# 树形动态规划

河南省实验中学信息技术组

2025 年 07 月 04 日

# 树形动态规划

- 树形动态规划是在树上进行递推的动态规划模型，因为树上无环，所以树上动态规划易满足无后效性原则。
- 在树上设计动态规划算法时，一般从叶子结点到根结点的顺序进行递推，即按照结点深度最为动态规划的阶段。
- 在状态设计过程中，第一维通常是结点编号，状态存储的一般是以该结点为根的子树的相关值。
- 在动态规划的实现时，通常采用递归的方式。对于每个结点 $x$，先在它的每个子结点上进行递归，在回溯时，实现从子结点到结点 $x$ 的状态转移。
- 树形动态规划的时间复杂度一般为 O($N\times$ 每个结点递推平均复杂度)，$N$ 为树的结点个数。

- 例如，求解树的深度。
- 定义 $f(x)$ 表示以 $x$ 为根的子树的高度。
- 初始状态：$f(x)=0$，$x$ 为叶子结点。
- 状态转移方程：

$$f(x)=\max_{y\in Son(x)}\{f(y)\}$$

```cpp
void dp(int x)
{
    f[x] = 0;
    for(int i = head[x]; i; i = nxt[i])
    {
        int y = ver[i];
        dp(y);
        f[x] = max(f[x], f[y]);
    }
}
// 求以 1 为根的树的高度
dp(1);
cout << f[1] << endl;
```

# 【例】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 $h_x$，但是没有职员愿意直接和上司一起参会。主办方希望邀请一部分职员参会，使得所有参会职员的快乐指数总和最大，求这个最大值。

# 【例】没有上司的舞会

**【输入格式】**

第一行一个整数 $n(1 \leq n \leq 6000)$，表示职员数。

接下来一行 $n$ 个整数，表示职员的快乐指数 $(-128 \leq h_i \leq 127)$。

接下俩 $n-1$ 行，每行两个整数 $x, y$，表示 $x$ 是 $y$ 的上司。

**【输出格式】**

一行一个整数，表示最大的快乐指数。

**【样例输入】**

```
7
1 1 1 1 1 1 1
1 3
2 3
6 4
7 4
4 5
3 5
```

**【样例输出】**

```
5
```

# 【例】没有上司的舞会

- 状态：因为每个职员都有参加或不参加的可能，所以
  - 定义 $f(x,0)$ 表示从以 $x$ 为根的子树中邀请一部分职员参加，并且 $x$不参加舞会时，快乐指数的总和的最大值。
  - 定义 $f(x,1)$ 表示从以 $x$ 为根的子树中邀请一部分职员参加，并且 $x$参加舞会时，快乐指数的总和的最大值。
- 状态转移方程：
  - 如果 $x$ 不参与舞会，则 $x$ 的直接下属可以参加也可以不参加，故

$$f(x,0)=\sum_{y\in Son(x)}\max\{f(y,0),f(y,1)\}$$

  - 如果 $x$ 参与舞会，则 $x$ 的直接下属必然不会参加，故

$$f(x,1)=h_x+\sum_{y\in Son(x)}f(y,0)$$

# 【例】没有上司的舞会

- 初始状态：$f(x,0)=0, f(x,1)=h_x$。
- 目标状态：$\max\{f(1,0), f(1,1)\}$，1 号结点为根结点。
- 时间复杂度：$O(N)$。

```
void dp(int x)
{
    f[x][0] = 0;
    f[x][1] = h[x];
    for(int i = head[x]; i; i = nxt[i])
    {
        int y = ver[i];
        dp(y);
        // 上司不参加，职员可以选择参加或不参加
        f[x][0] += max(f[y][0], f[y][1]);
        // 上次参加，职员只能不参加
        f[x][1] += f[y][0];
    }
}
```

# 【例】完美的服务

网络由 $n$ 台计算机组成，这些计算机通过 $n-1$ 个通信链路连接，使得任意两台计算机都可以通过唯一的路由进行通信。若两台计算机之间有通信链路，则称它们相邻。计算机的邻居是与其相邻的一组计算机。

现在需要选择一些计算机作为服务器，服务器可以为其所有邻居都提供服务。若每台客户机 (非服务器) 都只由一台服务器提供服务，则网络中的一组服务器就形成了完美服务，形成完美服务的最小服务器数叫作完美服务数。现在请你求出这个完美服务数。

【输入格式】

第一行一个正整数 $n(n \leq 10^4)$，表示计算机数，计算机编号为 $1 \sim n$。

接下来 $n-1$ 行，每行两个整数 $x, y$，表示计算机 $x$ 和 $y$ 之间有通信链路。

【输出格式】

一行一个整数，表示完美服务数。

# 【例】完美的服务

**【样例输入】**

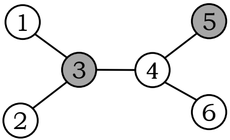
```
6
1 3
2 3
3 4
4 5
4 6
```

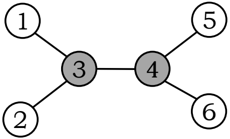
**【样例输出】**

```
2
```

**【样例解释】**

如果选择 $3, 5$，则无法服务 $6$；如果选择 $3, 4$，可以服务所有计算机，这也是最少的服务器。



# 【例】完美的服务

- 一台客户机只可以连接一台服务器，那么对于每一台计算机 $x$，有 3 种情况：

  1. 如果 $x$ 是服务器，那么其子结点可以是服务器，也可以不是服务器。
  2. 如果 $x$ 是客户机，$x$ 的父结点是服务器，那么其所有子结点都不是服务器。
  3. 如果 $x$ 是客户机，$x$ 的父结点也是客户机，那么 $x$ 的子结点恰好有一个是服务器。

- 定义 $f(x,0)$ 表示情况❶的最小服务器数；定义 $f(x,1)$ 表示情况❷的最小服务器数；定义 $f(x,2)$ 表示情况❸的最小服务器数。
- 对于情况❶，其子结点可以是也可以不是服务器，那么取最小值之加 1 即可：

$$f(x,0) = \sum_{y \in Son(x)} \min\{f(y,0), f(y,1)\} + 1$$

- 对于情况❷，其子结点都不是服务器，那么有：

$$f(x,1) = \sum_{y \in Son(x)} f(y,2)$$

# 【例】完美的服务

- 对于情况❸，其子结点中有一台服务器，其他都不是服务器，那么枚举其第 $i$ 个孩子 $y_i$ 为服务器，有：

$$f(x,2)=\min_{y_i\in Son(x)} f(y_1,2)+f(y_2,2)+\cdots+f(y_i,0)+\cdots+f(y_k,2), k=|Son(x)|$$

  计算该值的时间复杂度为 $\mathrm{O}(K^2)$。
- 实际上，$f(x,1)=\sum\limits_{y\in Son(x)} f(y,2)$ 表示所有子结点都不是服务器的情况，那么如果某一个子结点 $y$ 是服务器，那么只需要将 $f(y,2)$ 替换为 $f(y,0)$ 即可，也就是 $f(x,2)=\min\limits_{y\in Son(x)}\{f(x,1)-f(y,2)+f(y,0)\}$
- 目标状态：$\min\{f(1,0),f(1,2)\}$。
- 算法的时间复杂度为：$\mathrm{O}(N)$。

# 【例】完美的服务

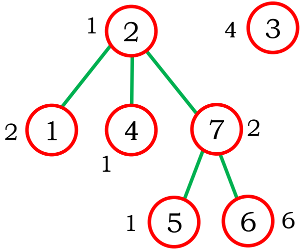
```
void dp(int x, int fa)
{
    f[x][0] = 1, f[x][1] = 0, f[x][2] = N;
    if(e[x].size() == 1 && fa) return; // 叶子
    for(int i = 0; i < e[x].size(); ++i)
    {
        int y = e[x][i];
        if(y == fa) continue;
        dp(y, x);
        f[x][0] += min(f[y][0], f[y][1]);
        f[x][1] += f[y][2];
    }
    for(int i = 0; i < e[x].size(); ++i)
    {
        int y = e[x][i];
        if(y == fa) continue;
        f[x][2] = min(f[x][2], f[x][1] - f[y][2] + f[y][0]);
    }
}
```

# 【例】选课

学校开设了 $n$ 门选修课程，每个学生最多能选择 $m$ 门课程。在选修课程中，有些课程可以直接选修，有些课程则需要修先修课程。已知，每门课的直接先修课程最多只有一门，但是两门课可能存在相同的先修课程。

你的任务是为自己确定一个选课方案，使得你能得到的学分最多。例如，在最多选择 4 门课程的情况下，选择课程 $2, 3, 6, 7$，能获得最多学分为 13。

| 课程 | 先修课程 | 学分 |
|---|---|---|
| 1 | 2 | 2 |
| 2 | 无 | 1 |
| 3 | 无 | 4 |
| 4 | 2 | 1 |
| 5 | 7 | 1 |
| 6 | 7 | 6 |
| 7 | 2 | 2 |



# 【例】选课

**【输入格式】**

第一行两个整数 $m, n(1 \le m \le n \le 500)$，分别表示课程总数和学生最多可以选的课程总数。

接下来 $m$ 行，第 $i$ 有两个数 $x, s$，表示课程 $i$ 的先修课程为 $x$(若不存在先修课程，则 $x = 0$)，学分为 $s$。

**【输出格式】**

第一行一个一个整数，表示所选课程的学分总数。

接下来一行 $n$ 个整数，表示学生选课的课程编号 (从小到大)。

**【样例输入】**

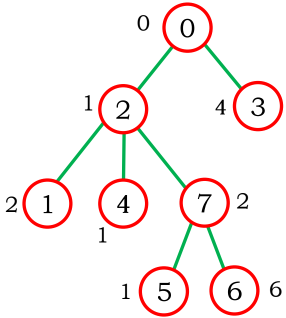
```
7 4
2 2
0 1
0 4
2 1
7 1
7 6
2 2
```

**【样例输出】**

```
13
2 3 6 7
```

# 【例】选课

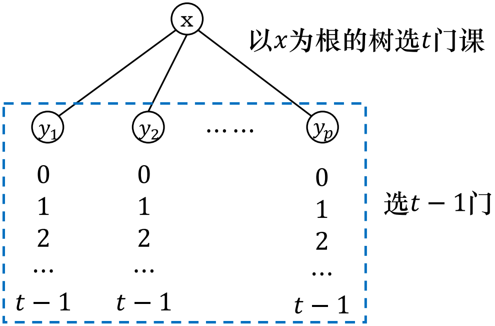
- 每门课的先修课最多只有一门，所以这 $m$ 门课程构成了森林结构。为了方便，我们建立一个虚拟根结点——0 号结点 (学分为 0)，作为“实际上没有先修课程的课程”的先修课，注意此时共需要选 $n=n+1$ 节课。
- 状态：定义 $f(x,t)$ 表示在以 $x$ 为根的子树中选 $t$ 门课程能够获得的最高学分。
- 初始状态：$f(x,0)=0(0\le x\le n)$，显然选择 0 门课程的最高学分为 0。
- 目标状态：$f(0,n)$。



# 【例】选课

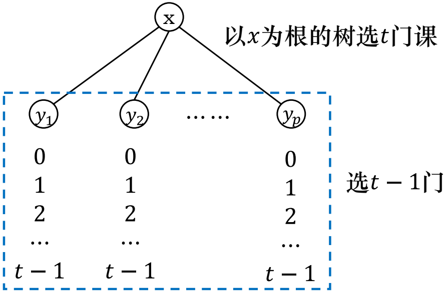
- 状态转移方程：修完 $x$ 这门课后，对于 $x$ 的每个子结点 $y_i \in Son(x)$，我们可以在以 $y_i$ 为根的子树中选修若干门课 (记为 $c_i$)，在满足 $\sum c_i = t-1$ 的基础上获得尽量多的学分，即

$$f(x,t) = \max_{\sum c_i = t-1} \left\{ \sum_{i=1}^{p} f(y_i, c_i) \right\} + s[x], p = |Son(x)|$$



# 【例】选课

- 该方程实际上是一个分组背包模型。共有 $p = |Son(x)|$ 组物品，每组物品有 $t-1$ 个，其中第 $i$ 组的第 $j$ 个物品的体积 (课程总数) 为 $j$，价值为 $f(y_i, j)$，背包的总体积为 $t-1$。




- 我们需要从每组物品中选择不超过 1 个物品 (每个子结点 $y_i$ 最多只能选一个状态转移到 $x$)，使得物品体积不超过 $t-1$ 的前提下 (在修完 $x$ 后，还能选修 $t-1$ 门课)，物品的价值总和最大 (获得最多学分)。

# 【例】选课

- 为了方便实现，定义状态 $f(x,i,t)$ 表示在以 $x$ 为根的前 $i$ 棵子树中选 $t$ 门课能够获得的最高学分。
- 每棵子树的最终状态为 $f(x, e[x].size(), 0 \sim n)$，即所有组 (子树) 都处理完后的状态。

```
void dp(int x)
{
    // 根结点必选，相当于第 0 组物品必选
    f[x][0][0] = 0;
    for(int i = 1; i <= n; ++i) f[x][0][i] = s[x];
    // 背包容量为 t−1，共有 tree[x].size() 组物品，每组物品有 t−1 个
    // 第 i 组的第 j 个物品的体积为 j 价值为 f(y_i, e[y_i].size(), j)
    for(int i = 1; i <= e[x].size(); ++i) // 枚举组
    {
        int y = e[x][i - 1], sz = e[y].size();
        dp(y);
        for(int t = 0; t <= n; ++t)         // 背包容量
            for(int j = 0; j <= t - 1; ++j) // 枚举组内物品 j=0 表示不选该组物品
                f[x][i][t] = max(f[x][i][t], f[x][i - 1][t - j] + f[y][sz][j]);
    }
}
```

# 【例】选课

- 对于每棵子树，第 $i$ 个阶段的状态只与第 $i-1$ 个阶段有关，可以降维处理。
- 此时，需要注意需要逆序枚举背包容量。

```
void dp(int x)
{
    // 根结点必选，相当于第 0 组物品必选
    f[x][0] = 0;
    for(int i = 1; i <= n; ++i) f[x][i] = s[x];
    // 背包容量为 t-1，共有 e[x].size() 组物品，每组物品有 t-1 个
    // 第 i 组的第 j 个物品的体积为 j 价值为 f(y_i,j)
    for(int i = 1; i <= e[x].size(); ++i)   // 枚举组
    {
        int y = e[x][i - 1];
        dp(y);
        for(int t = n; t >= 0; --t)          // 枚举背包容量
            for(int j = 0; j <= t - 1; ++j) // 枚举组内物品 j=0 表示不选该组物品
                f[x][t] = max(f[x][t], f[x][t - j] + f[y][j]);
    }
}
```

# 【例】选课

- 小优化：在以 $y$ 为根的子树中选择 $j$ 个结点，那么 $j$ 必然小于等于 $y$ 的子树大小。

```
void dp(int x)
{
    // 根结点必选，相当于第 0 组物品必选
    f[x][0] = 0;
    for(int i = 1; i <= n; ++i) f[x][i] = score[x];
    // 背包容量为 t-1，共有 e[x].size() 组物品，每组物品有 t-1 个
    // 第 i 组的第 j 个物品的体积为 j 价值为 f(y_i, j)
    for(int i = 1; i <= e[x].size(); ++i)   // 枚举组
    {
        int y = e[x][i - 1];
        dp(y);
        for(int t = n; t >= 0; --t)           // 枚举背包容量
            for(int j = 0; j <= t - 1 && j <= d[y]; ++j) // 枚举组内物品
                f[x][t] = max(f[x][t], f[x][t - j] + f[y][j]);
    }
}
```

# 【例】苹果树

有一棵虚拟的苹果树，树有 $n$ 个结点，每个结点上有若干个苹果。从结点 1 出发，可以吃掉到达的所有结点上的所有苹果。当从一个结点转移到另一个结点时，需要走 1 步。请你计算走 $k$ 步最多吃多少个苹果。

# 【例】苹果树

**【输入格式】**

输入包含多组测试用例。

对于每组测试用例，第一行包含两个整数 $n, k (1 \le n \le 100, 0 \le k \le 200)$。

接下来一行 $n$ 个整数，表示每个结点上的苹果数 $(\le 1000)$。

接下来 $n-1$ 行，每行两个整数 $x, y$，表示结点 $x$ 和 $y$ 是相邻的。

**【输出格式】**

对于组测试用例，输出一行一个整数，表示问题的答案。

**【样例输入】**

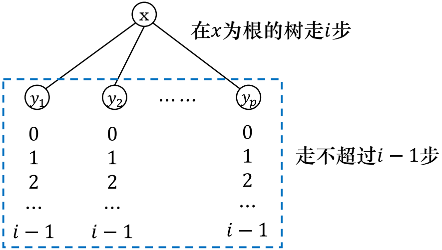
```
2 1
0 11
1 2
3 2
0 1 2
1 2
1 3
```

**【样例输出】**

```
11
2
```

# 【例】苹果树

- 定义 $f(x,i)$ 表示从 $x$ 出发在以 $x$ 为根的树中走 $i$ 步能吃到的最多的苹果。




- 显然也是一个类似与树形依赖的背包问题，每个子树也构成了分组背包。
- 但是从 $x$ 出发如果到 $y$，然后不回来，那么需要额外消耗 $x \to y$ 的 1 步；如果回到 $x$，那么需要额外消耗 $x \to y$ 和 $y \to x$ 的 2 步。

# 【例】苹果树

- 定义 $f(x,i,1), f(x,i,0)$ 分别表示从 $x$ 出发，在 $x$ 为根的子树中走 $i$ 步，回到/不回到 $x$，能吃到的最多的苹果数。
- 目标状态：$\max\{f(1,k,0), f(1,k,1)\}$。
- 如果从 $x$ 出发，到其中一棵子树吃苹果，然后回到 $x$，再到另一棵子树吃苹果，总步数为 $i$。这个过程分为：
  - 从 $x$ 到 $y$，遍历以 $y$ 为根的子树，从 $y$ 回到 $x$，去除从 $x\to y$ 和 $y\to x$ 的 2 步，再以 $v$ 为根的子树中走了 $j-2$ 步；
  - 用剩余的 $i-j$ 步，从 $x$ 出发，遍历其他的子树，再回到 $x$。

  状态转移方程为：

$$f(x,i,1) = \max\{f(y,j-2,1) + f(x,i-j,1)\}$$

# 【例】苹果树

- 如果从 $x$ 出发，不回到 $x$，那么会停在哪里呢？也需要分两种情况讨论。
- 从 $x$ 出发，最后停在以 $y$ 为根的子树中。那么分为两步：
  - 从结点 $x$ 遍历其余子树，回到 $x$；
  - 从结点 $x$ 遍历到子树 $y$，走 $j$ 步，最终停在 $y$ 的子树上[1]。

  状态转移方程为：

$$f(x,i,0)=\max\{f(x,i-j,1)+f(y,j-1,0)\}$$

- 从 $x$ 出发，最后停在其他子树中。那么分为两步：
  - 从结点 $x$ 遍历到子树 $y$，走 $j$ 步，回到 $x$；
  - 从结点 $x$ 遍历其余子树，走 $i-j$ 步，停在 $x$ 的其他子树上。

  状态转移方程为：

$$f(x,i,0)=\max\{f(y,j-2,1)+f(x,i-j,0)\}$$

[1]最终就算回到 $y$ 对答案也没有任何贡献。

# 【例】苹果树

```
void dp(int x, int fa)
{
    for(int i = 0; i <= k; ++i) f[x][i][0] = f[x][i][1] = a[x];
    for(int z = head[x]; z; z = nxt[z])
    {
        int y = ver[z];
        if(y == fa) continue;
        dp(y, x);
        for(int i = k; i >= 1; --i)
            for(int j = 1; j <= i; ++j)
            {
                // x-> 其他->x->y
                f[x][i][0] = max(f[x][i][0], f[x][i - j][1] + f[y][j - 1][0]);
                if(j >= 2)
                {
                    // x->y->x-> 其他
                    f[x][i][0] = max(f[x][i][0], f[y][j - 2][1] + f[x][i - j][0]);
                    // x->y->x-> 其他->x
                    f[x][i][1] = max(f[x][i][1], f[y][j - 2][1] + f[x][i - j][1]);
                }
            }
    }
}
```

# 【例】富集程度



有一个树形的水系，由 $n-1$ 条河道和 $n$ 个交叉点组成。我们可以把交叉点看作树中的结点，编号为 $1 \sim n$，河道则看作树中的无向边。

每条河道都有一个容量，连接 $x$ 与 $y$ 的河道的容量记为 $c(x,y)$。河道中单位时间流过的水量不能超过河道的容量。

有一个结点是整个水系的发源地，可以源源不断地流出水，我们称之为源点。除了源点之外，树中所有度数为 1 的结点都是入海口，可以吸收无限多的水，我们称之为汇点。也就是说，水系中的水从源点出发，沿着每条河道，最终流向各个汇点。

在整个水系稳定时，每条河道中的水都以单位时间固定的水量流向固定的方向。除源点和汇点之外，其余各点不贮存水，也就是流入该点的河道水量之和等于从该点流出的河道水量之和。整个水系的流量就定义为源点单位时间发出的水量。

在流量不超过河道容量的前提下，求哪个点作为源点时，整个水系的流量最大，输出这个最大值。

# 【例】富集程度

【输入格式】

第一行包含一个整数 $T$，表示有 $T$ 组数据。

对于每组数据，第一行一个整数 $n(n \leq 2 \times 10^5)$。

接下来 $n-1$ 行，每行三个整数 $x, y, z$，表示 $x, y$ 之间存在一条容量为 $z$ 的河道。

【输出格式】

对于每组数据，输入一个整数表示答案。

【样例输入】

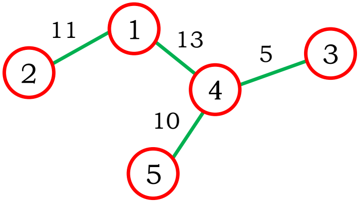
```
1
5
1 2 11
1 4 13
3 4 5
4 5 10
```

【样例输出】

```
26
```

# 【例】富集程度

- 选择 1 号点作为源点，流量为 $11+13=24$。
- 选择 2 号点作为源点，流量为 11。
- 选择 3 号点作为源点，流量为 5。
- 选择 4 号点作为源点，流量为 $11+5+10=26$。
- 选择 5 号点作为源点，流量为 10。



# 【例】富集程度

- 从 $1 \sim n$ 枚举源点 $s$，然后计算水系流量。把源点作为树根，整个水系就变成了一棵有根树，每条河道的方向也就确定了。
- 以树根为源点时，每个结点都只会从父结点获得水源，并流向它的子结点，每个结点的“流域”就是以该点为根的子树。
- 状态：定义 $f(x)$ 表示在 $x$ 为根的子树中，把 $x$ 作为源点，从 $x$ 出发流向子树的流量最大是多少。
- 初始状态：$f(i) = 0$，其中 $i$ 为汇点。
- 目标状态：$f(s)$，即根结点 $s$ 出发流向树的流量的最大值。
- 状态转移方程：每个结点 (度不为 1) 到子结点的流量由子结点为根的树的最大流量和边权的最小值决定

$$f(x) = \sum_{y \in Son(x)} \begin{cases} \min\{f(y), c(x,y)\} & y \text{ 的度} > 1 \\ c(x,y), & y \text{ 的度} = 1 \end{cases}$$

# 【例】富集程度

```
// 以 x 为根的子树的最大流量
void dp(int x, int fa)
{
    f[x] = 0;
    for(int i = head[x]; i; i = nxt[i])
    {
        int y = ver[i];
        if(y == fa) continue; // 不是父结点
        dp(y, x);
        if(deg[y] == 1) f[x] += w[i];
        else f[x] += min(f[y], w[i]);
    }
}

int ans = 0;
for(int i = 1; i <= n; ++i)
{
    dp(i, 0);
    ans = max(ans, f[i]);
}
```

# 【例】富集程度



- 对于枚举的每个源点 $s$，可以利用上面的算法在 $O(N)$ 的时间内求出以 $s$ 为根的树的流量，所以总的时间复杂度为 $O(N^2)$。
- 我们用“二次扫描与换根法”代替逐一枚举源点，可以在 $O(N)$ 的时间复杂度解决问题。
- 首先，任选一个顶点作为根结点 (例如 1 号结点)，利用上页的算法求出对应的 $f$ 数组。
- 设 $d(x)$ 表示把 $x$ 作为源点，流向整个水系，流量最大是多少，显然有 $d(1) = f(1)$。
- 假设 $d(x)$ 已被正确求出，考虑它的子结点 $y$，如何计算 $d(y)$?

# 【例】富集程度

- 假设 $d(x)$ 已被正确求出，考虑它的子结点 $y$，如何计算 $d(y)$？明显，$d(y)$ 包含两个部分：
  - 从 $y$ 流向以 $y$ 为根的子树的流量，这部分保存在 $f(y)$ 中。
  - 从 $y$ 沿着到父结点 $x$ 的河道，进而流向水系中其他部分的流量。

  问题是如何求结点 $y$ 沿着到父结点 $x$ 的的流量，如下图

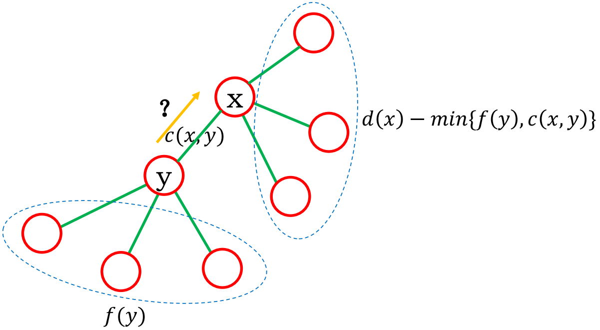



图: $? = \min\{d(x) - \min\{f(y), c(x,y)\}, c(x,y)\}$

# 【例】富集程度

- 注意：如果 $y$ 是叶子结点，从 $y$ 流向以 $y$ 为根的子树的流量 $f(y)=0$，那么从 $y$ 沿着到父结点 $x$ 的河道，进而流向水系中其他部分的流量就是 $\min\{d(x)-c(x,y),c(x,y)\}$，也就是以 $y$ 为源点的流量 $d(y)$。

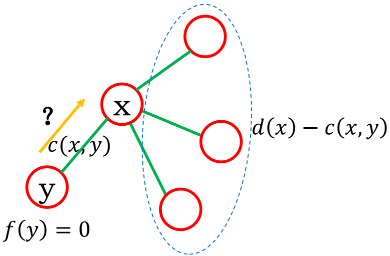



图: $d(y)=?=\min\{d(x)-c(x,y),c(x,y)\}$

# 【例】富集程度

- 假设 $d(x)$ 已被正确求出，考虑它的子结点 $y$，有

$$d(y)=\begin{cases} f(y)+\min\{d(x)-\min\{f(y),c(x,y)\},c(x,y)\} & y\text{ 的度数}>1 \\ \min\{d(x)-c(x,y),c(x,y)\} & y\text{ 的度数}=1 \end{cases}$$

- 与第一步自底向上递推不同，这一步是自顶向下递推。
- 第一步和第二步递推的时间复杂度均为 $O(N)$，故而总体的时间复杂度为 $O(N)$。

# 【例】富集程度

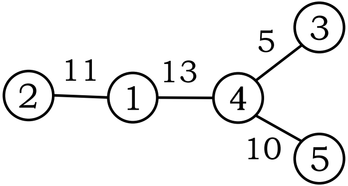


```cpp
void dfs(int x, int fa)
{
    for(int i = head[x]; i; i = nxt[i])
    {
        int y = ver[i];
        if(y == fa) continue;
        if(deg[y] == 1) d[y] = min(d[x] - w[i], w[i]);
        else d[y] = f[y] + min(d[x] - min(f[y], w[i]), w[i]);
        dfs(y, x);
    }
}

dp(1, 0);          // 求 f[i]
d[1] = f[1];
dfs(1, 0);         // 求 d[i]
int ans = 0;
for(int i = 1; i <= n; ++i) ans = max(ans, d[i]);
```

# 【例】计算机

学校不久前买了第 1 台计算机 (编号为 1)。后来学校又不断添置了 $n-1$ 台计算机，每台新计算机都被连接到先前安装的某一台计算机上。学校管理者担心网络运行缓慢，想知道第 $i$ 台计算机发送信号的最大距离 $s_i$(即到最远的计算机的电缆长度)。



# 【例】计算机

【输入格式】
输入包含多组测试用例。
对于每组测试用例，第一行包含一个整数 $n(1 \leq n \leq 10^4)$。
接下来 $n-1$ 行，第 $i$ 行包含两个整数 $x, w$，表示第 $i+1$ 台计算机连接的计算机编号和花费的电缆长度。
【输出格式】
对于组测试用例，输出一行一个 $n$ 整数，其中第 $i$ 个数表示第 $i$ 台电脑的 $s_i$。

【样例输入】

```
5
1 1
2 1
3 1
1 1
```

【样例输出】

```
3 2 3 4 4
```

# 【例】计算机

- 对于每一个结点，以其为根进行 BFS/DFS 求出最远距离即可。
- 时间复杂度：$O(N^2)$。

# 【例】计算机

- 考虑结点 $x$，它的最远距离是向下走的最远距离和向上走的最远距离取最大值即可。
- 从 $x$ 向下走的最远距离，进行一次 DFS 即可得到所有结点的值。
- 设结点 $y$ 的父结点为 $x$，则 $y$ 向上走的最远距离可以分为两种情况：
  1. $x$ 向下走的最远距离经过 $y$，那么 $y$ 向上走的最远距离为 $x$ 向上走的最远距离和 $x$ 向下走的次远距离取最大值 $+w(x,y)$；
  2. $x$ 向下走的最远距离不经过 $y$，那么 $y$ 向上走的最远距离为 $x$ 向上走的最远距离和 $x$ 向下走的最远距离取最大值 $+w(x,y)$。
- 定义 $f(x,0)$ 表示 $x$ 向下走的最远距离，$f(x,1)$ 表示 $x$ 向下走的次远距离，$f(x,2)$ 表示 $x$ 向上走的最远距离。
- 目标状态：对于每个结点 $x$，结果为 $\max\{f(x,0),f(x,2)\}$

# 【例】计算机



- 显然 $f(x,0)$ 和 $f(x,1)$ 显然进行一次 DFS 即可得出。

```
void dp(int x, int fa)
{
    for(int i = head[x]; i; i = nxt[i])
    {
        int y = ver[i];
        if(y == fa) continue;
        dp(y, x);
        int d = f[y][0] + w[i];
        // d 成了最大值 旧 f[x][0] 成了次大值
        if(f[x][0] <= d) f[x][1] = f[x][0], f[x][0] = d;
        else f[x][1] = max(f[x][1], d); // d 不是最大值，可能是次大值
    }
}
```

# 【例】计算机

- 求解 $f(x,2)$，需要考虑二次换根。
- 对于 $x$ 的孩子结点 $y$，如果 $x$ 向下走的最远距离经过$y$，那么有

$$f(y,2)=\max\{f(x,1),f(x,2)\}+w(x,y)$$

- 如果 $x$ 向下走的最远距离不经过$y$，那么有

$$f(y,2)=\max\{f(x,0),f(x,2)\}+w(x,y)$$

```cpp
void dfs(int x, int fa)
{
    for(int i = head[x]; i; i = nxt[i])
    {
        int y = ver[i];
        if(y == fa) continue;
        // y 在 f[x][0] 的路径上
        if(f[y][0] + w[i] == f[x][0]) f[y][2] = max(f[x][2], f[x][1]) + w[i];
        else f[y][2] = max(f[x][2], f[x][0]) + w[i]; // y 不在 f[x][0] 的路径上
        dfs(y, x);
    }
}
```

# 小结

- 树形动态规划是动态规划的一种常见模型，是在树形结构上进行递推的动态规划。
- 树形动态规划的重难点是从子节点到父节点的状态转移方程。
- 对于根不确定的树形动态规划问题可以使用二次扫描与换根法来解决。
- 对于树形动态规划，注意转化模型。

# 练习

- 没有上司的舞会 (COGS 3138)
- 完美的服务 (COGS 2448)
- 选课【CTSC 1997】(COGS 2466)
- 选课【CTSC 1997】(COGS 1199)
- 苹果树 (COGS 2454)
- 富集程度 (COGS 3542)
- 计算机 (COGS 2999)

# 练习

- 消防演练 (COGS 3651)
- 战略游戏 (COGS 2997)
- 贿赂 FIPA(COGS 2998)
- 路径覆盖 (COGS 4078)
- 赛道修建【NOIP 2018】(COGS 3055)
- 外卖 (COGS 4085)
- 喵星人集会 (COGS 4004)
- 苹果树【SDOI 2017】(COGS 3813)
- 建造军营【NOIP 2022】(COGS 3813)