# STA—Station

> 给定一颗树，找出一个节点，使得把这个该节点作为根时，所有子节点的深度这和最大。

### 20min

- 由于数据范围比较大。显然易见的使用，一维dp
- 这样，对状态的设计角度做出了一定的指导。
- 一个不行就搞两个出来：
  - 遍历的过程中，所处理的对象的情况如何？$f_i$表示当前情景之下下方的情况。尝试采取回溯法的方式，进行及时的调整。
  - 先把1作为根部，处理出一系列的成果出来。然后去利用该成果进行一个调整。

---

### $solve$

- 总的思路差别不大。但是几个关键问题没只是几个关键的问题的解决情况。
  - 处理$f[i]$的顺序问题。(可以认为是子问题的解决顺序的问题。)
  - 发现迁移方程。
- 从最接近1的一些点来看。

$$f_v = f_1 - size[u] + (n - sinze[u]);$$

- 推广得到，假设当前把u。以v为根的父节点已经求出。

  $f_u = f_v - size[v] + n - size[v];$
- 发现过程之中，$szie[v]$是可以重复应用的。

---

## code

```
#include <bits/stdc++.h>
using namespace std;

void MAIN();
int main()
{
    ios::sync_with_stdio(false);
    cin.tie(nullptr), cout.tie(nullptr);
    MAIN();
}
typedef long long ll;
const int maxn = 1e6 + 10;
//------code-----٩(´ω`*)و ------靓仔代码-----٩(´ω`*)و ----talk is cheap , show me
the code--------

struct eee
{
    int no;
    int to;
```

```
} e[maxn << 1];
int head[maxn];
int tot;
void add(int x, int y)
{
    e[++tot].no = y;
    e[tot].to = head[x];
    head[x] = tot;
}

ll f[maxn];
ll ans = 0;
ll sum[maxn];
int n;

void dfs(int now, int fa, ll depth)
{
    for (int x = head[now]; x; x = e[x].to)
        if (e[x].no != fa)
        {
            dfs(e[x].no, now, depth + 1);
            f[now] += f[e[now].no];
            sum[now] += sum[e[x].no];
        }
    f[now] += depth;
    sum[now]++;
}

void ddfs(int now, int fa)
{
    if (fa != 0)
        f[now] = f[fa] + n - 2 * sum[now];
    for (int x = head[now]; x; x = e[x].to)
        if (e[x].no != fa)
            ddfs(e[x].no, now);
}

void MAIN(){

    cin >> n;
    for (int i = 1; i < n; i++){
        int x, y;
        cin >> x >> y;
        add(x, y), add(y, x);
    }
    dfs(1, 0, 0); //计算f[1],以及记录根的个数。
    ddfs(1, 0);
    ll mmax = 0;
    for (int i = n; i >= 1; i--)
        if (f[i] >= mmax)
        {
            mmax = f[i];
            ans = i;
        }
    cout << ans << '\n';
```

```
}
```