最近公共祖先

Lowest Common Ancestor

倍增法

Idea：倍增。询问 u, v 两点时，两点往上以 2 的幂次方的距离跳。

Complexity：$O(n\lg n)$ 初始化，$O(\lg n)$ 查询。

Application：询问 LCA；在维护、询问 `fa[][]` 时，可维护、询问其他信息，如路径最大值等。

Code：

```cpp
int fa[N][25], dep[N];
void dfs(int x, int f, int depth){
    dep[x] = depth;
    fa[x][0] = f;
    for(int i = head[x]; i; i = edge[i].nxt){
        if(edge[i].to == f) continue;
        dfs(edge[i].to, x, depth+1);
    }
}
void init(){
    for(int j = 1; (1 << j) <= n; j++)
        for(int i = 1; i <= n; i++)
            if(fa[i][j-1])
                fa[i][j] = fa[fa[i][j-1]][j-1];
}
int lca(int x, int y){
    if(dep[x] < dep[y])
        swap(x, y);
    for(int i = 20; i >= 0; i--)
        if(dep[x] - (1 << i) >= dep[y])
            x = fa[x][i];
    if(x == y)  return x;
    for(int i = 20; i >= 0; i--){
        if(fa[x][i] && fa[x][i] != fa[y][i]){
            x = fa[x][i];
            y = fa[y][i];
        }
    }
    return fa[x][0];
}
```

Tarjan 算法

Idea：离线算法，事先存储好所有的询问（我的代码中，每个点开一个 `vector` 记录询问的编号和询问的另一个点）。dfs 整棵树，过程中用并查集维护，每 dfs 完一颗子树后，并查集已把子树的所有点并到根上。

Complexity：$O(n+q)$（忽略并查集的复杂度）

ATT：常数大。

Code：

```cpp
int fa[N];
int findfa(int x){ return x == fa[x] ? x : fa[x] = findfa(fa[x]); }
inline void unionn(int x, int y){ fa[findfa(y)] = findfa(x); }

struct Query{
    int id, ver;
};
vector<Query> query[N];
int ans[N];

bool vis[N];
```

```
void dfs(int x, int f){
    vis[x] = true;
    for(auto k: query[x]){
        if(ans[k.id])   continue;
        if(!vis[k.ver]) continue;
        ans[k.id] = findfa(k.ver);
    }
    for(int i = head[x]; i; i = edge[i].nxt){
        if(edge[i].to == f) continue;
        dfs(edge[i].to, x);
        unionn(x, edge[i].to);
    }
}
```

欧拉序列转 RMQ 问题

Idea：对一棵树 dfs，每一次到达时都记录下来（无论是第一次到达还是回溯时到达），可以得到一个长度为 $2n-1$ 的序列，称之为欧拉序列。记欧拉序列为 $E[1...2n-1]$，每个节点的 dfs 序为 $dfn[i]$，那么我们可以把 LCA 问题转化为 RMQ 问题：

$$dfn[\text{LCA}(u,v)] = \min\{dfn[x] \mid x \in E[dfn[u]\ldots dfn[v]]\}$$

换句话说，得到欧拉序列 $E[1...2n-1]$ 后，可得到序列 $dfn[E]$，在该序列上求解 RMQ 问题便可得到 $dfn[\text{LCA}]$。

Complexity：$O(n)$ 转换，$O(n \lg n)$ 使用 st 表解决 RMQ 问题。

ATT：转换后的欧拉序列长度为 $2n-1$，注意数组的大小。

Code：

```
int Euler[N<<1], dfn[N], dfntot;
void dfs(int x, int f){ // get Euler array
    Euler[dfn[x] = ++dfntot] = x;
    for(int i = head[x]; i; i = edge[i].nxt){
        if(edge[i].to == f) continue;
        dfs(edge[i].to, x);
        Euler[++dfntot] = x;
    }
}

namespace RMQ{
    int rmq[N<<1][25], lg[N<<1];
    void pre(){
        lg[1] = 0, lg[2] = 1;
        for(int i = 3; i <= dfntot; i++)    lg[i] = lg[i/2] + 1;
    }
    void init(){
        for(int j = 1; (1 << j) <= dfntot; j++)
            for(int i = 1; i + (1 << j) - 1 <= dfntot; i++)
                rmq[i][j] = min(rmq[i][j-1], rmq[i+(1<<(j-1))][j-1]);
    }
    int query(int l, int r){
        int k = lg[r - l + 1];
        return min(rmq[l][k], rmq[r-(1<<k)+1][k]);
    }
};

inline int lca(int x, int y){
    int l = min(dfn[x], dfn[y]), r = max(dfn[x], dfn[y]);
    return Euler[RMQ::query(l, r)];
}

int main(){
    //...
    dfs(rt, 0); // get Euler array

    RMQ::pre();
    for(int i = 1; i <= dfntot; i++)    RMQ::rmq[i][0] = dfn[Euler[i]];
    RMQ::init();

    while(q--){
        int u, v; scanf("%d%d", &u, &v);
        printf("%d\n", lca(u, v));
    }
```

```
    return 0;
}
```