$Tarjan$

强连通分量

```
vector<int> low(n + 10), dfn(n + 10);
vector<int> stac(n + 10), pd(n + 10);
vector<int> col(n + 10);
int top = 0, id = 0, tot = 0;
auto tarjan = [&](auto self, int x) -> void {
    low[x] = dfn[x] = ++ id;
    stac[++ top] = x;
    pd[x] = 1;
    for(auto v : g[x]){
        if(dfn[v] == 0){
            self(self, v);
            low[x] = min(low[x], low[v]);
        }else if(pd[v] == 1){
            low[x] = min(low[x], low[v]);
        }
    }
    if(low[x] == dfn[x]){
        tot ++;
        while(stac[top + 1] != x){
            sum[tot] += nums[stac[top]];
            pd[stac[top]] = 0;
            col[stac[top --]] = tot;
        }
    }
    return ;
};
```

通过$tarjan$建立dfs树，将同一强连通分量的点标记上相同的颜色编号。

强连通分量是有向图的概念，是指强连通分量中删除一个点之后原分量依然联通。

缩点

```
vector<vector<int>> g1(n + 10);
for(int i = 1;i <= n;i ++){
    for(auto v : g[i]){
        if(col[i] == col[v]) continue;
        g1[col[i]].push_back(col[v]);
    }
}
```

求出强连通分量之后通过col编号重新建图得到DAG。

点双连通分量

```
vector<vector<int>> bcc;
vector<int> dfn(n + 10), low(n + 10), stck(n + 10);
int id = 0, top = 0;
auto tarjan = [&](auto self, int fa, int x) -> void {
    int son = 0;
    dfn[x] = low[x] = ++ id;
    stck[++ top] = x;
```

```
        for(auto v : g[x]){
            if(!dfn[v]){
                son ++;
                self(self, x, v);
                low[x] = min(low[x], low[v]);
                if(low[v] >= dfn[x]){
                    vector<int> tmp;
                    while(stck[top + 1] != v) tmp.push_back(stck[top --]);
                    tmp.push_back(x);
                    bcc.push_back(tmp);
                }
            }else if(v != fa){
                low[x] = min(low[x], dfn[v]);
            }
        }
        if(fa == -1 && son == 0){
            vector<int> tmp;
            tmp.push_back(x);
            bcc.push_back(tmp);
        }
        return ;
    };
```

点双连通分量：无向图连通分量中去掉任意一个点之后分量依然联通

基于割点求联通分量同时特判孤立点。

割点

```
vector<int> dfn(n + 10), low(n + 10), cut(n + 10);
int id = 0;
auto tarjan = [&](auto self, int fa, int x) -> void {
    dfn[x] = low[x] = ++ id;
    int chd = 0;
    for(auto v : g[x]){
        if(v == fa) continue;
        if(dfn[v] == 0){
            self(self, x, v);
            low[x] = min(low[x], low[v]);
            if(low[v] >= dfn[x] && fa != -1){
                cut[x] = 1;
            }
            if(fa == -1) chd ++;
        }
        low[x] = min(low[x], dfn[v]);
    }
    if(chd >= 2 && fa == -1) cut[x] = 1;
};
```

割点：删除这个点之后图不连通

dfs树父节点和子节点分开讨论分别求出割点标记。

点双连通分量缩点

```
vector<vector<int>> bcc;
vector<int> dfn(n + 10), low(n + 10), stck(n + 10);
vector<bool> cut(n + 10);
int id = 0, top = 0;
auto tarjan = [&](auto self, int fa, int x) -> void {
    int son = 0;
    dfn[x] = low[x] = ++ id;
```

```
        stck[++ top] = x;
        for(auto v : g[x]){
            if(!dfn[v]){
                son ++;
                self(self, x, v);
                low[x] = min(low[x], low[v]);
                if(low[v] >= dfn[x]){
                    vector<int> tmp;
                    while(stck[top + 1] != v) tmp.emplace_back(stck[top --]);
                    tmp.emplace_back(x);
                    bcc.emplace_back(tmp);
                    if(fa != -1) cut[x] = 1;
                }
            }else if(v != fa){
                low[x] = min(low[x], dfn[v]);
            }
        }
        if(fa == -1 && son == 0){
            vector<int> tmp;
            tmp.emplace_back(x);
            bcc.emplace_back(tmp);
        }
        if(son >= 2 && fa == -1) cut[x] = 1;
        return ;
    };
    for(int i = 1;i <= n;i ++) if(dfn[i] == 0) tarjan(tarjan, -1, i);
    int cnt = bcc.size();
    vector<int> cutid(n + 10);
    for(int i = 1;i <= n;i ++) if(cut[i]) cutid[i] = ++ cnt;
    vector<vector<int>> ng((n << 1) + 10);
    for(int i = 0;i < bcc.size();i ++){
        auto vec = bcc[i];
        for(auto v : vec){
            if(!cut[v]) continue;
            ng[cutid[v]].emplace_back(i + 1);
            ng[i + 1].emplace_back(cutid[v]);
        }
    }
```

$tarjan$求点双连通分量以及割点并通过割点连接其所在的点双连通分量。

边双连通分量

```
vector<int> dfn(n + 10), low(n + 10), stck(n + 10);
vector<vector<int>> bcc;
int id = 0, top = 0;
auto tarjan = [&](auto self, int fa, int x) -> void {
    dfn[x] = low[x] = ++ id;
    stck[++ top] = x;
    for(int i = head[x];~ i;i = nxt[i]){
        int v = to[i];
        if(!dfn[v]){
            self(self, i, v);
            low[x] = min(low[x], low[v]);
        }else if(fa != (i ^ 1)){
            low[x] = min(low[x], dfn[v]);
        }
    }
    if(dfn[x] == low[x]){
        vector<int> tmp;
        while(stck[top + 1] != x){
            tmp.push_back(stck[top --]);
        }
        bcc.push_back(tmp);
```

```
        }
        return ;
    };
```

边双连通分量：无向图中去掉任意一条边原来的连通分量依然联通

强制要求前向星建图，父节点存边的编号。

可以确定桥然后dfs得到也可以将点压入栈中然后弹出得到。

桥

```
vector<int> dfn(n + 10), low(n + 10), stck(n + 10);
vector<int> bridge(m * 2 + 10);
int id = 0, top = 0;
auto tarjan = [&](auto self, int fa, int x) -> void {
    dfn[x] = low[x] = ++ id;
    stck[++ top] = x;
    for(int i = head[x];~ i;i = nxt[i]){
        int v = to[i];
        if(!dfn[v]){
            self(self, i, v);
            low[x] = min(low[x], low[v]);
            if(low[v] > dfn[x]){
                bridge[i] = bridge[i ^ 1] = true;
            }
        }else if(fa != (i ^ 1)){
            low[x] = min(low[x], dfn[v]);
        }
    }
    return ;
};
```

桥：删除后原来联通的图不再联通。

强制前向星建图，父节点存边的编号。