



---

## Stratosphere's XCPC Templates

---

南京大学

平流层 Stratosphere

October 10, 2024

# Contents

|                           |           |
|---------------------------|-----------|
| <b>0 Header 与约定</b>       | <b>1</b>  |
| <b>1 图论</b>               | <b>2</b>  |
| 1.1 欧拉回路 . . . . .        | 2         |
| 1.2 Tarjan-SCC . . . . .  | 3         |
| 1.3 点双 . . . . .          | 3         |
| 1.4 边双 . . . . .          | 3         |
| 1.5 2-SAT . . . . .       | 3         |
| 1.6 最大流 (Dinic) . . . . . | 4         |
| 1.7 最小费用最大流 . . . . .     | 4         |
| 1.8 三四元环计数 . . . . .      | 4         |
| 1.9 支配树 . . . . .         | 4         |
| <b>2 数论</b>               | <b>6</b>  |
| <b>3 数学</b>               | <b>7</b>  |
| <b>4 字符串</b>              | <b>8</b>  |
| <b>5 数据结构</b>             | <b>9</b>  |
| <b>6 计算几何</b>             | <b>10</b> |
| <b>7 三维计算几何</b>           | <b>11</b> |
| <b>8 杂项</b>               | <b>12</b> |

## 0 Header 与约定

# 1 图论

## 1.1 欧拉回路

```

1000 namespace Euler {
1001     bool directed;
1002     vector<pii> G[maxn];
1003     vector<int> ans;
1004     int vis[maxm];
1005     int dfs(int x) {
1006         vector<int> t;
1007         while (G[x].size()) {
1008             auto [to, id] = G[x].back();
1009             G[x].pop_back();
1010             if (!vis[abs(id)]) {
1011                 vis[abs(id)] = 1, t.push_back(dfs(to)), ans.push_back(id);
1012             }
1013         }
1014         for (int i = 1; i < t.size(); i++) {
1015             if (t[i] != x) ans.clear();
1016         }
1017         return t.size() ? t[0] : x;
1018     }
1019     int n, m;
1020     pii e[maxm];
1021     int deg[maxn], vv[maxn];
1022     void clr() {
1023         for (int i = 1; i <= n; i++) G[i].clear(), deg[i] = vv[i] = 0;
1024         for (int i = 1; i <= m; i++) vis[i] = 0;
1025         ans.clear();
1026         n = m = 0;
1027     }
1028     void addedge(int x, int y) {
1029         chkmax(n, x), chkmax(n, y);
1030         e[++m] = {x, y};
1031         if (directed) {
1032             G[x].push_back({y, m});
1033             ++deg[x], --deg[y], vv[x] = vv[y] = 1;
1034         } else {
1035             G[x].push_back({y, m});
1036             G[y].push_back({x, -m});
1037             ++deg[x], ++deg[y], vv[x] = vv[y] = 1;
1038         }
1039     }
1040     using vi = vector<int>;
1041     pair<vi, vi> work() {
1042         if (!m) return clr(), pair<vi, vi>{{1}, {}};
1043         int S = 1;
1044         for (int i = 1; i <= n; i++)
1045             if (vv[i]) S = i;
1046         for (int i = 1; i <= n; i++)
1047             if (deg[i] > 0 && deg[i] % 2 == 1) S = i;
1048         dfs(S);
1049         if ((int)ans.size() != m) return clr(), pair<vi, vi>();
1050         reverse(ans.begin(), ans.end());
1051         vi ver, edge = ans;
1052         if (directed) {
1053             ver = {e[ans[0]].fir};
1054             for (auto t : ans) ver.push_back(e[t].sec);
1055         } else {
1056             ver = {ans[0] > 0 ? e[ans[0]].fir : e[-ans[0]].sec};
1057             for (auto t : ans) ver.push_back(t > 0 ? e[t].sec : e[-t].fir);
1058         }
1059         clr();
1060         return {ver, edge};
1061     }
1062 } // namespace Euler

```

## 1.2 Tarjan-SCC

```

1000 void tarjan(int u) {
1001     dfn[u] = low[u] = ++tim;
1002     in[u] = 1;
1003     st[++top] = u;
1004     for (int v : G[u]) {
1005         if (!dfn[v])
1006             tarjan(v), ckmin(low[u], low[v]);
1007         else if (in[v])
1008             ckmin(low[u], dfn[v]);
1009     }
1010     if (dfn[u] == low[u]) {
1011         ++totc;
1012         int x;
1013         do { x = st[top--], in[x] = 0, bel[x] = totc; } while (x != u);
1014     }
1015 }

```

## 1.3 点双

```

1000 int T; // assign = n
1001 void tarjan(int u, int fa) {
1002     dfn[u] = low[u] = ++tim;
1003     stk[++top] = u;
1004     for (int v : G[u]) {
1005         if (v == fa) continue;
1006         if (!dfn[v])
1007             dfs(v, u), ckmin(low[u], low[v]);
1008         else
1009             ckmin(low[u], dfn[v]);
1010     }
1011     if (fa && low[u] >= dfn[fa]) {
1012         int y;
1013         ++T;
1014         do {
1015             y = stk[top--];
1016             G2[T].push_back(y), G2[y].push_back(T);
1017         } while (y != u);
1018         G2[T].push_back(fa), G2[fa].push_back(T);
1019     }
1020 }

```

## 1.4 边双

```

1000 void tarjan(int u, int f) {
1001     dfn[u] = low[u] = ++tim;
1002     for (int v : G[u]) {
1003         int v = e[i].to;
1004         if (v == f) continue;
1005         if (!dfn[v]) {
1006             tarjan(v, u);
1007             ckmin(low[u], low[v]);
1008             if (low[v] > dfn[u]) vis[i] = vis[i ^ 1] = 1; // cut edge
1009         } else
1010             ckmin(low[u], dfn[v]);
1011     }
1012 }

```

## 1.5 2-SAT

构造方案时可以通过变量在图中的拓扑序确定该变量的取值。

如果变量  $x$  的拓扑序在  $\neg x$  之后, 那么取  $x$  值为真。

因为 Tarjan 算法求强连通分量时使用了栈, 所以 Tarjan 求得的 SCC 编号相当于反拓扑序。

```

1000 for (int i = 1; i <= n; i++)
1001 | if (bel[i << 1] == bel[i << 1 | 1]) return puts("IMPOSSIBLE"), 0;
1002 puts("POSSIBLE");
1003 for (int i = 1; i <= n; i++) printf("%d ", bel[i << 1] > bel[i << 1 | 1]);

```

## 1.6 最大流 (Dinic)

## 1.7 最小费用最大流

## 1.8 三四元环计数

```

1000 static int id[maxn], rnk[maxn];
1001 for (int i = 1; i <= n; i++) id[i] = i;
1002 sort(id + 1, id + n + 1, [](int x, int y) {
1003 | return pii{deg[x], x} < pii{deg[y], y};
1004 });
1005 for (int i = 1; i <= n; i++) rnk[id[i]] = i;
1006 for (int i = 1; i <= n; i++)
1007 | for (int v : G[i])
1008 | | if (rnk[v] > rnk[i]) G2[i].push_back(v);
1009 int ans3 = 0; // 3-cycle
1010 for (int i = 1; i <= n; i++) {
1011 | static int vis[maxn];
1012 | for (int v : G2[i]) vis[v] = 1;
1013 | for (int v1 : G2[i])
1014 | | for (int v2 : G2[v1])
1015 | | | if (vis[v2]) ++ans3; // (i,v1,v2)
1016 | for (int v : G2[i]) vis[v] = 0;
1017 }
1018 ll ans4 = 0; // 4-cycle
1019 for (int i = 1; i <= n; i++) {
1020 | static int vis[maxn];
1021 | for (int v1 : G[i])
1022 | | for (int v2 : G2[v1])
1023 | | | if (rnk[v2] > rnk[i]) ans4 += vis[v2], vis[v2]++;
1024 | for (int v1 : G[i])
1025 | | for (int v2 : G2[v1]) vis[v2] = 0;
1026 }

```

## 1.9 支配树

DAG 支配树

```

1000 namespace DomTree_DAG {
1001 | int idom[maxn];
1002 | vector<int> G[maxn], ANS[maxn]; // ANS: final tree
1003 | int deg[maxn];
1004 | int fa[maxn][25], dep[maxn];
1005 | int lca(int x, int y) {
1006 | | if (dep[x] < dep[y]) swap(x, y);
1007 | | for (int i = 20; i >= 0; i--)
1008 | | | if (fa[x][i] && dep[fa[x][i]] >= dep[y]) x = fa[x][i];
1009 | | if (x == y) return x;
1010 | | for (int i = 20; i >= 0; i--)
1011 | | | if (fa[x][i] != fa[y][i]) x = fa[x][i], y = fa[y][i];
1012 | | return fa[x][0];
1013 | }
1014 | void work() {
1015 | | queue<int> q;

```

```

1016 |     q.push(1);
1017 |     while (!q.empty()) {
1018 |         int x = q.front();
1019 |         q.pop();
1020 |         ANS[idom[x]].push_back(x);
1021 |         fa[x][0] = idom[x];
1022 |         dep[x] = dep[idom[x]] + 1;
1023 |         for (int i = 1; i <= 20; i++) fa[x][i] = fa[fa[x][i - 1]][i - 1];
1024 |         for (int v : G[x]) {
1025 |             --deg[v];
1026 |             if (!deg[v]) q.push(v);
1027 |             if (!idom[v])
1028 |                 idom[v] = x;
1029 |             else
1030 |                 idom[v] = lca(idom[v], x);
1031 |         }
1032 |     }
1033 | }
1034 | // namespace DomTree_DAG
1035 | namespace DomTree {
1036 |     vector<int> V[sz], rV[sz];
1037 |     int dfn[sz], id[sz], anc[sz], cnt;
1038 |     void dfs(int x) {
1039 |         id[dfn[x] = ++cnt] = x;
1040 |         for (int v : V[x])
1041 |             if (!dfn[v]) {
1042 |                 BuildTree::V[x].push_back(v);
1043 |                 BuildTree::deg[v]++;
1044 |                 anc[v] = x;
1045 |                 dfs(v);
1046 |             }
1047 |     }
1048 |     int fa[sz], mn[sz];
1049 |     int find(int x) {
1050 |         if (x == fa[x]) return x;
1051 |         int tmp = fa[x];
1052 |         fa[x] = find(fa[x]);
1053 |         chkmin(mn[x], mn[tmp]);
1054 |         return fa[x];
1055 |     }
1056 |     int semi[sz];
1057 |     void work() {
1058 |         dfs(1);
1059 |         rep(i, 1, n) fa[i] = i, mn[i] = 1e9, semi[i] = i;
1060 |         drep(w, n, 2) {
1061 |             int x = id[w];
1062 |             int cur = 1e9;
1063 |             if (w > cnt) continue;
1064 |             for (int v : rV[x]) {
1065 |                 if (!dfn[v]) continue;
1066 |                 if (dfn[v] < dfn[x])
1067 |                     chkmin(cur, dfn[v]);
1068 |                 else
1069 |                     find(v), chkmin(cur, mn[v]);
1070 |             }
1071 |             semi[x] = id[cur];
1072 |             mn[x] = cur;
1073 |             fa[x] = anc[x];
1074 |             BuildTree::V[semi[x]].push_back(x);
1075 |             BuildTree::deg[x]++;
1076 |         }
1077 |     }
1078 |     void link(int x, int y) {
1079 |         V[x].push_back(y), rV[y].push_back(x);
1080 |     }
1081 | } // namespace DomTree

```

## 2 数论

Delete This

---



### 3 数学

Delete This

|  |
|--|
|  |
|--|

## 4 字符串

Delete This

|  |
|--|
|  |
|--|

## 5 数据结构

Delete This

|  |
|--|
|  |
|--|

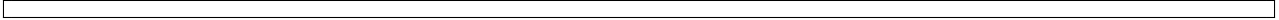
## 6 计算几何

Delete This

---

## 7 三维计算几何

Delete This



## 8 杂项

Delete This

|  |
|--|
|  |
|--|