y-1)两个点来。**显然从(0, 0)到那两个点的路线也必须是能够获得最大价值的路线，否则我们就可以通过替换为该路线来**

获得一条比原来总价值更大的路线。而求(x–1, y)与(x–1, y–1)的最大收益的问题，则是比之前问题规模小的子问题，所以“钻石金字塔”满足最优子结构。

在算法中我们将会重复到达金字塔中的结点，即问题满足重复子问题性质。

1. 分治法

使用从上往下的思考方式，考虑现在所在的位置为（x, y），能够供我们选择的路线无非是（x + 1, y）与（x + 1, y + 1），分别递归求解即可，实现代码如下：

```
int pyramid::DC_make_money(int x, int y)
{
    if (x == n - 1)
        return value[x][y];
    else
    {
        int l = DC_make_money(x + 1, y),
            r = DC_make_money(x + 1, y + 1);

        if (l >= r)
        {
            route[x + 1][y] = y;
            return value[x][y] + l;
        }
        else
        {
            route[x + 1][y] = y + 1;
            return value[x][y] + r;
        }
    }
}
```

2. 动态规划法

使用从下往上的方式，从最后一层开始，然后逐层向上遍历，具体代码如下：

```
int pyramid::DP_make_money()
{
    vector < vector<int> > f(n);
    int ans = 0;

    for (int i = 0; i < n; ++i)
        f[i].resize(n);
```

```
    for (int j = 0; j < n; ++j)
    {
        route[n - 1][j] = j;
        if (ans < f[n - 1][j]) ans = f[n - 1][j];
    }

    for (int i = n - 2; i >= 0; --i)
        for (int j = 0; j <= i; ++j)
            if (f[i + 1][j] > f[i + 1][j + 1])
                route[i][j] = j;
            else
                route[i][j] = j + 1;

    return ans;
}
```

三、结果分析

1. 正确性分析

我建立了另一个Target Scheme，用于验证两个算法的正确性，其做法：

① 随机生成一个层数为15的的钻石金字塔

② 分别运行分治算法与动态规划算法，求得结果

③ 比较二者的答案是否正确

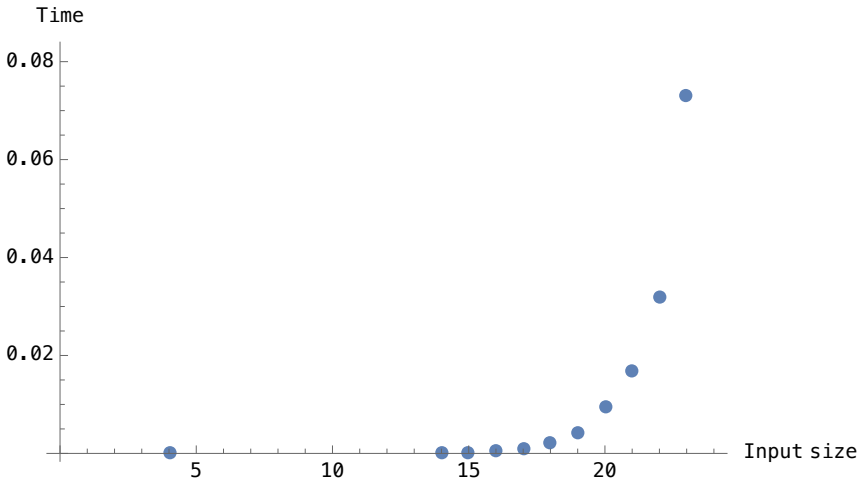
如此循环测试了1000组数据，结果如下：

```
Having tested 1000 examples,
the two algorithm give 1000 right answers of 1000,
accept rate is 100%.
Program ended with exit code: 0
```

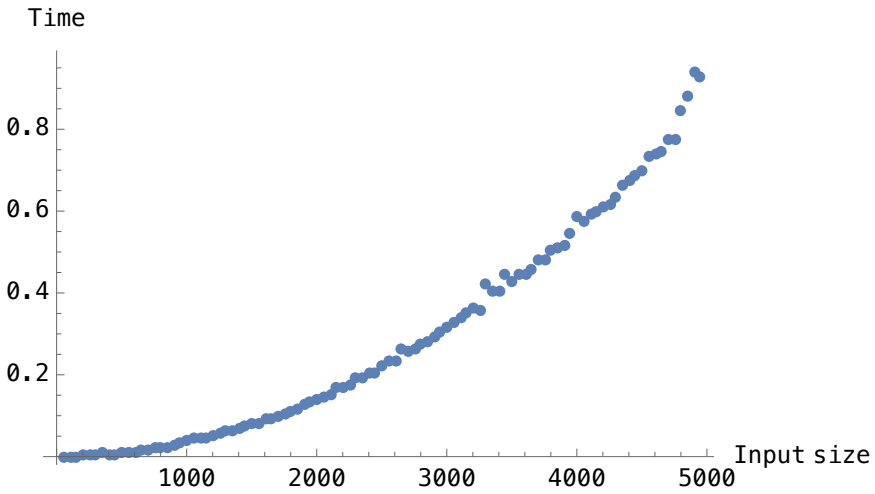
足以说明程序的正确性。

2. 时间复杂度分析

考虑到分治每次递归两个状态，没有做备忘录，其理论复杂度应该在$O(2^N)$，而动态规划的状态空间为$O(N^2)$，每次状态转移花费的时间是O(1)，故动态规划的理论复杂度应该为$O(N^2)$。实际用命令行统计其运算时间，结果如下两张图所示：



FIUGRE 1. 分治法时间复杂度



FIUGRE 2. 动态规划时间复杂度

从上两张图可以看出，测试结果佐证了我们的推导。