算法设计与分析 C3-H

21377206 阮阳栋

题目描述

对于集合$\{1,2,...,n\}$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 $p_1,p_2,...,p_n$，定义该二叉搜索树的代价和为：

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

需要求解集合$\{1,2,...,n\}$所有二叉搜索树中最小的代价和，并输出任意一种最小代价的二叉搜索树结构。

题目分析

定义 $e(i,j)$ 为节点 i 到 j 的最小代价和，参考P229(15.14)，$e(i,j)$ 的递归定义如下：

$$e(i,j)=\begin{cases}0 & i=j+1\\ \min\{e(i,r-1)+e(r+1,j)+w(i,j)\} & i\le r\le j\end{cases}$$

其中$w(i,j)=\sum_{k=i}^{k=j}p_k=\sum_{k=0}^{k=j}p_k-\sum_{k=0}^{k=i-1}p_k$（定义$p_0=0$），可以用前缀和解决效率问题，用数组 `ll sump[]` 存储前缀和。

```
ll sumpp = 0;
for (int i=1;i<=n;i++) {
    cin >> p[i];
    sumpp += p[i];
    sump[i] = sumpp;
}
```

因为要输出树的连结情况，可以用数组 `int pos[][]` 来存储两个节点之间选取的根，用数组 `int tr[][]` 递归存储左节点和右节点：

```
int mktr(int l0, int r0){ //返回根节点序号
    if (l0 == r0) { //当前节点已经不可能有左右子节点
        tr[l0][0] = -1, tr[l0][1] = -1;
        return l0;
    }
    if (l0 > r0) return -1; //无节点
    int y = pos[l0][r0]; //存储根
    //对根节点递归存储左右子节点
    tr[y][0] = mktr(l0,y-1), tr[y][1] = mktr(y+1,r0);
    return y; //返回根
}
```

完整代码

```cpp
#include <bits/stdc++.h>
using namespace std;
#define ll long long

ll p[510], sump[510];
int n, tr[510][2], pos[510][510];

ll cache[510][510], vis[510][510];
ll dp(int l0, int r0){
    if (l0 > r0) return 0;
    if (l0 == r0) return p[l0];
    if (vis[l0][r0]) return cache[l0][r0];
    ll r = dp(l0,l0-1)+dp(l0+1,r0)+sump[r0]-sump[l0-1];
    int gg = l0;
    for (int k=l0+1;k<=r0;k++){
        ll u = dp(l0,k-1)+dp(k+1,r0)+sump[r0]-sump[l0-1];
        if (u < r) r = u, gg = k;
    }
    pos[l0][r0] = gg, vis[l0][r0] = 1, cache[l0][r0] = r;
    return r;
}

int mktr(int l0, int r0){
    if (l0 == r0) {
        tr[l0][0] = -1, tr[l0][1] = -1;
        return l0;
    }
    if (l0 > r0) return -1;
    tr[y][0] = mktr(l0,y-1), tr[y][1] = mktr(y+1,r0);
    return pos[l0][r0];
}

int main(){

    cin >> n;
    ll sumpp = 0;
    for (int i=1;i<=n;i++)
        cin >> p[i], sumpp += p[i], sump[i] = sumpp;

    cout << dp(1, n) << endl;
    mktr(1, n);
    for (int i=1;i<=n;i++)
        cout << tr[i][0] << " " << tr[i][1] << endl;

    return 0;
}
```