① 树状DP　从根节点向下DFS，记录合并子树的信息。

常用：状态定义　dp[u][0]选u的最优值

dp[u][1]不选u的最优值

转移方程

用dfs

for v in children[u]

dfs(v)

dp[u]=合并(dp[u],dp[v])

☆经典应用：

1. 树的最长路径

☆找到所有子树最大两个深度，DP

```
def dfs(node):
    x_len = 0
    for y in x.children
        y_len = dfs(y)
        ans = max(ans, x_len+y_len)
        x_len = max(x_len, y_len)
    return x_len
```

① 树的独立最大集　不能选择相邻的两个节点，每个节点有权值，求最大权值和

```python
    dp = [[0, 0] for _ in range(n + 1)]  # dp[u][0] 不选，dp[u][1] 选
    visited = [False] * (n + 1)

    def dfs(u):
        visited[u] = True
        dp[u][1] = weights[u - 1]  # 节点编号从1开始
        for v in tree[u]:
            if not visited[v]:
                dfs(v)
                dp[u][0] += max(dp[v][0], dp[v][1])
                dp[u][1] += dp[v][0]

    dfs(1)
    return max(dp[1][0], dp[1][1])
```

显式dp

```python
from functools import lru_cache
n=int(input())
value=[0]+list(map(int,input().split()))
@lru_cache(None)  3 usages
def dfs(i):
    if i>n:
        return (0,0)
    left=i*2
    right=i*2+1
    l0,l1=dfs(left)
    r0,r1=dfs(right)
    not_take=max(l0,l1)+max(r0,r1)
    take=value[i]+l0+r0
    return (not_take,take)
print(max(dfs(1)))
```

返回值dp

② 树上背包问题　在树上取点，需获得权值最大的取法，满足子树大小限制（对每个u，以u为根节点的子树不能选取超k_u个）

★ 部分课有先修课程要求，求总课程数限制为m节时，最多可以学多少学分

```python
# 构建图
tree = defaultdict(list)
score = [0] * (n + 1)

for i in range(1, n + 1):
    fa, s = prerequisites[i - 1]
    score[i] = s
    tree[fa].append(i)  # 建树，从先修课程指向当前课程

# 初始化 DP 表：f[u][j] 表示以 u 为根的子树中选 j 门课程的最大得分
f = [[0] * (m + 2) for _ in range(n + 1)]  # 多开一位方便处理 f[0][m+1]

def dfs(u):
    # 选了自己这门课
    f[u][1] = score[u]

    for v in tree[u]:
        dfs(v)
        # 树形背包：合并子树 v 到当前子树 u
        for j in range(m + 1, 0, -1):          # 当前树选了 j 门
            for k in range(0, j):              # 子树可以贡献 0~j-1 门（包括不选）
                if f[v][k]:                    # 有方案才合并
                    f[u][j] = max(f[u][j], f[u][j - k] + f[v][k])

dfs(0)  # 虚拟根节点编号为 0
print(f[0][m + 1])  # 从虚拟根选 m + 1 门课（含根）
```

★ 虚拟节点0（树用散列表表示）

（n+1个节点选m+1个）

k<j，必须选u

f[u][j] 表示以节点u为根的子树，选j门课的最大得分

⑤寻找树的重心 （重心：删除某个节点后，剩余连通分量中最大节点数最小的节点）

```python
def dfs(u, fa, g, n, sum_sz, max_part):
    global min_max_part, ans_node
    sum_sz[u] = 1  # 初始化子树大小
    max_child = 0  # 子树中最大节点数
    for v in g[u]:
        if v != fa:
            dfs(v, u, g, n, sum_sz, max_part)
            sum_sz[u] += sum_sz[v]  # 累加子树大小
            max_child = max(max_child, sum_sz[v])  # 更新子树最大节点数
    # 删除 u 后，最大连通分量为 max(子树最大节点数, 剩余节点数)
    max_part[u] = max(max_child, n - sum_sz[u])
    # 更新答案
    if max_part[u] < min_max_part or (max_part[u] == min_max_part and u < ans_node):
        min_max_part = max_part[u]
        ans_node = u

# 输入处理
n = int(input())
g = defaultdict(list)
for _ in range(n - 1):
    u, v = map(int, input().split())
    g[u].append(v)
    g[v].append(u)  # 无向边

# 初始化
sum_sz = [0] * (n + 1)  # 子树大小
max_part = [0] * (n + 1)  # 删除节点后的最大连通分量
min_max_part = n  # 最小最大连通分量大小
ans_node = n + 1  # 重心节点

# DFS
dfs(1, 0, g, n, sum_sz, max_part)
```

⑥树上距离和最小的点（换根dp） 其实就是重心

（找N个节点的树上的某个结点，使所有节点到该点距离之和最小）

- 选择一个节点作为根，计算所有节点到根的距离和。
- 使用换根DP，当根从节点 u 转移到其邻接节点 v 时，距离和的变化可以高效计算：
  - 原本以 u 为根，距离和为 $dis[u]$。
  - 换到 v 为根后，v 的子树节点到根的距离减少 1，其他节点到根的距离增加 1。
  - 因此，$dis[v] = dis[u] + (n - sum[v]) - sum[v] = dis[u] + n - 2 \cdot sum[v]$，其中 $sum[v]$ 是以 v 为根的子树节点数。
- 通过一次DFS预处理子树大小 $sum[]$，再用第二次DFS或递归计算每个节点的距离和，找出最小值和对应节点。

```python
from collections import defaultdict
import sys
sys.setrecursionlimit(1000000)

def dfs1(u, fa, g, sum_sz, dis):
    sum_sz[u] = 1
    for v in g[u]:
        if v != fa:
            dfs1(v, u, g, sum_sz, dis)
            sum_sz[u] += sum_sz[v]
            dis[u] += dis[v] + sum_sz[v]

def dfs2(u, fa, g, n, sum_sz, dis):
    global min_dis, ans
    if dis[u] < min_dis or (dis[u] == min_dis and u < ans):
        min_dis = dis[u]
        ans = u
    for v in g[u]:
        if v != fa:
            dis[v] = dis[u] + (n - 2 * sum_sz[v])
            dfs2(v, u, g, n, sum_sz, dis)
```

初始化 dis[1]

对u其子树的所有节点要比u多走1步

⑤ 拓扑排序+dp　求有向图中一条路径上出现次数最多的颜色的次数最大值 (all路径, all颜色)

用入度法拓扑排序, 按顺序对每个颜色. 节点 DP(二维列表)