

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

```
1  #include<bits/stdc++.h>
2  using namespace std;
3  using ll = long long;
4
5  const int N = 2010 , inf = - 1E9;
6  vector<int> e[N];
7  int w[N] , sz[N] , n , q;
8  ll f[N][N];
9
10 void dfs(int u) {
11     sz[u] = 0;
12     for (auto v : e[u]) {
13         dfs(v);
14         vector<ll> temp(sz[u] + sz[v] + 1 , inf);
15         for (int i = 0; i <= sz[u]; i++)
16             for (int j = 0; j <= sz[v]; j++) {
17                 temp[i + j] = max(temp[i + j] , f[u][i] + f[v][j]);
18             }
19         sz[u] += sz[v];
20         for (int i = 0; i <= sz[u]; i++) {
```

```

21         f[u][i] = temp[i];
22     }
23 }
24 sz[u] ++;
25 for (int i = sz[u]; i >= 1 ; i--) {
26     f[u][i] = f[u][i - 1] + w[u];
27 }
28 }
29
30 int main() {
31     ios::sync_with_stdio(false);
32     cin.tie(0);
33
34     cin >> n >> q;
35     for (int i = 2; i <= n; i++) {
36         int x; cin >> x;
37         e[x].push_back(i);
38     }
39     for (int i = 1; i <= n; i++) {
40         cin >> w[i];
41     }
42     dfs(1);
43     while (q--) {
44         int x, y;
45         cin >> x >> y;
46         cout << f[x][y] << '\n';
47     }
48 }

```

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。

```

1  #include<bits/stdc++.h>
2  using namespace std;
3  using ll = long long;
4
5  const int N = 2010 , inf = - 1E9;
6  vector<int> e[N];
7  int w[N] , sz[N] , n , q;
8  ll f[N][N];

```

```

9
10 void dfs(int u) {
11     sz[u] = 1;
12     f[u][0] = inf;
13     f[u][1] = w[u];
14     for (auto v : e[u]) {
15         dfs(v);
16         vector<ll> temp(sz[u] + sz[v] + 1, inf);
17         for (int i = 0; i <= sz[u]; i++)
18             for (int j = 0; j <= sz[v]; j++) {
19                 temp[i + j] = max(temp[i + j], f[u][i] + f[v][j]);
20             }
21         sz[u] += sz[v];
22         for (int i = 0; i <= sz[u]; i++) {
23             f[u][i] = temp[i];
24         }
25     }
26     f[u][0] = 0;
27 }
28
29 int main() {
30     ios::sync_with_stdio(false);
31     cin.tie(0);
32
33     cin >> n >> q;
34     for (int i = 2; i <= n; i++) {
35         int x; cin >> x;
36         e[x].push_back(i);
37     }
38     for (int i = 1; i <= n; i++) {
39         cin >> w[i];
40     }
41     dfs(1);
42     while (q--) {
43         int x, y;
44         cin >> x >> y;
45         cout << f[x][y] << '\n';
46     }
47 }

```

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

```
1  #include<bits/stdc++.h>
2  using namespace std;
3  using ll = long long;
4
5  const int N = 1E6 + 10;
6  const int inf = -1E9;
7
8
9  vector<int> e[N];
10 int w[N] , siz[N];
11 int f[N][110];
12 int n;
13
14 void dfs(int u) {
15     siz[u] = 0;
16     //然后应该怎么办呢?
17     for (auto v : e[u]) {
18         dfs(v);
19         vector<int> temp (min(siz[u] + siz[v] + 1 , 101) , inf );
20         for (int i = 0; i <= siz[u] && i <= 100; i++)
21             for (int j = 0; j <= siz[v] && (i + j) <= 100 ; j++) {
22                 temp[i + j] = max(temp[i + j] , f[u][i] + f[v][j]);
23             }
24         siz[u] += siz[v];
25         for (int i = 0; i < (int)temp.size(); i++) {
26             f[u][i] = temp[i];
27         }
28     }
29     siz[u]++;
30     for (int i = min(siz[u] , 100); i >= 1; i--) {
31         f[u][i] = f[u][i - 1] + w[u];
32     }
33 }
34
35
36 int main()
37 {
38     ios::sync_with_stdio(false);
39     cin.tie(0);
40     int q;
41     cin >> n >> q;
42     for (int i = 2; i <= n; i++) {
43         int x; cin >> x;
44         e[x].push_back(i);
45     }
46     for (int i = 1; i <= n; i++) {
```

```

47     cin >> w[i];
48 }
49 dfs(1);
50 while (q--) {
51     int u, m;
52     cin >> u >> m;
53     cout << f[u][m] << '\n';
54 }
55 }
56
57 /* stuff you should look for
58 * int overflow, array bounds
59 * special cases (n=1?)
60 * do smth instead of nothing and stay organized
61 * WRITE STUFF DOWN
62 * DON'T GET STUCK ON ONE APPROACH
63 */

```

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情况。

状态转移方程：

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

5. 初始化： $f[n+1][1\dots j] = -\infty$;

code

```

1  #include <bits/stdc++.h>
2  using namespace std;
3
4  const int N=1010, M = 10010;
5  const int inf = 1<<29;
6
7  vector<int> son[N];
8
9  int n, m, tot, l[N], r[N], id[N];
10 int dp[N][M], a[N], w[N];
11
12 void dfs(int u) {

```

```

13     l[u] = ++tot;
14     id[tot] = u;
15     for (auto v : son[u]) {
16         dfs(v);
17     }
18     r[u] = tot;
19 }
20
21 int main() {
22     scanf("%d%d", &n, &m);
23     for (int i = 2; i <= n; i++) {
24         int f;
25         scanf("%d", &f);
26         son[f].push_back(i);
27     }
28     for (int i = 1; i <= n; i++)
29         scanf("%d", &a[i]);
30     for (int i = 1; i <= n; i++)
31         scanf("%d", &w[i]);
32     dfs(1);
33     for (int j = 1; j <= m; j++) dp[n + 1][j] = -inf;
34     for (int i = n; i >= 1; i--) {
35         int u = id[i];
36         for (int j = 0; j <= m; j++) {
37             dp[i][j] = dp[r[u] + 1][j];
38             if (j >= w[u])
39                 dp[i][j] = max(dp[i][j], dp[i + 1][j - w[u]] + a[u]);
40         }
41     }
42     for (int i = 0; i <= m; i++) {
43         if (dp[1][i] >= 0) printf("%d\n", dp[1][i]);
44         else printf("0\n");
45     }
46 }

```

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。

