树形 DP

树形 DP，即在树上进行的 DP。由于树固有的递归性质，树形 DP 一般都是递归进行的。

基础

以下面这道题为例，介绍一下树形 DP 的一般过程。

例题 洛谷 P1352 没有上司的舞会

某大学有 n 个职员，编号为 $1 \sim N$。他们之间有从属关系，也就是说他们的关系就像一棵以校长为根的树，父结点就是子结点的直接上司。现在有个周年庆宴会，宴会每邀请来一个职员都会增加一定的快乐指数 a_i，但是呢，如果某个职员的直接上司来参加舞会了，那么这个职员就无论如何也不肯来参加舞会了。所以，请你编程计算，邀请哪些职员可以使快乐指数最大，求最大的快乐指数。

我们设 $f(i,0/1)$ 代表以 i 为根的子树的最优解（第二维的值为 0 代表 i 不参加舞会的情况，1 代表 i 参加舞会的情况）。

对于每个状态，都存在两种决策（其中下面的 x 都是 i 的儿子）：

- 上司不参加舞会时，下属可以参加，也可以不参加，此时有 $f(i,0) = \sum \max\{f(x,1), f(x,0)\}$；
- 上司参加舞会时，下属都不会参加，此时有 $f(i,1) = \sum f(x,0) + a_i$。

我们可以通过 DFS，在返回上一层时更新当前结点的最优解。

```cpp
#include <algorithm>
#include <iostream>
using namespace std;

struct edge {
  int v, next;
} e[6005];

int head[6005], n, cnt, f[6005][2], ans, is_h[6005], vis[6005];

void addedge(int u, int v) {  // 建图
  e[++cnt].v = v;
  e[cnt].next = head[u];
  head[u] = cnt;
}

void calc(int k) {
```

```cpp
  vis[k] = 1;
  for (int i = head[k]; i; i = e[i].next) {  // 枚举该结点的每个子结点
    if (vis[e[i].v]) continue;
    calc(e[i].v);
    f[k][1] += f[e[i].v][0];
    f[k][0] += max(f[e[i].v][0], f[e[i].v][1]);  // 转移方程
  }
  return;
}

int main() {
  cin.tie(nullptr)->sync_with_stdio(false);
  cin >> n;
  for (int i = 1; i <= n; i++) cin >> f[i][1];
  for (int i = 1; i < n; i++) {
    int l, k;
    cin >> l >> k;
    is_h[l] = 1;
    addedge(k, l);
  }
  for (int i = 1; i <= n; i++)
    if (!is_h[i]) {  // 从根结点开始DFS
      calc(i);
      cout << max(f[i][1], f[i][0]);
      return 0;
    }
}
```

通常，树形 DP 状态一般都为当前节点的最优解。先 DFS 遍历子树的所有最优解，然后向上传递给子树的父节点来转移，最终根节点的值即为所求的最优解。

习题

- HDU 2196 Computer
- POJ 1463 Strategic game
- [POI2014]FAR-FarmCraft

树上背包

树上的背包问题，简单来说就是背包问题与树形 DP 的结合。

例题 洛谷 P2014 CTSC1997 选课

现在有 n 门课程，第 i 门课程的学分为 a_i，每门课程有零门或一门先修课，有先修课的课程需要先学完其先修课，才能学习该课程。

一位学生要学习 m 门课程，求其能获得的最多学分数。

$n, m \leq 300$

每门课最多只有一门先修课的特点，与有根树中一个点最多只有一个父亲结点的特点类似。

因此可以想到根据这一性质建树，从而所有课程组成了一个森林的结构。为了方便起见，我们可以新增一门 0 学分的课程（设这个课程的编号为 0），作为所有无先修课课程的先修课，这样我们就将森林变成了一棵以 0 号课程为根的树。

我们设 $f(u,i,j)$ 表示以 u 号点为根的子树中，已经遍历了 u 号点的前 i 棵子树，选了 j 门课程的最大学分。

转移的过程结合了树形 DP 和 背包 DP 的特点，我们枚举 u 点的每个子结点 v，同时枚举以 v 为根的子树选了几门课程，将子树的结果合并到 u 上。

记点 x 的儿子个数为 s_x，以 x 为根的子树大小为 siz_x，可以写出下面的状态转移方程：

$$f(u,i,j) = \max_{v, k \leq j, k \leq siz_v} f(u, i-1, j-k) + f(v, s_v, k)$$

注意上面状态转移方程中的几个限制条件，这些限制条件确保了一些无意义的状态不会被访问到。

f 的第二维可以很轻松地用滚动数组的方式省略掉，注意这时需要倒序枚举 j 的值。

可以证明，该做法的时间复杂度为 $O(nm)$[1]。

参考代码

```cpp
#include <algorithm>
#include <iostream>
#include <vector>
using namespace std;
int f[305][305], s[305], n, m;
vector<int> e[305];

int dfs(int u) {
  int p = 1;
  f[u][1] = s[u];
  for (auto v : e[u]) {
    int siz = dfs(v);
    // 注意下面两重循环的上界和下界
    // 只考虑已经合并过的子树，以及选的课程数超过 m+1 的状态没有意义
    for (int i = min(p, m + 1); i; i--)
      for (int j = 1; j <= siz && i + j <= m + 1; j++)
        f[u][i + j] = max(f[u][i + j], f[u][i] + f[v][j]);  // 转移方程
    p += siz;
  }
  return p;
}

int main() {
  cin.tie(nullptr)->sync_with_stdio(false);
  cin >> n >> m;
  for (int i = 1; i <= n; i++) {
    int k;
    cin >> k >> s[i];
    e[k].push_back(i);
  }
  dfs(0);
  cout << f[0][m + 1];
  return 0;
}
```

习题

- 「CTSC1997」选课
- 「JSOI2018」潜入行动
- 「SDOI2017」苹果树
- 「Codeforces Round 875 Div. 1」Problem D. Mex Tree

换根 DP

> 树形 DP 中的换根 DP 问题又被称为二次扫描，通常不会指定根结点，并且根结点的变化会对一些值，例如子结点深度和、点权和等产生影响。

通常需要两次 DFS，第一次 DFS 预处理诸如深度，点权和之类的信息，在第二次 DFS 开始运行换根动态规划。

接下来以一些例题来带大家熟悉这个内容。

例题 [POI2008]STA-Station

给定一个 n 个点的树，请求出一个结点，使得以这个结点为根时，所有结点的深度之和最大。

不妨令 u 为当前结点，v 为当前结点的子结点。首先需要用 s_i 来表示以 i 为根的子树中的结点个数，并且有 $s_u = 1 + \sum s_v$。显然需要一次 DFS 来计算所有的 s_i，这次的 DFS 就是预处理，我们得到了以某个结点为根时其子树中的结点总数。

考虑状态转移，这里就是体现 " 换根 " 的地方了。令 f_u 为以 u 为根时，所有结点的深度之和。

$f_v \leftarrow f_u$ 可以体现换根，即以 u 为根转移到以 v 为根。显然在换根的转移过程中，以 v 为根或以 u 为根会导致其子树中的结点的深度产生改变。具体表现为：

- 所有在 v 的子树上的结点深度都减少了一，那么总深度和就减少了 s_v；
- 所有不在 v 的子树上的结点深度都增加了一，那么总深度和就增加了 $n - s_v$；

根据这两个条件就可以推出状态转移方程 $f_v = f_u - s_v + n - s_v = f_u + n - 2 \times s_v$。

于是在第二次 DFS 遍历整棵树并状态转移 $f_v = f_u + n - 2 \times s_v$，那么就能求出以每个结点为根时的深度和了。最后只需要遍历一次所有根结点深度和就可以求出答案。

参考代码

```cpp
#include <iostream>
using namespace std;

int head[1000010 << 1], tot;
long long n, sz[1000010], dep[1000010];
long long f[1000010];

struct node {
  int to, next;
} e[1000010 << 1];

void add(int u, int v) {  // 建图
  e[++tot] = {v, head[u]};
  head[u] = tot;
}

void dfs(int u, int fa) {  // 预处理dfs
  sz[u] = 1;
  dep[u] = dep[fa] + 1;
  for (int i = head[u]; i; i = e[i].next) {
    int v = e[i].to;
    if (v != fa) {
      dfs(v, u);
      sz[u] += sz[v];
    }
  }
}

void get_ans(int u, int fa) {  // 第二次dfs换根dp
  for (int i = head[u]; i; i = e[i].next) {
    int v = e[i].to;
    if (v != fa) {
      f[v] = f[u] - sz[v] * 2 + n;
      get_ans(v, u);
    }
  }
}

int main() {
  cin.tie(nullptr)->sync_with_stdio(false);
  cin >> n;
  int u, v;
  for (int i = 1; i <= n - 1; i++) {
    cin >> u >> v;
    add(u, v);
    add(v, u);
  }
  dfs(1, 1);
  for (int i = 1; i <= n; i++) f[1] += dep[i];
```

```
  get_ans(1, 1);
  long long int ans = -1;
  int id;
  for (int i = 1; i <= n; i++) {  // 统计答案
    if (f[i] > ans) {
      ans = f[i];
      id = i;
    }
  }
  cout << id << '\n';
  return 0;
}
```

习题

- Atcoder Educational DP Contest, Problem V, Subtree
- Educational Codeforces Round 67, Problem E, Tree Painting
- POJ 3585 Accumulation Degree
- [USACO10MAR]Great Cow Gathering G
- CodeForce 708C Centroids

参考资料与注释

1. 子树合并背包类型的 dp 的复杂度证明 - LYD729 的 CSDN 博客 ↩

本页面最近更新：2025/7/13 17:32:21，更新历史

发现错误？想一起完善？ 在 GitHub 上编辑此页！

本页面贡献者：StudyingFather, H-J-Granger, Ir1d, NachtgeistW, countercurrent-time, Early0v0, Enter-tainer, ShaoChenHeng, sshwy, aaron20100919, AngelKitty, CCXXXI, cjsoft, diauweb, ezoixx130, GekkaSaori, greyqz, Henry-ZHR, Konano, LovelyBuggies, lychees, Makkiy, mgt, minghu6, ouuan, P-Y-Y, PotassiumWings, SamZhangQingChuan, Suyun514, Tiphereth-A, weiyong1024, amakerlife, billchenchina, GavinZhengOI, Gesrua, isdanni, kenlig, ksyx, kxccc, Marcythm, Peanut-Tang, qz-cqy, ShizuhaAki, SukkaW, thredreams, widsnoy, Xeonacid