树形背包1

[树上背包1 - 题目 - Daimayuan Online Judge]()

[F - Components.md]()

问题简述：

1. 上述树形背包问题

   选择节点数量恰好为m的连通块的最大节点权值和。

2. 上笔记中的例题

   连通块数量为m的方案个数。

事实上都是树形背包问题。在子树中选择独立结构。

solve

1. 定义$f_{i,j}$表示根为i的子树中，选择了根并且节点数量为j的连通块， 的最大权值。
2. 初始化。
   1. 对于所有方案初始化为0.
3. 转移。
   1. 合并所有儿子的dp数组。
   2. 对于一个节点合并完成后统计答案。

看上去复杂度为$O(N^3)$
对于节点i， 有意义的选择数量的值域只可能是（1....size(i)）.
只求有意义的合并数组。

但是如果关注每一个点对，发现它们只合并一次。而所有子树的情形下，都有一个dp元素与之中匹配。

code

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

const int N = 2010 , inf = - 1E9;
vector<int> e[N];
int w[N] , sz[N] , n ,  q;
ll f[N][N];

void dfs(int u) {
    sz[u] = 0;
    for (auto v : e[u]) {
        dfs(v);
        vector<ll> temp(sz[u] + sz[v] + 1 , inf);
        for (int i = 0; i <= sz[u]; i++)
            for (int j = 0; j <= sz[v]; j++) {
                temp[i + j] = max(temp[i + j] , f[u][i] + f[v][j]);
            }
        sz[u] += sz[v];
        for (int i = 0; i <= sz[u]; i++) {
```

```
                f[u][i] = temp[i];
            }
        }
        sz[u] ++;
        for (int i = sz[u]; i >= 1 ; i--) {
            f[u][i] = f[u][i - 1] +  w[u];
        }
}

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

    cin >> n >> q;
    for (int i = 2; i <= n; i++) {
        int x; cin >> x;
        e[x].push_back(i);
    }
    for (int i = 1; i <= n; i++) {
        cin >> w[i];
    }
    dfs(1);
    while (q--) {
        int x, y;
        cin >> x >> y;
        cout << f[x][y] << '\n';
    }
}
```

生长思考：

1. 处理树的根节点：
    1. 计算的过程是，利用子问题的最优结构子问题的解。
    2. 对于根节点，如果开始就处理。设计$f_{u,1} = w[u]$。出现问题。因为只关注，迁移过程中用到的为1的节点数。迁移过程中，指标函数的解可能不再含有根。
    3. 处理方法是，记录除根外的结构。最后补充上。
2. 计算dp数组过程中。不同阶段，该数据具有不同的意义：
    1. 0 ... i个节点， 选择j个节点连通块的最大权值和。
    2. 表示整一颗子树选择了j个节点的最大权值和。

补充：

另外一种写法：关于迁移时初始化的方式不同。

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

const int N = 2010 , inf = - 1E9;
vector<int> e[N];
int w[N] , sz[N] , n ,  q;
ll f[N][N];
```

```cpp

void dfs(int u) {
    sz[u] = 1;
    f[u][0] = inf;
    f[u][1] = w[u];
    for (auto v : e[u]) {
        dfs(v);
        vector<ll> temp(sz[u] + sz[v] + 1 , inf);
        for (int i = 0; i <= sz[u]; i++)
            for (int j = 0; j <= sz[v]; j++) {
                temp[i + j] = max(temp[i + j] , f[u][i] + f[v][j]);
            }
        sz[u] += sz[v];
        for (int i = 0; i <= sz[u]; i++) {
            f[u][i] = temp[i];
        }
    }
    f[u][0] = 0;
}

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

    cin >> n >> q;
    for (int i = 2; i <= n; i++) {
        int x; cin >> x;
        e[x].push_back(i);
    }
    for (int i = 1; i <= n; i++) {
        cin >> w[i];
    }
    dfs(1);
    while (q--) {
        int x, y;
        cin >> x >> y;
        cout << f[x][y] << '\n';
    }
}
```

树形背包2

[树上背包2 - 题目 - Daimayuan Online Judge]

发散点

- 改变数据特点。

  n - > 500000

  m - > 100

solve

使用和上面树形背包1的问题解决技巧。同时基于问题，只关注规模的小于等于m的规模的问题。

状态迁移还是一样的。

平方复杂度的树形dp.md 类似这一个证明方法，可以完成对问题的证明。

code

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

const int N = 1E6 + 10;
const int inf = -1E9;


vector<int> e[N];
int w[N] , siz[N];
int f[N][110];
int n;

void dfs(int u) {
    siz[u] = 0;
    //然后应该怎么办呢？
    for (auto v : e[u]) {
        dfs(v);
        vector<int> temp (min(siz[u] + siz[v] + 1 , 101) , inf );
        for (int i = 0; i <= siz[u] && i <= 100; i++)
            for (int j = 0; j <= siz[v] && (i + j) <= 100 ; j++) {
                temp[i + j] = max(temp[i + j] , f[u][i] + f[v][j]);
            }
        siz[u] += siz[v];
        for (int i = 0; i < (int)temp.size(); i++) {
            f[u][i] = temp[i];
        }
    }
    siz[u]++;
    for (int i = min(siz[u] , 100); i >= 1; i--) {
        f[u][i] = f[u][i - 1] + w[u];
    }
}


int main()
{
    ios::sync_with_stdio(false);
    cin.tie(0);
    int  q;
    cin >> n >> q;
    for (int i = 2; i <= n; i++) {
        int x; cin >> x;
        e[x].push_back(i);
    }
    for (int i = 1; i <= n; i++) {
```

```
        cin >> w[i];
    }
    dfs(1);
    while (q--) {
        int  u , m;
        cin >> u >> m;
        cout << f[u][m] << '\n';
    }
}

/* 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
*/
```

树上背包3

[树上背包3 - 题目 - Daimayuan Online Judge]

就是经典的树上背包问题。
严格控制复杂度为$O(N \times M)$

solve

如果参照之前的方法。（或者是一般泛化物品的方法)由于m比较大。并且无法对合并dp数组的长度进行优化。所以用前面的思想行不通。

以下是一种很神奇的方法。利用dfs序。将问题转换成一个线性问题。如下：

1. 求出当前树的dfs序列。
2. 定义r_i表示跳过i节点为根节点的子树，的第一个节点。
3. 定义$f_{i,j}$表示考虑dfs序中$[i, n]$这一段的节点。选的重量和不超过j的点集的最大权值和，并且要求这个点集不存咋一个点选了。但它在点集中的祖先没有被选的i情况。

状态转移方程：

$$f_{i,j} = max(f_{r_i,j}, f_{i+1,j-w_i} + v_i) \tag{1}$$

5. 初始化： f[n + 1][1...j] = -oo;

code

```
#include <bits/stdc++.h>
using namespace std;

const int N=1010, M = 10010;
const int inf = 1<<29;

vector<int> son[N];

int n, m, tot, l[N], r[N], id[N];
int dp[N][M], a[N], w[N];

void dfs(int u) {
```

```
    l[u] = ++tot;
    id[tot] = u;
    for (auto v : son[u]) {
        dfs(v);
    }
    r[u] = tot;
}

int main() {
    scanf("%d%d",&n, &m);
    for (int i = 2; i <= n; i++) {
        int f;
        scanf("%d", &f);
        son[f].push_back(i);
    }
    for (int i = 1; i <= n; i++)
        scanf("%d", &a[i]);
    for (int i = 1; i <= n; i++)
        scanf("%d", &w[i]);
    dfs(1);
    for (int j = 1; j <= m; j++) dp[n + 1][j] = -inf;
    for (int i = n; i >= 1; i--) {
        int u = id[i];
        for (int j = 0; j <= m; j++) {
            dp[i][j] = dp[r[u] + 1][j];
            if (j >= w[u])
                dp[i][j] = max(dp[i][j], dp[i + 1][j - w[u]] + a[u]);
        }
    }
    for (int i = 0; i <= m; i++) {
        if (dp[1][i] >= 0) printf("%d\n", dp[1][i]);
        else printf("0\n");
    }
}
```

生长思考

1. 第一次碰到的关于，将树的dfs序找出来。然后在利用dfs序处理问题的图论问题。（可能强连通分量，相关定理算其中一个）。
2. 该问题的方法思想， 比较特殊少见。

总结以及拓展

1. 距离相关问题，进行树形dp时也会出现相关的结论。
    1. 在树上找一个连通块。满足任意两点之间的距离不超过d。使得权值和最大，权值可能为负数。
    2. 在树上找一个点集，满足任意两点之间的距离不小于d。使得权值和最大，权值可能为负数。
2. 合并的dp域，和子树的节点数相关。那么就会有类似的结论。