

# 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:**

```
1  stack<int> sta;
2  bool insta[N];
3  int scc, belong[N], dfn[N], low[N], dfsClock;
4  void tarjan(int x){
5      dfn[x] = low[x] = ++dfsClock;
6      sta.push(x);
7      insta[x] = 1;
8      for(int i = head[x]; i; i = edge[i].nxt){
9          if(!dfn[edge[i].to]){
10             tarjan(edge[i].to);
11             low[x] = min(low[x], low[edge[i].to]);
12          }
13          else if(insta[edge[i].to])
14             low[x] = min(low[x], dfn[edge[i].to]);
15      }
16      if(dfn[x] == low[x]){
17          scc++;
18          while(1){
19              int cur = sta.top();
20              sta.pop();
21              insta[cur] = 0;
22              belong[cur] = scc;
23              if(cur == x)
24                 break;
25          }
26      }
27  }
28  int main(){
29      //...;
30      for(int i = 1; i <= n; i++){
31          if(!dfn[i])
32             tarjan(i);
33      }
34      // build new graph
35      for(int i = 1; i <= n; i++){
36          for(int j = head[i]; j; j = edge[j].nxt)
37             if(belong[i] != belong[edge[j].to])
38                 naddEdge(belong[i], belong[edge[j].to]);
39      }
40      //...;
```

## 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:**

```
1  bool iscut[N]; // iscut[i]==1 if edge[i] is a cut vertex
2  int dfn[N], low[N], dfsClock;
3  void tarjan(int x, int f){
4      int son = 0;
5      dfn[x] = low[x] = ++dfsClock;
6      for(int i = head[x]; i; i = edge[i].nxt){
7          if(!dfn[edge[i].to]){
8              son++;
9              tarjan(edge[i].to, x);
10             low[x] = min(low[x], low[edge[i].to]);
11             if(f == 0 && son > 1)
12                 iscut[x] = 1;
13             if(f != 0 && low[edge[i].to] >= dfn[x])
14                 iscut[x] = 1;
15         }
16         else if(edge[i].to != f)
17             low[x] = min(low[x], dfn[edge[i].to]);
18     }
19 }
20 int main(){
21     //...;
22     for(int i = 1; i <= n; i++){
23         if(!dfn[i])
24             tarjan(i, 0);
25     }
26     //...;
27 }
```

## 无向图 - 割边/桥

**Idea:** 只需将割点判断条件的 `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:**

```
1  bool iscut[M<<1]; // iscut[i]==1 if edge i is a cut edge
2  int dfn[N], low[N], dfsClock;
3  void tarjan(int x, int f){
4      dfn[x] = low[x] = ++dfsClock;
5      for(int i = head[x]; i; i = edge[i].nxt){
6          if(!dfn[edge[i].to]){
7              tarjan(edge[i].to, x);
8              low[x] = min(low[x], low[edge[i].to]);
9              if(low[edge[i].to] > dfn[x])
10                 iscut[i] = iscut[i^1] = 1;
11         }
12         else if(edge[i].to != f)
13             low[x] = min(low[x], dfn[edge[i].to]);
14     }
15 }
16 int main(){
17     //...;
18     for(int i = 1; i <= n; i++){
19         if(!dfn[i])
```

```

20         tarjan(i, 0);
21         //...;
22     }

```

## 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:**

```

1  bool iscut[M<<1]; // iscut[i]==1 if edge i is a cut edge
2  int dfn[N], low[N], dfsClock;
3  void tarjan(int x, int f){
4      dfn[x] = low[x] = ++dfsClock;
5      for(int i = head[x]; i; i = edge[i].nxt){
6          if(!dfn[edge[i].to]){
7              tarjan(edge[i].to, x);
8              low[x] = min(low[x], low[edge[i].to]);
9              if(low[edge[i].to] > dfn[x])
10                 iscut[i] = iscut[i^1] = 1;
11          }
12          else if(edge[i].to != f)
13              low[x] = min(low[x], dfn[edge[i].to]);
14      }
15  }
16
17  int belong[N], tot;
18  void dfs(int x, int now){
19      belong[x] = now;
20      for(int i = head[x]; i; i = edge[i].nxt)
21          if(!belong[edge[i].to] && !iscut[i])
22              dfs(edge[i].to, now);
23  }
24
25  int main(){
26      //...;
27      for(int i = 1; i <= n; i++){
28          if(!dfn[i])
29              tarjan(i, 0);
30      }
31      for(int i = 1; i <= n; i++){
32          if(!belong[i])
33              dfs(i, ++tot);
34      }
35      // build new graph
36      for(int i = 1; i <= n; i++){
37          for(int j = head[i]; j; j = edge[j].nxt){
38              int to = edge[j].to;
39              if(belong[i] != belong[to])
40                  vec[belong[i]].push_back(belong[to]);
41          }
42      }
43      //...;

```

