树

树的直径

树上两个结点之间最长的一条简单路径是树的直径。

两次 dfs

1. 从任意结点 x 开始进行第一次 dfs，找到距离 x 最远的结点，记作 y；
2. 从 y 结点开始进行第二次 dfs，找到距离 y 最远的结点，记作 z。

答案（直径）为：$y \rightsquigarrow z$

局限性：树上边权非负。

```
int fur, dep[maxn];
void dfs(int u,int f)
{
    dep[u] = dep[f]+1;
    for(int i=Head[u];~i;i=Edge[i].next)
    {
        int v = Edge[i].to;
        if(v==f) continue;
        dfs(v,u);
        if(dep[v]>dep[fur]) fur = v;
    }
}
dfs(1,0); dfs(fur,0);
ans = dep[fur]-1;
```

树上 dp

任一根结点（1 号）

- 设计状态：$dp[u]$ 代表以 1 号结点为根结点时，从结点 u 往其子树走能够到达的最远距离。
- 初始状态：$dp[l] = 0$
- 状态转移方程：

$$dp[u] = \max_{v \in son[u]} (dp[v] + w(u, v))$$

经过 u 结点的最长链的长度，记作 $L[u]$，有：

$$\begin{aligned} L[u] &= \max_{v_1, v_2 \in son[u]} (dp[v_1] + w(u, v_1) + w(u, v_2) + dp[v_2]) \\ &= \max_{v_1 \in son[u]} (dp[v_1] + w(u, v_1) + \max_{v_2 \in son[u]} (dp[v_2] + w(u, v_2)) \qquad (v_1 \neq v_2) \\ &= \max_{v_1, v_2 \in son[u]} (dp[v_1] + w(u, v_1) + dp[u]) \end{aligned}$$

答案（直径）为：$\max(L[u])$

```cpp
int ans, dp[maxn];
void dfs(int u,int f)
{
    for(int i=Head[u];~i;i=Edge[i].next)
    {
        int v = Edge[i].to;
        if(v==f) continue;
        dfs(v,u);
        ans = max(ans, dp[v]+1+dp[u]);
        dp[u] = max(dp[u],dp[v]+1);
    }
}
dfs(1,0);
```

树的重心

不带点权

```cpp
int n,rt,siz[maxn],maxp[maxn];
void dfs1(int u,int f)
{
    siz[u] = 1;
    for(int i=Head[u];~i;i=Edge[i].next)
    {
        int v = Edge[i].to;
        if(v==f) continue;
        dfs1(v,u);
        siz[u] += siz[v];
        maxp[u] = max(maxp[u], siz[v]);
    }
    maxp[u] = max(maxp[u],n-siz[u]);
    if(maxp[u]<maxp[rt]) rt = u;
}
ll ans;
void dfs2(int u,int f,int dep)
{
    ans += dep;
    for(int i=Head[u];~i;i=Edge[i].next)
    {
        int v = Edge[i].to;
        if(v==f) continue;
        dfs2(v,u,dep+1);
    }
}
maxp[rt] = n;
dfs1(1,0); dfs2(rt,0,0);
```

带点权

解决方法：树上 DP

有如下定义：

1. 权值：$v[u]$
2. 深度：$dep[u]$（根结点为 0）
3. 子树大小（权值和）：$siz[u]$

- 设计状态：$dp[u]$ 代表以 u 作为根结点时的答案（总距离）

- 初始状态:

$$dp[rt] = \sum_{u \in V} v[u] \times dep[u]$$

- 状态转移: $dp[u] = dp[f] - siz[u] + (siz[1] - siz[u])$
- 结果:

$$\min_{u \in V}(dp[u])$$

```
int v[maxn], siz[maxn];
ll dp[maxn];
void dfs1(int u,int f=0,int dep=0)
{
    siz[u] = v[u];
    for(int i=Head[u];~i;i=Edge[i].next)
    {
        int v = Edge[i].to;
        if(v==f) continue;
        dfs1(v,u,dep+1);
        siz[u] += siz[v];
    }
    dp[1] += v[u]*dep;
}
ll ans = inf;
void dfs2(int u,int f=0)
{
    ans = min(ans, dp[u]);
    for(int i=Head[u];~i;i=Edge[i].next)
    {
        int v = Edge[i].to;
        if(v==f) continue;
        dp[v] = dp[u]-siz[v]+siz[1]-siz[v];
        dfs2(v,u);
    }
}
dfs1(1); dfs2(1);
```

树上随机游走

设 $f(u)$ 代表 u 结点走到其父结点 p_u 的期望距离，则有:

$$f(u) = \sum_{(u,t) \in E} w(u,t) + \sum_{v \in son_u} f(v)$$

初始状态为 $f(leaf) = 1$。当树上所有边的边权都为 1 时，上式可化为:

$$f(u) = d(u) + \sum_{v \in son_u} f(v)$$

即 u 子树的所有结点的度数和，也即 u 子树大小的两倍 -1。

设 $g(u)$ 代表 p_u 结点走到其子结点 u 的期望距离，则有:

$$g(u) = \sum_{(p_u,t) \in E} w(p_u,t) + g(p_u) + \sum_{s \in sibling_u} f(s)$$

初始状态为 $g(root) = 0$。当树上所有边的边权都为 1 时，上式可化为:

$$g(u) = d(p_u) + g(p_u) + \sum_{s \in sibling_u} f(s)$$

```
int d[maxn], siz[maxn], ss[maxn], dp[maxn]; // 预处理 d 为结点度数
void dfs1(int u,int f)
{
    siz[u] = 1;
    for(auto v : G[u])
    {
        if(v==f) continue;
        dfs1(v,u);
        siz[u] += siz[v];
        ss[u] += 2*siz[v]-1;
    }
}
void dfs2(int u,int f)
{
    if(u!=1) dp[u] = dp[f]+d[f]+ss[f]-(2*siz[u]-1);
    for(auto v : G[u])
    {
        if(v==f) continue;
        dfs2(v,u);
    }
}
dfs1(1,0); dfs2(1,0);
//? f(u) = siz[u]*2-1, g[u] = dp[u]
```

树上背包

$dp[i][j]$ 表示 i 子树中在 j 的容量范围内，最大可以获得多少收益。

答案显然 $dp[root][W]$

```
function<void(int,int)> dfs = [&](int u,int f)
{
    for(auto v : G[u])
    {
        if(v==f) continue;
        dfs(v,u);
        for(int j=m;j>=0;j--)    // 当前子树使用多少容量
            for(int k=0;k<=j;k++)    // 给 v 子树分配 k 容量
                dp[u][j] = max(dp[u][j], dp[u][j-k]+dp[v][k]+v[u]);
    }
}
```

有依赖的树上背包

如果选择一个物品，则必须选择它的父结点

```
function<void(int)> dfs = [&](int u,int f)
{
    for(int i=w[u];i<=W;i++) dp[u][i] = v[u];    // 因为要选儿子必选自己所以先把自己选上
    for(auto v : G[u])
    {
        if(v==f) continue;
        dfs(v,u);
        for(int j=W;j>=w[u];j--)
```

```
            for(int k=0;k<=j-w[u];k++)
                dp[u][j] = max(dp[u][j], dp[u][j-k]+dp[v][k]);
    }
};
```

也可以利用 dfs 序 $O(n^2)$ 快速 dp 出来

```
int tim = 0;
vector<int> nfd(n+1), siz(n+1), pre(n+1);
function<void(int)> dfs = [&](int u)
{
    nfd[tim++] = u;
    siz[u] = 1;
    for(auto v : G[u])
    {
        pre[v] = pre[u]+w[u];
        dfs(v);
        siz[u] += siz[v];
    }
};
dfs(1);
vector<vector<int>> dp(tim+1, vector<int>(W+1));
for(int i=0;i<tim;i++)
{
    for(int j=pre[nfd[i]];j<=W-w[nfd[i]];j++)    // 选当前结点
        dp[i+1][j+w[nfd[i]]] = max(dp[i+1][j+w[nfd[i]]], dp[i][j]+v[nfd[i]]);
    for(int j=pre[nfd[i]];j<=W;j++)              // 不选当前结点
        dp[i+siz[nfd[i]]][j] = max(dp[i+siz[nfd[i]]][j], dp[i][j]);
}
cout<<dp[tim][W]<<endl;
```