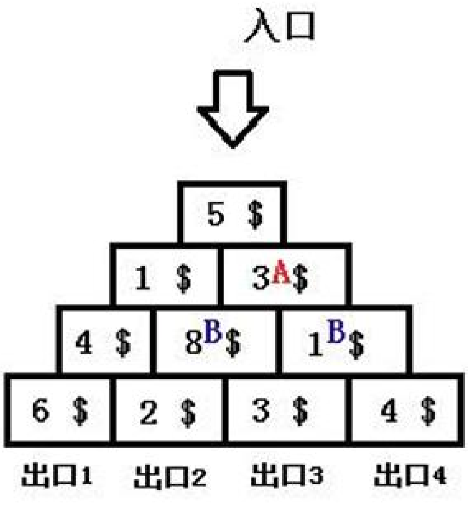
钻石金字塔动态规划解决策略

2014211314 班 201421152 袁振宇

一、问题描述

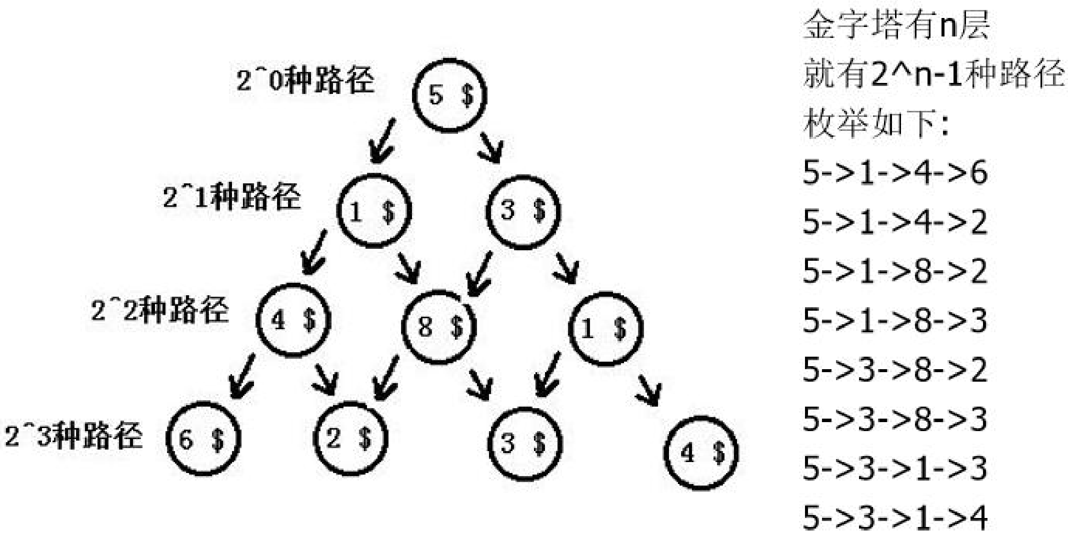
曾经有一个富翁给 heimengnan 出了一道题， 题目是这样的:有一座金字塔，金字塔的每块石头上都镶有对应的钻石，钻石可以被取下来，不同的钻石有着不同的价值，你可以将钻石换成价值相同的美元，如下：

现在你的任务是从金字塔的顶端 向金字塔的底端收集钻石，并且尽 可能收集价值高的钻石，但是只能从一块砖斜向左下或斜向右下走到 另一块砖上，如从上图从用红色 A 标记的砖走向用蓝色 B 标记的砖上。富翁希望 heimengnan 找到一个收集最高价值钻石的路线，并且把可能收集的最大价值告诉富翁。heimengnan 想了半天也没想出好方法，聪明的你能帮助他吗？



二、初步分析

暴力枚举不可行，复杂度高达 O(2^n)。



三、动态规划

用 D(x,y)表示通过第(x,y)位置时的最大钻石价值和，通过分析，显然 D(x,y)的最优路径 Path(x,y)一定包含子问题 D(x+1,y)或 D(x+1,y+1)的最优路径，否则，取 D(x+1,y)和 D(x+1,y+1)的最优路径中收获钻石价值大的那条路经加上第 x 层第 y 个位置构成的路径总价值必然答大 Path(x,y)的总价值，这与 Path(x,y)的最优性矛盾。

递推关系：

Path(x-1,y) += max(D(x,y),D(x,y+1));　x<n

D(x,y) = a(x,y);　x=n

算法描述：

```
int DPMaxValue(int pyramid[][N])            //动态规划
{
    for(int i = N-1; i > 0; i--)
        for(int j = 0; j < i; j++)
            pyramid[i-1][j] += pyramid[i][j] > pyramid[i][j+1] ? pyramid[i][j] : pyramid[i][j+1];

    return pyramid[0][0];
}
```

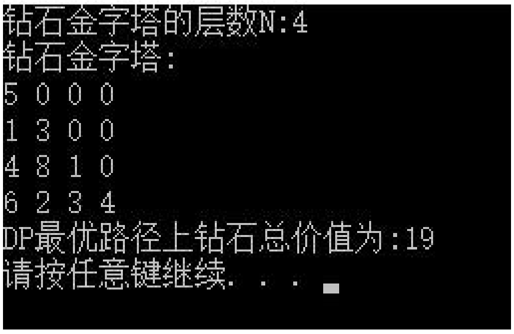
四、源代码

```cpp
#include <iostream>
#include <cstdlib>
#include <fstream>
using namespace std;

const int N = 4;
//#define Layer N

int DPMaxValue(int pyramid[][N])                //动态规划
{
    for(int i = N-1; i > 0; i--)
        for(int j = 0; j < i; j++)
            pyramid[i-1][j] += pyramid[i][j] >
            pyramid[i][j+1] ? pyramid[i][j] : pyramid[i][j+1];

    return pyramid[0][0];
}
```

```cpp
int main()
{
    fstream input;

    input.open("金字塔.txt");
    if(input.fail())    exit(0);

    cout << "钻石金字塔的层数N:" << N << endl;

    int pyramid[N][N];
    for(int i = 0; i < N; i++)
        for(int j = 0; j < N; j++)
            input >> pyramid[i][j];

    cout << "钻石金字塔:" << endl;
    for(int i = 0; i < N; i++)
    {
        for(int j = 0; j < N; j++)
            cout << pyramid[i][j] << " ";
        cout << endl;
    }
```

```cpp
    cout << "DP最优路径上钻石总价值为:" << DPMaxValue(pyramid) << endl

    input.close();

    system("pause");
    return 0;
}
```

五、测试实例



六、时间复杂度分析

本算法采用动态规划，由于 N 表示金字塔层数，而实际工作时访问的是节点数，故时间复杂度为 O(n^2)。

七、实验总结与心得体会

通过本次实验，我进一步体会到了动态规划的美妙。动态规划问题往往满足以下两个条件：

1.最优子结构性质

原问题的最优解包含了其子问题的最优解，即原 问题可以由子问题的最优解组合而成，这就使得 问题可以拆分成若干个子问题。

2.子问题重叠性质

每次产生的子问题并不总是新问题，有些子问题 被反复计算多次。

动态规划算法与分治法类似，其基本思想也是将待求解 问题分解成若干个子问题；但是经分解得到的子问题往往不是互相独立的。不同子问题的数目常常只有多项式量级。在用分治法求解时，有些子问题被重复计算了许多次，时间复杂性呈指数增长。如果能够保存已解决的子问题的答案，而在需要时再找出已求得的答案，就可以避免大量重复计算，从而得到多项式时间算法。