M-Monster Hunter

M-Monster Hunter_第 45 届国际大学生程序设计竞赛（ICPC）亚洲区域赛（南京）_(nowcoder.com)

题目简介

1. 很多个monster由父子关系，组成了一棵树。如果想要消灭一个monster。首先要将其父亲消灭。
2. 每一个monster都有一个生命值。
3. 消灭一个monster的花销如下：

$$sum = hp_i + \sum hp_{j:j必须是活着的妖怪} \tag{1}$$

可以对整颗子树，使用m次魔术。使用0花销选择消灭掉一个妖怪。

solve

状态定义

定义状态$dp[i][j]$表示在i子树下使用j次魔术，消灭该子树上所有点的花费。发现迁移不了，因为转移过程中，我们同时也要关注当前的儿子有没有被消灭掉。

1. 考虑加多一维，表示当前子树的根有没有使用魔法。0，表示没有被删除。1表示已经被删除了。

状态转移方程

经典的一个树形背包问题：

$$\begin{gathered} 关于dp[u][0..m][0] \\ dp[u][i][0] + dp[v][j][1] -> (min)temp[i+j][0]; \\ dp[u][i][0] + dp[v][j][0] -> (min)temp[i+j][0]; \end{gathered} \tag{2}$$

$$\begin{gathered} 关于dp[u][0..m][1] \\ dp[u][i][1] + dp[v][j][0] -> (min)temp[i+j][1] \\ dp[u][i][1] + dp[v][j][1] + a[v] -> (min)temp[i+j][1] \end{gathered} \tag{3}$$

$$\begin{gathered} 迁移完成之后： \\ dp[u][1..i][1] += a[u]; \end{gathered}$$

code

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

const ll oo = 1LL << 60;
const int N = 2E3 + 10;
vector<int>son[N];
int size[N];
ll dp[N][N][2] , a[N];

// 0 .表示什么？有没有使用魔法。有没有消灭掉

void dfs(int u) {
    //怎么转移。还出现了其它一些问题。那么到底是什么问题呢？
```

```cpp
    //应该怎么初始化/
    //状态定义是什么？完全没有头绪。
    dp[u][1][0] = 0;
    dp[u][0][0] = oo;
    dp[u][1][1] = oo;
    dp[u][0][1] = a[u];
    size[u] = 1;

    for (auto v : son[u]) {
        dfs(v);
        vector<vector<ll>> tmp(size[u] + size[v] + 1 , vector<ll>(2 , oo));
        for (int i = 0; i <= size[u]; i++)
            for (int j = 0; j <= size[v]; j++) {
                tmp[i + j][0] = min(dp[u][i][0] + min(dp[v][j][0] , dp[v][j][1]) , tmp[i + j][0]);
                tmp[i + j][1] = min(dp[u][i][1] + dp[v][j][0] , tmp[i + j][1]);
                tmp[i + j][1] = min(dp[u][i][1] + dp[v][j][1] + a[v] , tmp[i + j][1]);
            }
        size[u] += size[v];
        for (int i = 0; i <= size[u]; i++) {
            dp[u][i][0] = tmp[i][0];
            dp[u][i][1] = tmp[i][1];
        }
    }
}

void work(int testNo)
{
    int n;
    cin >> n ;
    for (int i = 1; i <= n; i++) son[i].clear();
    for (int i = 2; i <= n; i++) {
        int f; cin >> f;
        son[f].push_back(i);
    }
    for (int i = 1; i <= n; i++)cin >> a[i];
    dfs(1);
    for (int i = 0; i <= n; i++) {
        cout << min(dp[1][i][0] , dp[1][i][1]) << " \n"[i == n];
    }
}


int main()
{
    ios::sync_with_stdio(false);
    cin.tie(0);

    int t; cin >> t;
    for (int i = 1; i <= t; i++)work(i);
}

/* stuff you should look for
```

```
* int overflow, array bounds
* special cases (n=1?)
* do smth instead of nothing and stay organized
* WRITE STUFF DOWN
* DON'T GET STUCK ON ONE APPROACH
*/
```

生长思考

1. 自己一些偏见。浮出了水面。

   总是在想，做树形dp时候。根节点总是要最后处理。而这里，根节点的贡献一开始就已经选择了。

   偏见来源： 树上背包.md （树上背包1）

   本质上都是树形背包问题，但是为什么对于根节点的贡献却有两种相反的处理?

2. 在dp过程中。dp[u]有滚动数组的意味。它的子问题是。根节点加上前i个儿子子树结构中。分配j魔法的合法结构的最优解。

3. 考虑初始化，dp数组最终意义是我们一开始定义的。但是为了合并儿子节点的dp数组。计算过程中， 存储结构可能呈现不一样的定义。如上树上背包1.合并过程中的意义是，将j个节点分配个0...i儿子的连通块的最大权值和。

   在合并完成之后，再进行处理。然后回到了最初的定义。