动态规划—树形DP 学习笔记

引入

前置知识：树基础。

树形 DP，即在树上进行的 DP，最常见的状态表示为 $f_{u,\dots}$，表示以 u 为根的子树的某个东东。

本文将讲解一些经典题目（树的子树个数、树的最大独立集、树的最小点覆盖、树的最小支配集、树的直径、树的重心、树的中心），以及一些常见形式及思路（树上背包、换根 DP）。

分类

树形 DP 问题的划分方法有多种方式。

按照「求解目的」进行划分

- 选择节点类：在树上进行选择，相邻节点不允许同时选；
- 树形背包类：在树上进行背包式的选择；
- 树的直径类：各种树上最近、最远、最长一类。

按照「阶段转移的方向」进行划分

- 自底向上：通过递归的方式求解每棵子树，然后在回溯时，自底向上地从子节点向上进行状态转移。只有在当前节点的所有子树求解完毕之后，才可以求解当前节点，以及继续向上进行求解。
- 自顶向下：从根节点开始向下递归，逐层计算子节点的状态。这种方法常常使用记忆化搜索来避免重复计算，提高效率。

自顶向下的树形 DP 问题比较少见，大部分树形 DP 都是采用「自底向上」的方向进行推导。

按照「是否有固定根」进行划分

- 固定根的树形 DP：事先指定根节点的树形 DP 问题，通常只需要从给定的根节点开始，使用 1 次深度优先搜索。
- 不定根的树形 DP：事先没有指定根节点的树形 DP 问题，并且根节点的变化会对一些值，例如子节点深度和、点权和等产生影响。通常需要使用 2 次深度优先搜索，第 1 次预处理诸如深度，点权和之类的信息，第 1 次开始运行换根动态规划。

树的子树个数

题目：P2796 Facer的程序

题意：统计树的子树个数。

设 f_u 为以 u 为根的子树的子树数量，对于每个节点，可以选择任意儿子 v 的 f_v 种形态，也可以不选，因此：$f_u = \prod_{v\in \mathrm{son}_u}(f_v + 1)$；最后的，$\mathrm{ans} = \sum\limits_{i=1}^{n} f_i$，注意取模就可以了。

代码：

```
ll dp[N];
void dfs(int u, int fa) {
    dp[u] = 1;
    for (int v : g[u]) if (v != fa) {
        dfs(v, u); dp[u] = dp[u] * ((dp[v] + 1) % MOD) % MOD;
    }
} // 树的子树个数
```

树的最大独立集

例题：P1352 没有上司的舞会

问题描述：对于无根树，选出若干的点，相邻的节点不能同时选，最大化选的点的个数。

因为染色至于相邻节点有关，因此可以任选一个点为根，一般选 1 作为根即可。

分析：每个点只有两种选择，因此设 $f_{i,0/1}$ 表示第 i 是否选择，对应的最大个数。

有：

- $f_{u,0} = \sum_{v\in \mathrm{son}_u} \max\{f_{v,0}, f_{v,1}\}$，即这个点不选，它的子节点可选可不选；
- $f_{u,1} = (\sum_{v\in \mathrm{son}_u} f_{v,0}) + 1$，即这个点选，它的子节点一定不能选。

代码：

```
void dfs(int u, int fa) {
    for (int i = h[u] ; i != -1 ; i = ne[i]) {
        int v = e[i]; if (v == fa) continue;
        dfs(v, u);
        dp[u][0] += max(dp[v][0], dp[v][1]), dp[u][1] += dp[v][0];
    } ++dp[u][1];
} // 树的最大独立集
```

拓展：对于基环树呢（题目：P2607 骑士）？非常简单，将这个环上的任意一个边 (s, t) 断开，然后最终结果取 $\text{ans} = \max\{f_{s,0}, f_{t,0}\}$。

树的最小点覆盖

例题：P2016 战略游戏、UVA1292 Strategic game

题目描述：在树的节点上放置士兵，每个士兵可以看守与它相邻的边，即，在 u 点上的士兵可以看守边 $(u, v) \mid v \in \text{neighbor}_u$，最小化放置的士兵数量。

和上一题相似，设 $f_{i,0/1}$ 表示看守以 i 为根的子树上的所有边，第 i 号点是否放置士兵，对应的最小士兵数量。

有：

- $f_{u,0} = \sum_{v \in \text{son}_u} f_{v,1}$，即 u 不放士兵，为看守 (u, v)，它的儿子必须放士兵；
- $f_{u,1} = (\sum_{v \in \text{son}_u} \min\{f_{k,0}, f_{k,1}\}) + 1$，即 u 放置士兵，它的儿子可以放也可以不放。

代码：

```
int f[N][2];
void dfs(int u, int fa) {
    f[u][0] = 0, f[u][1] = 1;
    for (int v : g[u]) if (v != fa) {
        dfs(v, u); f[u][0] += f[v][1];
        f[u][1] += min(f[v][0], f[v][1]);
    }
} // 树的最小点覆盖
```

树的最小支配集

题目：P2899 Cell Phone Network G

强烈推荐这篇题解 $>_{\sim}<$：https://www.luogu.com.cn/blog/HSH/post-shu-xing-dp-shou-ji-wang-lao（大概除了奇怪的公式~~罢子~~），这里大概的转述一下。

设 $f_{u,0/1/2}$ 表示考虑 u 及其子树，是［自己努力-0；坑爹-1；找儿子帮忙-2］。

考虑转移边 (u,v)，有：

- $f_{u,0}=(\sum_{v\in \mathrm{son}_u}\min\{f_{v,0},f_{v,1},f_{v,2}\})+1$：因为 u 点有一个通讯塔，所以 v 有可能从它的父亲（也就是 u 点）转移过来，也有可能有它的儿子转移过来，也有可能它自己本身就有一个通讯塔；
- $f_{u,1}=\sum_{v\in \mathrm{son}_u}\min\{f_{v,0},f_{v,2}\}$：因为 u 点没有通讯塔，所以 v 也不可能从它的父亲（也就是 u 点）转移过来，它只有可能由它的儿子转移过来，或者是它自己有一个通讯塔；
- $f_{u,2}=\sum_{v\in \mathrm{son}_u}\min\{f_{v,0},f_{v,2}\}$：因为 u 点没有通讯塔，所以 v 也不可能从它的父亲（也就是 u 点）转移过来，它只有可能由它的儿子转移过来，或者是它自己有一个通讯塔。

但这样写我们很容易发现一个问题（事实上观察公式，会发现 $f_{u,1}\overset{?}{=}f_{u,2}$ 然而这是 obviously impossible 的），如果 $f_{u,2}$ 全是从 $f_{v,2}$ 转移过来的话，就证明它的所有儿子都没有通讯塔，那么就不可能将它覆盖，如何避免这种情况呢？

其实我们只需要记录下每一次 $f_{v,0}-f_{v,2}$ 的差值，再取一个 $\min$ 值，简单的来说，就是记 $p=\min\{f_{v,0}-f_{v,2}\}$；在最后，如果我们发现 $f_{u,2}$ 全是从 $f_{v,2}$ 转移过来的话，就再加上一个 p，就相当于将其中的一个 $f_{v,2}$ 强制转换为了 $f_{v,0}$，同时还保证了最小。

最后的，结果是 $\min\{f_{\mathrm{root},0},f_{\mathrm{root},2}\}$，因为最后的根结点是不可能坑爹的（它没有父亲）。

对于有点权的（U364106 皇宫看守），求解方式不变，只需要将 $f_{u,0}$ 的转移方程的 $+1$ 改为 $+$点权，即下面代码的第 4 行的 `f[u][0] = 1` 改为 `f[u][0] = |点权|`。

代码：

```
1 int f[N][3];
2 void dfs(int u, int fa) {
3     int p = 2e9; bool flag = 1;
4
```

```cpp
    f[u][0] = 1, f[u][1] = f[u][2] = 0;
    for (int v : g[u]) if (v != fa) {
        dfs(v, u);
        f[u][0] += min(f[v][0], f[v][1], f[v][2]);
        f[u][1] += min(f[v][0], f[v][2]);
        if (f[v][0] <= f[v][2]) { f[u][2] += f[v][0], flag = 0; }
        else { f[u][2] += f[v][2], p = min(p, f[v][0] - f[v][2]); }
    } if (flag) f[u][2] += p;
} // 树的最小支配集
```

树的直径

定义

树上任意两节点之间最长的简单路径即为树的直径。

显然，一棵树可以有多条直径，他们的长度相等。

性质

1. 若树上所有边边权均为正，则树的所有直径有交，且中点重合；
2. 有树的直径 (p,q)，则距离任意点 x 最远的点一定为 p 或 q；
3. 树的直径的中点到其他所有点的最大距离最小（详见下面，树的中心）；
4. 两个树的一条直径分别为 (s_1,t_1) 和 (s_2,t_2)，把这两个树通过一条边合并成一棵大树，大树直径的两个端点必在 s_1,t_1,s_2,t_2 中取，共有 $\binom{4}{2}=6$ 种情况；
5. 两个树的直径分别为 ℓ_1,ℓ_2，把这两个树直径的中点相连，新生成的树直径最小，且新直径长度为 $\max\{\ell_1,\ell_2,\lceil\ell_1/2\rceil+\lceil\ell_2/2\rceil+1\}$。

求解：两遍 DFS

仅适用于，边权非负；否则贪心不成立。

1. 从任意点 x 出发，找到树上距离点 x 最远的点 p；
2. 从点 p 出发，找到树上距离点 p 最远的点 q；
3. 则 (p,q) 为该树的直径。

证明：见 OI-Wiki。

拓展：求方案，记录每个点的父亲是谁，然后从 q 一步步推到 p 就行了。

代码：

```cpp
vector<int> g[N]; int c, d[N];
void dfs(int u, int fa) {
    for (int v : g[u])
        if (v != fa) { if ((d[v] = d[u] + 1) > d[c]) c = v; dfs(v, u); }
} inline int solve() {
    d[1] = 0, c = 1; dfs(1, -1);
    d[c] = 0; dfs(c, -1);
    return d[c];
} // 两遍 DFS
```

求解：树形 DP

定理：树上每条链 (s,t) 都可以拆成两条直链 $(s,\text{lca}(s,t))+(t,\text{lca}(s,t))$；感性理解。

那么，对于每个点 x，就可以求出以这个点为 lca 的直径，即这个点下面的最长链和次长链 (m_1,m_2)，即可合并为以点 x 为 lca 的直径；最终取最大值即可。

拓展：求方案，记录每个点是由哪个点转移来，以及找到直径时的 lca 点 x。

代码：

```cpp
vector<int> g[N]; int res;
int dfs(int u, int fa) { // 返回以 u 为 lca 的树的直径
    int m1 = 0, m2 = 0; for (int v : g[u]) {
        if (v == fa) continue;
        int d = dfs(v, u) + 1;
        if (d >= m1) m2 = m1, m1 = d;
        else if (d >= m2) m2 = d;
    } res = max(res, m1 + m2); return m1;
} int solve() {
    res = 0; dfs(1, 0); return res;
} // 树形 DP
```

题目

实测第一种方法略慢，因为要跑两遍 DFS；但是第一种方法更方便记录方案。

模板题：SP1437、U283565；应用题：CF455C (Civilization)、U364101 旅游规划。

树的重心

定义

对无根树，每个点 x 的子树定义为：删去 x 后所形成的各个连通块；最大子树最小的点称为树的重心。

性质

1. 树的重心最多只有两个，且如果有两个，则它们相邻；
2. 重心的所有子树大小都不超过整棵树大小的一半；
3. 重心到树上所有点的距离和最小，如果有两个重心，则到它们的距离和相等；
4. 插入或删除一个叶子，树的重心的位置最多移动一个点；
5. 用一条边连接两棵树，新的数的重心在连接原来两棵树的重心的路径上；
6. 一棵树的重心一定在根节点所在的重链上。

求解：树形 DP

树形 DP，然后卡定义，枚举找最大子树最小的点。

对于有点权的，求解方式不变，只需要将下面代码的第 3 的 `sz[u] = 1` 改为 `sz[u] = |点权|` 即可。

代码：

```
1 vector<int> g[N]; int n, sz[N], mc[N], mx;
2 void dfs(int u, int fa) {
3     sz[u] = 1; for (int v : g[u]) if (v != fa) {
4         dfs(v, u), sz[u] += sz[v], mc[u] = max(mc[u], sz[v]);
5     } mc[u] = max(mc[u], n - sz[u]), mx = min(mx, mc[u]);
6 } void solve() {
7     mx = 2e9; dfs(1, -1);
8     for (int i = 1; i <= n; ++i) if (mc[i] == mx) printf("%d", i);
9 } // 输出所有重心
```

```
1 vector<int> g[N]; int n, sz[N], mc[N], r;
2 void dfs(int u, int fa) {
3     sz[u] = 1; for (int v : g[u]) if (v != fa) {
4         dfs(v, u), sz[u] += sz[v], mc[u] = max(mc[u], sz[v]);
5     } mc[u] = max(mc[u], n - sz[u]);
6     if (mc[u] < mc[r] || (mc[u] == mc[r] && u < r)) r = u;
7 } void solve() {
8     r = 0, mc[0] = 2e9; dfs(1, -1); printf("%d\n%d\n", r, mc[r]);
```

```
} // 输出重心及其最大子树大小
```

题目

模板题：U104609、U164672；应用题：P1670 (Tree Cutting)、P2986（Great Cow Gathering G，有点权）。

树的中心

定义

所有节点中，到树中其他节点的最远距离最小的节点，叫树的中心。

性质

性质即定义，到树中其他节点的最远距离最小。

求解：树的直径

分析定义，你会发现与上面树的直径的性质 (3) 一模一样。

证明：一棵树上到点 x 最远的点一定是其直径 (S, T) 的一个端点，根据三角不等式 $\operatorname{dis}(S,x)+\operatorname{dis}(x,T)\geq \operatorname{dis}(S,T)$，所以 $\max\{\operatorname{dis}(S,x),\operatorname{dis}(x,T)\}\geq \frac{1}{2}\operatorname{dis}(S,T)$，而等号是在 x 为 (S,T) 中点时取到。

然后就可以用树的直径求解了。注意这里要记录路径信息（中点难道不是路径上的吗），因此可以用 Solution 1 更加方便，虽然速度不是最优的。

代码：

```
vector<int> g[N];
int c, d[N], f[N];
void dfs(int u, int fa) {
    for (int v : g[u])
        if (v != fa) { if ((d[v] = d[u] + 1) > d[c]) c = v; f[v] = u;
dfs(v, u); }
} void solve() {
    d[1] = 0, c = 1; dfs(1, -1);
    d[c] = 0, f[c] = -1; dfs(c, -1);
    int len = d[c] + 1, lc = len >> 1;
    int q = c; for (int i = 1; i < lc; ++i) q = f[q];
    if (len & 1) printf("%d", f[q]);
```

```
    else printf("%d %d", min(q, f[q]), max(q, f[q]));
} // 树的直径
```

求解：树形 DP

又要 down 又要 up 的，两遍 DFS 居然还不一样？太麻烦不想写。

可以借鉴：https://www.cnblogs.com/Liuz8848/p/11726834.html

代码：

```
int dfs_down(int u, int fa) {
    down1[u] = down2[u] = -INF;
    for (int i = h[u]; i != -1; i = ne[i]) {
        int v = e[i]; if (v == fa) continue;
        int d = dfs_down(v, u) + 1;
        if (d > down1[u]) down2[u] = down1[u], down1[u] = d, p[u] = v;
        else if (d > down2[u]) down2[u] = d;
    } if (down1[u] == -INF && down2[u] == -INF) down1[u] = down2[u] = 0;
    return down1[u];
} void dfs_up(int u, int fa) {
    for (int i = h[u]; i != -1; i = ne[i]) {
        int v = e[i]; if (v == fa) continue;
        if (p[u] == v) up[v] = max(up[u], down2[u]) + 1;
        else up[v] = max(up[u], down1[u]) + 1;
        dfs_up(v, u);
    }
} void solve() {
    dfs_down(1, 0), dfs_up(1, 0);
    int res = INF; vector<int> ans;
    for (int i = 1; i <= n; ++i) {
        int t = max(down1[i], up[i]);
        if (t == res) ans.push_back(i);
        else if (t < res) res = t, ans.clear(), ans.push_back(i);
    } for (int i : ans) printf("%d ", i);
} // 树形 DP
```

经典问题 1： 最小化最大距离

选出 k 个连续的

问题简述

题目：P5536 核心城市

从一个 n 个点的树中选出相邻的 k 个点（ $k \le n$ ），使未被选出的点，到这个被选出来的块，最大距离最小。

求解：树的中心

考虑 $k=1$ 的情况，显然这是树的中心的模板题。

拓展到 $k>1$ 的情况，显然树的中心依旧要取，因为如果不取树的中心，树的直径的端点 (S,T) 到任意一个点的距离都 $> \frac{1}{2}\text{dis}(S,T)$。

因此，我们先选择直径中点，然后提根。

下一个选谁？选根的儿子中，子树最深的那个；因为如果它没有被选，那么它的儿子们也不能被选，那么到已选城市的距离最大值始终不会减小。然后，以此类推，每次都选一个子树最深的。

最大距离是多少？显然是未被选出的点中，最大的子树深度 -1。

总结：提树的中心为根，选择子树深度最大的 k 个，然后最小的最大距离为，子树深度第 $k+1$ 大的子树深度 -1。

代码：

```cpp
int c, a[N];
int d[N], f[N];
void dfs(int u, int fa) {
    for (int v : g[u]) if (v != fa) {
        d[v] = d[u] + 1, f[v] = u; dfs(v, u);
        if (d[v] > d[c]) c = v;
    }
} void dfk(int u, int fa) {
    for (int v : g[u]) if (v != fa)
        dfk(v, u), a[u] = max(a[u], a[v] + 1);
} int solve() {
    c = 1, d[1] = 1; dfs(1, -1);
    d[c] = 1; dfs(c, -1);
    int lc = d[c] >> 1; for (int i = 1; i <= lc; ++i) c = f[c];
    dfk(c, -1); sort(a + 1, a + 1 + n, greater<int>());
    return a[k + 1] + 1;
} // P5536 核心城市
```

选出任意 2 个

问题简述

题目：ZOJ3820（卡空间）、GYM100554B（空间足够）、U370080 Building Fire Stations（有翻译＋我自己出的数据，不卡空间）

一棵树，在其中找两个点，使得其他点到这两个的距离的较小值的最大值的最小值及其方案。

可以看我的题解：https://www.cnblogs.com/RainPPR/articles/solution-zoj3820.html

求解：树的中心

选一个点的情况？树的中心模板！

两个点呢？

假设我们已经选出了两个点 (a,b) 那么，显然，树中所有的点要么去 a 要么去 b，且一定有一条边作为分界线；即有一条边 (p,q) 其两边子树中的点，都去 a 或都去 b；且点 a 和点 b 一定不在同一棵子树中。

因此，我们就可以选点 a 为 p 一侧的子树的中心，点 b 为 q 一侧的子树的中心。然后考虑 (p,q) 怎么选？（感性理解）可以大胆猜测，边 (p,q) 一定是树的直径中，中间的那一条边！

为什么呢？与树的中心的证明类似：一棵树上到点 x 最远的点一定是其直径 (S,T) 的一个端点，根据三角不等式 $\operatorname{dis}(S,x)+\operatorname{dis}(x,T)\ge\operatorname{dis}(S,T)$，所以 $\max\{\operatorname{dis}(S,x),\operatorname{dis}(x,T)\}\ge\frac{1}{2}\operatorname{dis}(S,T)$，而等号是在 x 为 (S,T) 中点时取到。

我们可以得出：$\max\{\operatorname{dis}(S,p),\operatorname{dis}(q,T)\}\le\frac{1}{2}\operatorname{dis}(S,T)\quad(\operatorname{map}_{p,q}=1)$，因此有 (p,q) 是 (S,T) 上最中间的一段。

证明结束。

注意，原数据会爆栈，所以不能用 DFS，可以换成 BFS（但是 Codeforces GYM 上空间加了一倍，到了 256 MB，这就卡卡能过了）。

代码：

```cpp
// ......
int n; vector<int> g[N];
int lc, rc, d[N], f[N];
#define get find_bfs::find  // 在这里切换 DFS 和 BFS
namespace find_bfs {        // 返回距离 u 最远的点
    pii q[N]; int st, ed;
    int find(int u) {
        d[u] = 0, f[u] = -1; st = 0, ed = 0, q[ed++] = {u, -1};
        int c = u; while (st < ed) {
            pii now = q[st++]; int u = now.first;
            for (int v : g[u]) if (v != now.second) {
                if ((u == lc && v == rc) || (u == rc && v == lc))
continue;
                q[ed++] = {v, u}; f[v] = u;
                if ((d[v] = d[u] + 1) > d[c]) c = v;
        }} return c;
    }
}; namespace find_dfs {     // 返回距离 u 最远的点
    int c; void dfs(int u, int fa) {
        for (int v : g[u]) if (v != fa) {
            if ((u == lc && v == rc) || (u == rc && v == lc)) continue;
            if ((d[v] = d[u] + 1) > d[c]) c = v; dfs(v, u); f[v] = u;
    }} int find(int u) {
        d[u] = 0, f[u] = -1, c = u;
        dfs(u, -1); return c;
    }
} int main() {
    int T = rr; while (T--) {
        n = rr; for (int i = 1; i < n; ++i) add(rr, rr);
        lc = rc = -1; int s = get(1), e = get(s);
        lc = e, rc = f[e]; for (int i = 1; i < d[e]; i += 2) tie(lc, rc) = make_tuple(rc, f[rc]);
        int le = get(get(lc)), ls = d[le] + 1; for (int i = d[le]; i > 1; i -= 2) le = f[le];
        int re = get(get(rc)), rs = d[re] + 1; for (int i = d[re]; i > 1; i -= 2) re = f[re];
        printf("%d %d %d\n", max(ls, rs) >> 1, le, re);
        for (int i = 1; i <= n + 1; ++i) g[i].clear();
    }
    return 0;
} // Building Fire Stations
```

经典问题2：求树的直径上的点

题目描述

题目：U364101 旅游规划、LOJ10159 旅游规划

题面简述：给定一棵树，树的最长路径可能不唯一，求出所有在树的最长路径（树的直径）上的点。

分析

考虑最原始的树的直径的求法：找到两条最长链；因此，当我们知道树的最长路径的时候，也可以以类似的方法求出一个点是否在树的直径上：

- 考虑 i 是否在树的直径上，也就是用所有可能求出来的信息拼凑出一个包含 i 的长度与树的直径相等的链；
- 因为 i 一定要向下走一条路，所以我们取 i 下面的最长链，设为 s_i；
- 然后再选出一条链来，这里不要想当然的认为是 i 下面的次长链，因为也有可能 i 不是这条链的转折点；
- 所以我们除了 i 下面的次长链，设为 t_i，还要考虑从 i 往上走的最长链，记为 c_i。

整理一下：设 s_i 为 i 下面的最长链，t_i 为 i 下面的（非严格）次长链，c_i 为 i 上面（可以拐弯）的最长链。

然后我们会发现 s_i 与 t_i 可以在一遍 DFS 中求出来，其实也就是第 2 中求树的直径的方法，然后 c_i 怎么求？分类讨论，假设我们现在要从节点 u 转移到其儿子节点 v：

- 节点 v 向上，可以经过父亲节点 u 然后继续往上，即 $c_v = c_u + 1$；
- 节点 v 还可能经过父亲节点后，向下，从父亲的另外一个儿子节点下去，现在考虑这个怎么算：
  - 如果节点 v 在父亲节点的最长链上，那么 v 就不可能从它自己在下去了，即 $c_v = t_u + 1$；
  - 如果节点 v 不在父亲节点的最长链上，则从父亲的最长链下去，一定是最优的，即 $c_v = s_u + 1$；

整理一下：$\left\{\begin{array}{l|l} c_v = \max\{c_u, s_u\} + 1 & (\text{if } s_v \Rightarrow s_u) \\ c_v = \max\{c_u, t_u\} + 1 & (\text{else}) \end{array}\right.$

代码：

```
int m1[N], m2[N], c1[N], res;
void dfs1(int u, int fa) {
    for (int v : g[u]) if (v != fa) {
        dfs1(v, u); int d = m1[v] + 1;
        if (d > m1[u]) m2[u] = m1[u], m1[u] = d, c1[u] = v;
        else if (d > m2[u]) m2[u] = d;
    } res = max(res, m1[u] + m2[u]);
} int up[N];
void dfs2(int u, int fa) {
    for (int v : g[u]) if (v != fa) {
        if (v == c1[u]) up[v] = max(up[u], m2[u]) + 1;
        else up[v] = max(up[u], m1[u]) + 1;
        dfs2(v, u);
    }
} // 树的直径上的点
```

树上背包

回忆背包问题

设 $f_{i,j}$ 为考虑前 i 个物品，总花费为 j 时的最大收益。

对于 $0/1$ 背包，有 $f_{i,j} = \max\{f_{i-1,j}, f_{i-1,j-v_i} + w_i\}$，即分类讨论第 i 个物品选不选，然后可以滚动数组优化为一维，也就是倒序枚举 j 而省去第一维空间。

类比树上背包

树上背包其实是一种分配子树的思想，状态设计类似于常规树上问题与背包问题，一般设计为：

设 $f_{u,i,j}$ 为，看以 u 为根的子树，考虑前 i 个子树，总花费为 j 时的最大收益，同样可以使用滚动数组优化掉第二维的 i。

常见的转移方程形式为：$f_{u,i,j} = \max\limits_{v \in \text{son}_u} \max\limits_{k \le j, k \le s_v} (f_{u,i-1,i-k} + f_{v,s_v,k})$；其实转移的时候更常见的转移形式为：$f_{u,i+j} = \sum_{v \in \text{son}_u} (f_{u,i} + f_{v,j})$，这样需要考虑的边界情况更少一些。

例题 1： 二叉苹果树

题目：P2015 二叉苹果树

设 $f_{u,i}$ 表示以 u 为根的子树，保留 i 个树枝，最多能留住的苹果数量，

有：$f_{u,i}=\max\limits_{(u,v,w)\in\text{son}_u}\max\limits_{j<i,j\le s_v}(f_{u,i-j-1}+f_{v,j}+w)$.

即以节点 v 为根的子树保留 j 个树枝的苹果数量 $f_{v,j}$，加上本身结点 u 剩下 $i-j-1$ 个树枝保留的苹果数量（多减的一个 1 是因为要保留边 (u,v)），即从 $f_{u,i-j-1}$ 转移过来。

注意：i,j 需要倒序枚举，因我们此处进行的是 $0/1$ 背包。

代码：

```
void dfs(int u, int fa) {
    for (pair<int, int> i : g[u]) if(i.first != fa) {
        int &v = i.first, &w = i.second; dfs(v, u);
        for(int j = q; j >= 0; --j) for(int k = 0; k < j; ++k)
            f[u][j] = max(f[u][j], f[u][j - k - 1] + f[v][k] + w);
    }
} // 二叉苹果树
```

例题 2：选课

题目：P2014 选课

每个点最多只有一个父亲，因此这是一个森林。

把 0 号也看做一个节点，则森林化为树，因此可以强制选 0 号点，然后考虑选出 $m+1$ 个节点。

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 x 的孩子个数为 c_x，以 x 为根的子树大小为 s_x，可以写出下面的状态转移方程：

常见的转移方程形式为：$f_{u,i,j}=\max\limits_{v\in\text{son}_u}\max\limits_{k\le j,k\le s_v}(f_{u,i-1,j-k}+f_{v,c_v,k})$.

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；而且 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。

代码：

```
int f[N][M];
int dfs(int u) {
    int sz = 1; f[u][1] = s[u];
    for (int v : g[u]) {
        int up = dfs(v);
        for (int i = min(sz, m + 1); i; --i)
            for (int j = 1; j <= up && i + j <= m + 1; ++j)
                f[u][i + j] = max(f[u][i + j], f[u][i] + f[v][j]);
        sz += up;
    } return sz;
} // 选课
```

换根 DP

定义

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。

例题：STA-Station

题目：P3478 STA-Station

问题描述：给出一个 n 个节点的无根树，选一个节点作为根，使树上所有节点的深度和最大。

设 f_i 为以 i 为根的树的深度和。

我们先随便选一个点作为根，可以选 1 号节点为根，然后进行一次 DFS 就可以求以 f_1 以及每个子树的大小。

假设我们已经求出了 f_u（如，此时 $u = 1$），然后考虑状态转移，这里就是体现 " 换根 " 的地方了，也就是 $f_v \leftarrow f_u$ 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：

（非常）易知 $f_v = f_u - down_v + up_v$，其中 $down_v$ 表示以 v 为根的子树大小，up_v 表示除 v 及其子树外的树的大小，而 $up_v = n - down_v$。

也就是下面的点上来了，深度 -1；上面的点下去了，深度 $+1$。

代码：

```
int dfs_f(int u, int fa, int deep) {
    ra += deep;
    for (int i = h[u]; i != -1; i = ne[i]) {
        int v = e[i]; if (v == fa) continue;
        s[u] += dfs_f(v, u, deep + 1);
    } return s[u] + 1;
} int res, ans;
void dfs_s(int u, int fa, int rt)
{
    int now = rt - s[u] + (n - s[u] - 1);
    if (now > res) res = now, ans = u;
    for (int i = h[u]; i != -1; i = ne[i]) {
        int v = e[i]; if (v == fa) continue;
        dfs_s(v, u, now);
    }
} // 树的深度和
```

练习题

见：https://www.luogu.com.cn/training/364552

Reference

[1] https://oi-wiki.org/graph/tree-diameter/

[2] https://oi-wiki.org/graph/tree-centroid/

[3] https://oi-wiki.org/dp/tree/

[4] https://www.cnblogs.com/RioTian/p/15110212.html

[5] https://www.cnblogs.com/RioTian/p/15163878.html

[6] https://www.luogu.com.cn/blog/BreakPlus/dp-on-tree

[7] https://www.luogu.com.cn/blog/HSH/post-shu-xing-dp-shou-ji-wang-lao

[8] https://blog.csdn.net/lyd_7_29/article/details/79854245

[9] https://www.luogu.com.cn/blog/103452/solution-p3629

[10] http://www.manongjc.com/detail/29-huvpgpevtxovflb.html

[11] https://algo.itcharge.cn/10.Dynamic-Programming/06.Tree-DP/01.Tree-DP/

[12] https://www.luogu.com.cn/blog/Fireworks-Rise/post-dong-tai-gui-hua-shu-xing-dptree-dp

---

合集：学习笔记

标签：算法 ，学习笔记