

结论 1

1、树上最远距离

树上到每个点距离最远的距离一定为树上一条直径的两个端点之一。

分别从两个端点开始 dfs 即可 $O(n)$ 求取每个点的树上最远点。

证明见 [链接](#)

或者可以考虑树形 dp 第一次 dfs 维护每个节点子树方向上的最远距离和次远距离。

第二次 dfs 维护每个节点祖先方向上的最远距离，答案即为子树方向上的最远距离和祖先方向上的最远距离的较大者。

```
int dp[MAXN][3], hson[MAXN], dis[MAXN];
void dfs1(int u, int fa) {
    dp[u][0] = dp[u][1] = dp[u][2] = hson[u] = 0;
    for (int i = head[u]; i; i = edge[i].next) {
        int v = edge[i].to;
        if (v == fa) continue;
        dfs1(v, u);
        if (dp[v][0] + edge[i].w > dp[u][0]) {
            hson[u] = v;
            dp[u][1] = dp[u][0];
            dp[u][0] = dp[v][0] + edge[i].w;
        }
        else if (dp[v][0] + edge[i].w > dp[u][1])
            dp[u][1] = dp[v][0] + edge[i].w;
    }
}
void dfs2(int u, int fa) {
    dis[u] = max(dp[u][0], dp[u][2]);
    for (int i = head[u]; i; i = edge[i].next) {
        int v = edge[i].to;
        if (v == fa) continue;
        if (v == hson[u])
            dp[v][2] = edge[i].w + max(dp[u][1], dp[u][2]);
        else
            dp[v][2] = edge[i].w + max(dp[u][0], dp[u][2]);
        dfs2(v, u);
    }
}
```

2、树的遍历代价

给定一棵树，树上有一些关键点，问任意从树上选一点出发，遍历所有关键点后回到起点的最短路径长度

为多少。

建立虚树。先考虑虚树上的叶子结点 v 必为关键节点，设 $u = fa(v)$

发现不管 v 是否为起点，都需要遍历 $edge(u, v)$ 两次(出发与返回)。

而要达到 v 必须达到 u 于是考虑把 v 删去，把 u 作为关键节点，答案加上 $2 \times edge(u, v)$ 继续重复上述过程。

最终发现答案为虚树上边权和的两倍。

设关键点按原树上的 dfs 序排序后为 $u_1, u_2, u_3 \cdots u_k$

通过画图观察发现虚树上边权和的两倍恰好等于

$$2 \times (\text{dis}(u_1, u_2) + \text{dis}(u_2, u_3) + \cdots + \text{dis}(u_{k-1}, u_k) + \text{dis}(u_k, u_1))$$

而加入一个点 v 作为新关键点时只需要查询 dfs 序与 v 相邻的两个点，记为 u_i, u_{i+1}

$$\text{则答案增量为 } 2 \times (\text{dis}(u_i, v) + \text{dis}(v, u_{i+1}) - \text{dis}(u_i, u_{i+1}))$$

删除一个关键点操作类似。

3、连通分量转化

无向图的边双连通分量可以通过给所有边定向转化为强连通分量。

必要性证明：

如果图中存在桥，则无论怎么给桥定向，图最终均不能成为强连通分量。

充分性证明：

如果一个图是边双连通分量，考虑任选两个点，于是从其中一个点出发必有两条边不重复的路径到达另一个点。

考虑将这两条路径定向，使之构成一个有向环，然后缩点，继续重复上述操作，最终可以将整个图缩成一个点，即整个图变成一个强连通分量。

注意路径定向时只定向缩点间的路径，缩点内部点可以互达，所以缩点内部路径不需要再次定向，于是也不会产生矛盾。

4、平方和

$$\sum_{i=1}^n i = m \text{ 的解只有 } (n, m) = (1, 1), (24, 70)$$

5、随机排序

随机打乱一个长度为 n 的循环，则循环的每个元素将等概率出现在长度为 $1 \sim n$ 的循环中。

考虑某个元素出现在长度为 s_i 的循环中的概率，有

$$p = \frac{C_{n-1}^{i-1} (i-1)! (n-i)!}{n!} = \frac{1}{n}$$

其中 C_{n-1}^{i-1} 表示从其他 $n-1$ 个元素中选择 $i-1$ 个元素与该元素构成循环 $(i-1)!$ 表示长度为 i 的循环的所有圆排列可能。

$(n-i)!$ 表示满足该元素处于长度为 i 的循环的条件后其他元素的排列可能 $n!$ 表示全部可能性。

6、组合数

$$\{n \choose m-1\} + \{n \choose m\} = \{n+1 \choose m\} \tag{1}$$

证明

$$\begin{aligned} & \frac{n!}{m!(n-m)!} + \frac{n!}{m!(n-m+1)!} = \frac{n!}{m!(n-m)!} \left(1 + \frac{1}{n-m+1} \right) \\ &= \frac{n!}{m!(n-m)!} \cdot \frac{n-m+2}{n-m+1} = \frac{(n+1)!}{(m+1)!(n-m)!} = \frac{(n+1)!}{(m+1)!(n-(m+1))!} \\ &= \frac{(n+1)!}{(m+1)!(n-m-1)!} = \frac{(n+1)!}{(m+1)!(n-(m+1))!} = \frac{(n+1)!}{(m+1)!(n-m-1)!} = \frac{(n+1)!}{(m+1)!(n-m-1)!} \end{aligned}$$

$m > 0$ 时, 有

$$\frac{1}{(1-x)^{m+1}} = \frac{1}{(1-x)^m} \frac{1}{(1-x)} \\ = \left(\sum_{n=0}^{\infty} \binom{m+n-1}{n} x^n \right) \left(\sum_{n=0}^{\infty} x^n \right) \\ = \sum_{n=0}^{\infty} x^n \sum_{i=0}^n \binom{m+i-1}{i} = \sum_{n=0}^{\infty} \binom{m+n}{n} x^n$$

From:
<https://wiki.cvbbacm.com/> - **CVBB ACM Team**

Permanent link:
https://wiki.cybbacm.com/doku.php?id=2020-2021:teams:legal_string:jxm2001:other:%E7%BB%93%E8%AE%BA 1

Last update: **2021/03/06 10:58**

