Table of Contents

强连通分量

```
void init() {
    tot = dfs_clock = scc_cnt = 0;
    memset(head, 0, sizeof(head)); memset(dfn, 0, sizeof(dfn));
    memset(sccno, 0, sizeof(sccno));
}

void addEdge(int u, int v) {
    to[++tot] = v, nxt[tot] = head[u], head[u] = tot;
}

void tarjan(int u) {
    dfn[u] = low[u] = ++dfs_clock;
    S.push(u);
    for(int i = head[u]; i; i = nxt[i]) {
        int v = to[i];
        if(!dfn[v]) tarjan(v), low[u] = min(low[u], low[v]);
        else if(!sccno[v]) low[u] = min(low[u], dfn[v]);
    }
    if(low[u] == dfn[u]) {
        scc_cnt++;
        for(; ; ) {
            int x = S.top(); S.pop();
            sccno[x] = scc_cnt;
            if(x == u) break ;
        }
    }
}
```

2-SAT

字典序

逐一考虑每个没有赋值的变量，先假定为真，然后沿有向边标记所有能标记的节点，如果过程中发现某变量的两个对立节点都被标记，则不能为真。然后假定为假，再次进行标记，如果还是不能标记，则无解，不需要回溯。

```
/* poi2001 和平委员会
根据宪法，Byteland 民主共和国的公众和平委员会应该在国会中通过立法程序来创立。不幸的是，由于某些党派代表之间的不和睦而使得这件事存在障碍。

此委员会必须满足下列条件：
每个党派都在委员会中恰有1个代表，
如果2个代表彼此厌恶，则他们不能都属于委员会。

每个党在议会中有2个代表。代表从1编号到2n。 编号为2i-1和2i的代表属于第i个党派。
*/

int tot, c;
int to[M], nxt[M], head[N], mark[N], S[N];

void addEdge(int u, int v) {
    to[++tot] = v, nxt[tot] = head[u], head[u] = tot;
}

void init() {
    tot = 0; memset(mark, 0, sizeof(mark));
    memset(head, 0, sizeof(head));
}

bool dfs(int u) {
    if(mark[u^1]) return false;
    if(mark[u]) return true;
    mark[u] = true, S[c++] = u;
    for(int i = head[u]; i; i = nxt[i]) if(!dfs(to[i])) return false;
    return true;
}

void twoSAT(int n) {
    for(int i = 0; i < 2*n; i += 2) if(!mark[i] && !mark[i^1]) {
        c = 0;
        if(!dfs(i)) {
            while(c > 0) mark[S[--c]] = false;
            if(!dfs(i+1)) {
                printf("NIE\n"); return ;
            }
        }
    }
    for(int i = 0; i < 2*n; i++) if(mark[i]) printf("%d\n", i+1);
}
```

```
int main() {
    int n, m;
    while(scanf("%d%d", &n, &m) != EOF) {
        init();
        for(int i = 1, a, b; i <= m; i++) {
            scanf("%d%d", &a, &b);
            a--, b--;
            addEdge(a, b^1); addEdge(b, a^1);
        }
        twoSAT(n);
    }
}
```

强连通分量

对于每个析取范式$a \bigvee b$，连两条边$(\neg a, b)$和$(\neg b, a)$，然后求强连通分量，进行缩点

如果存在两个对立的变量在一个强连通分量里面，则无解，否则必有解

缩点之后的DAG里面，对于两个对立变量a和b，若a的拓扑序在后边，则a为真

```
bool twoSAT(int n) {
    for(int i = 1; i <= 2*n; i++) if(!dfn[i]) tarjan(i);
    for(int i = 1; i <= n; i++) if(sccno[i] == sccno[i+n]) {
        printf("NO\n"); return ;
    };
    printf("YES\n");
    for(int i = 1; i <= n; i++) {
        if(sccno[i] < sccno[i+n]) printf("true\n");
        else printf("false\n");
    }
}
```

双联通分量

割点

1. 树根是割点当且仅当它有两个及以上的孩子
2. 非根节点u是割点当且仅当u存在一个子节点v，v以及其子节点都没有反向边连向u的子节点

```
int tot, dfs_clock, bcc_cnt;
int to[M], nxt[M], head[N], dfn[N], low[N], iscut[N], bccno[N];

void init() {
    tot = dfs_clock = bcc_cnt = 0; memset(head, 0, sizeof(head));
    memset(dfn, 0, sizeof(dfn)); memset(iscut, 0, sizeof(iscut));
    memset(bccno, 0, sizeof(bccno));
}

void addEdge(int u, int v) {
    to[++tot] = v, nxt[tot] = head[u], head[u] = tot;
}

void dfs(int u, int fa = -1) {
    dfn[u] = low[u] = ++dfs_clock;
    int child = 0;
    for(int i = head[u]; i; i = nxt[i]) {
        int v = to[i];
        if(!dfn[v]) {
            child++;
            dfs(v, u);
            low[u] = min(low[u], low[v]);
            if(low[v] >= dfn[u]) iscut[u] = true;
        } else if(dfn[v] < dfn[u] && v != fa)
            low[u] = min(low[u], dfn[v]);
    }
    if(fa < 0 && child == 1) iscut[u] = 0;
}
```

点双联通分量

不同双联通分量之间最多只有一个公共点，且一定是割点。计算点双联通分量的过程和计算割点类似，用一个栈来保存在当前BCC中的边。

```
int tot, dfs_clock, bcc_cnt;
int to[M], nxt[M], head[N], dfn[N], low[N], iscut[N], bccno[N];
stack<pair<int, int> >stk;
vector<int>bcc[N];

void init() {
    tot = dfs_clock = bcc_cnt = 0; memset(head, 0, sizeof(head));
    memset(dfn, 0, sizeof(dfn)); memset(iscut, 0, sizeof(iscut));
    memset(bccno, 0, sizeof(bccno));
}
```

```
void addEdge(int u, int v) {
    to[++tot] = v, nxt[tot] = head[u], head[u] = tot;
}

void dfs(int u, int fa = -1) {
    dfn[u] = low[u] = ++dfs_clock;
    int child = 0;
    for(int i = head[u]; i; i = nxt[i]) {
        int v = to[i];
        if(!dfn[v]) {
            stk.push(make_pair(u, v)), child++;
            dfs(v, u);
            low[u] = min(low[u], low[v]);
            if(low[v] >= dfn[u]) {
                iscut[u] = true; bcc_cnt++;
                bcc[bcc_cnt].clear();
                for(; ; ) {
                    int a = stk.top().first, b = stk.top().second; stk.pop();
                    if(bccno[a] != bcc_cnt) {
                        bcc[bcc_cnt].push_back(a); bccno[a] = bcc_cnt;
                    }
                    if(bccno[b] != bcc_cnt) {
                        bcc[bcc_cnt].push_back(b); bccno[b] = bcc_cnt;
                    }
                    if(a == u && b == v) break ;
                }
            }
        } else if(dfn[v] < dfn[u] && v != fa) {
            stk.push(make_pair(u, v));
            low[u] = min(low[u], dfn[v]);
        }
    }
    if(fa < 0 && child == 1) iscut[u] = 0;
}
```

最大流

```
const int N = 4100, M = 200010;
int tot = 1, src, sink;
int to[M], _next[M], cap[M], head[N], pre[N], vis[N], cur[N], used[N];

void addEdge(int u, int v, int c) {
    to[++tot] = v, _next[tot] = head[u], cap[tot] = c, head[u] = tot;
}

void init() {
    tot = 1; memset(head, 0, sizeof(head));
}

bool BFS() {
    memset(vis, false, sizeof(vis));
    queue<int>q; q.push(src);
    vis[src] = true;
    while(!q.empty()) {
        int u = q.front(); q.pop();
        for(int i = head[u]; i; i = _next[i]) {
            int v = to[i];
            if(!cap[i] || vis[v]) continue ;
            vis[v] = true, pre[v] = pre[u]+1;
            q.push(v);
        }
    }
    return vis[sink];
}

int DFS(int u, int c) {
    if(u==sink || c==0) return c;
    int flow = 0, f;
    for(int &i = cur[u]; i; i = _next[i]) {
        int v = to[i];
        if(pre[v]==pre[u]+1 && (f=DFS(v, min(c, cap[i])))>0) {
            flow += f, c -= f, cap[i] -= f, cap[i^1] += f;
        }
    }
    return flow;
}

int maxFlow() {
    int flow = 0;
    while(BFS()) {
        memcpy(cur, head, sizeof(cur));
        flow += DFS(src, INT_INF);
    }
    return flow;
}
```