

题目描述

对于集合 $\{1, 2, \dots, n\}$ 的一棵二叉搜索树，定义一个节点的深度为该节点到根节点的距离，设权值为 i 的点深度为 d_i ，给定序列 p_1, p_2, \dots, p_n ，定义该二叉搜索树的**代价和**为：

$$\sum_{i=1}^n (d_i + 1) p_i$$

请你求解集合 $\{1, 2, \dots, n\}$ 所有二叉搜索树中最小的代价和。

求解思路

本题为区间动态规划问题，可以参考《算法导论》p226

具体的思路和形式与之前的矩阵链相乘问题相似

p_1

p_2

p_3

p_4

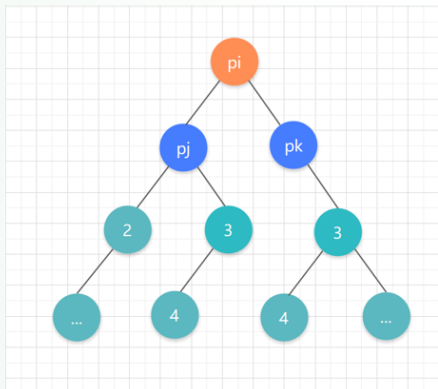
.

.

.

p_n

将这样一组数分别填入二叉树各个位置，使得其代价和最小，可以称之为最优结构

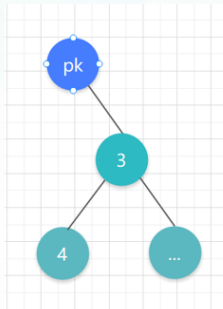
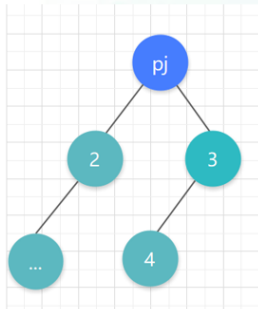
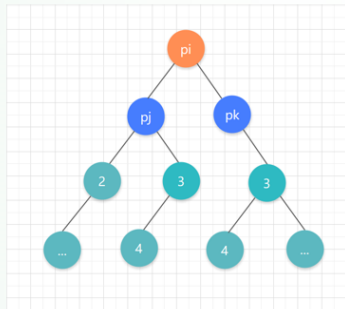


任意一棵满足最优结构的二叉树，其根节点的两颗子树也一定满足这样的结构

该二叉树的代价和 = 根节点值 + 左子树(深度+1后的)代价和 + 右子树(深度+1后的)代价和

$$\sum_{i=1}^n (d_i + 2) p_i = \sum_{i=1}^n (d_i + 1) p_i + \sum_{i=1}^n p_i$$

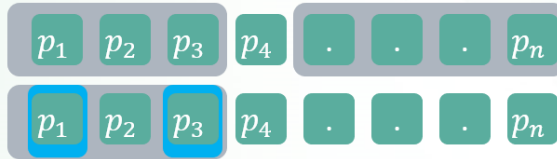
左子树(深度+1后的)代价和 = 左子树原本的代价和 + 其所有节点的值之和



区间动态规划，类似于矩阵链乘法

p_1 p_2 p_3 p_4 . . . p_n

每次计算一个区间内的值
形成二叉树的最小代价和



以区间其中一个点作为二叉树的
根节点，其结果取决于其**自身值**
以及**两侧区间**的结果

每个区间所得的最小代价和结果即
为，其中每一个点作为根节点所得
到的代价和的最小值

用一个二维数组 val 记录最小代价和， $val[i][j]$ 即表示区间 (i,j) 最小代
价和的结果；
用二维数组 sum 记录节点的值， $sum[i][j]$ 即表示区间 (i,j) 节点值的和
 p 数组记录节点的值， $p[i]$ 为第 i 个节点的值

状态转移方程：

$$Val[i][j] = \begin{cases} 0, i = j \\ \min(val[i][k] + val[k + 1][j] + \\ sum[i][k] + sum[k + 1][j] + p[k]), x \geq 0 \end{cases}$$

代码

```
#include <stdio.h>
#define maxSize 510
#define maxData 0x7fffffff

int p[maxSize];
int pSum[maxSize][maxSize];
int resSum[maxSize][maxSize];

void getMinCost(int n);

int main()
{
    int n = 0;

    scanf("%d", &n);
    for (int i = 0; i < n; ++i)
        scanf("%d", &p[i]);
    getMinCost(n);
}
```

```

    printf("%d", resSum[0][n]);

    return 0;
}

void getMinCost(int n)
{
    for (int i = 0; i <= n; ++i)
        pSum[i][i] = resSum[i][i] = 0;
    for (int i = 0; i <= n; ++i)
    {
        for (int j = i + 1; j <= n; ++j)
            pSum[i][j] = pSum[i][j - 1] + p[j - 1];
    }
    for (int l = 2; l <= n + 1; ++l)
    {
        for (int i = 0; i <= n - l + 1; ++i)
        {
            int j = i + l - 1;
            int temp = maxData, min = maxData;
            for (int k = i; k < j; ++k)
            {
                temp = resSum[i][k] + resSum[k + 1][j] +
                    pSum[i][k] + pSum[k + 1][j] + p[k];
                if (temp < min) min = temp;
            }
            resSum[i][j] = min;
        }
    }
}

```