Tarjan相关

有向图 - 强连通分量 SCC

Idea：在 dfs **搜索树**上，记录每个节点的 dfs 序 `dfn[i]` 和其及其**子树中能通过非树边到达的最早的点**的 dfs 序 `low[i]`，视 `dfn[i] == low[i]` 的点 i 为强连通分量的"树根"。具体地，搜索时若搜到已访问过且仍在栈中的节点，说明形成了环，更新当前点 `low` 值；若没访问过，则搜索后在回溯时更新当前点 `low` 值。

Complexity：$O(V+E)$

ATT：由于原图不一定联通，应遍历每一个点，若没有搜到过就 `tarjan` 一次。

Code：

```
stack<int> sta;
bool insta[N];
int scc, belong[N], dfn[N], low[N], dfsClock;
void tarjan(int x){
    dfn[x] = low[x] = ++dfsClock;
    sta.push(x);
    insta[x] = 1;
    for(int i = head[x]; i; i = edge[i].nxt){
        if(!dfn[edge[i].to]){
            tarjan(edge[i].to);
            low[x] = min(low[x], low[edge[i].to]);
        }
        else if(insta[edge[i].to])
            low[x] = min(low[x], dfn[edge[i].to]);
    }
    if(dfn[x] == low[x]){
        scc++;
        while(1){
            int cur = sta.top();
            sta.pop();
            insta[cur] = 0;
            belong[cur] = scc;
            if(cur == x)
                break;
        }
    }
}
int main(){
    //...;
    for(int i = 1; i <= n; i++)
        if(!dfn[i])
            tarjan(i);

    // build new graph
    for(int i = 1; i <= n; i++)
        for(int j = head[i]; j; j = edge[j].nxt)
            if(belong[i] != belong[edge[j].to])
                naddEdge(belong[i], belong[edge[j].to]);
    //...;
}
```

无向图 - 割点

Idea：在**搜索树**上，对于树根，若其含有两个及以上个子树，则树根为割点；对于非树根，若一条边 (u, v) 满足 `low[v] >= dfn[u]`，则意味着 u 是割点。

Complexity：$O(V+E)$

Code：

```
bool iscut[N]; // iscut[i]==1 if edge[i] is a cut vertex
int dfn[N], low[N], dfsClock;
void tarjan(int x, int f){
    int son = 0;
    dfn[x] = low[x] = ++dfsClock;
    for(int i = head[x]; i; i = edge[i].nxt){
        if(!dfn[edge[i].to]){
            son++;
            tarjan(edge[i].to, x);
            low[x] = min(low[x], low[edge[i].to]);
            if(f == 0 && son > 1)
                iscut[x] = 1;
            if(f != 0 && low[edge[i].to] >= dfn[x])
                iscut[x] = 1;
        }
        else if(edge[i].to != f)
            low[x] = min(low[x], dfn[edge[i].to]);
    }
}
int main(){
    //...;
    for(int i = 1; i <= n; i++)
        if(!dfn[i])
            tarjan(i, 0);
    //...;
}
```

无向图 - 割边/桥

Idea：只需将割点判断条件件的 `low[v] >= dfn[u]` 改为 `low[v] > dfn[u]` 即可。

ATT：建图时 `edgeNum` 应从 `1` 开始方便对边打标记。

Complexity：$O(V+E)$

Code：

```
bool iscut[M<<1]; // iscut[i]==1 if edge i is a cut edge
int dfn[N], low[N], dfsClock;
void tarjan(int x, int f){
    dfn[x] = low[x] = ++dfsClock;
    for(int i = head[x]; i; i = edge[i].nxt){
        if(!dfn[edge[i].to]){
            tarjan(edge[i].to, x);
            low[x] = min(low[x], low[edge[i].to]);
            if(low[edge[i].to] > dfn[x])
                iscut[i] = iscut[i^1] = 1;
        }
        else if(edge[i].to != f)
            low[x] = min(low[x], dfn[edge[i].to]);
    }
}
int main(){
    //...;
    for(int i = 1; i <= n; i++)
        if(!dfn[i])
```

```
            tarjan(i, 0);
    //...;
}
```

无向图 - 点双连通分量

无向图 - 边双连通分量

Idea：边双连通分量其实就是不含割边的子图，所以用 Tarjan 求出无向图割边，标记出来，再 dfs 一遍不走割边即可。

另外，如果把边双连通分量缩成一个点，那么原图形成一棵树，树边即割边。

Code：

```
bool iscut[M<<1]; // iscut[i]==1 if edge i is a cut edge
int dfn[N], low[N], dfsClock;
void tarjan(int x, int f){
    dfn[x] = low[x] = ++dfsClock;
    for(int i = head[x]; i; i = edge[i].nxt){
        if(!dfn[edge[i].to]){
            tarjan(edge[i].to, x);
            low[x] = min(low[x], low[edge[i].to]);
            if(low[edge[i].to] > dfn[x])
                iscut[i] = iscut[i^1] = 1;
        }
        else if(edge[i].to != f)
            low[x] = min(low[x], dfn[edge[i].to]);
    }
}

int belong[N], tot;
void dfs(int x, int now){
    belong[x] = now;
    for(int i = head[x]; i; i = edge[i].nxt)
        if(!belong[edge[i].to] && !iscut[i])
            dfs(edge[i].to, now);
}

int main(){
    //...;
    for(int i = 1; i <= n; i++)
        if(!dfn[i])
            tarjan(i, 0);
    for(int i = 1; i <= n; i++)
        if(!belong[i])
            dfs(i, ++tot);

    // build new graph
    for(int i = 1; i <= n; i++){
        for(int j = head[i]; j; j = edge[j].nxt){
            int to = edge[j].to;
            if(belong[i] != belong[to])
                vec[belong[i]].push_back(belong[to]);
        }
    }
    //...;
}
```