



Stratosphere's XCPC Templates

南京大学

平流层 Stratosphere

October 16, 2024

Contents

0 Header	1
1 图论	2
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	4
1.6 最大流	4
1.7 最小费用最大流	6
1.8 匹配	7
1.8.1 二分图最大匹配-Hungary	7
1.8.2 二分图最大匹配-HK	8
1.8.3 二分图最大权匹配-KM	8
1.8.4 一般图最大匹配-带花树	9
1.8.5 一般图最大权匹配	10
1.9 最短路相关	10
1.9.1 差分约束	10
1.9.2 最小环	10
1.9.3 Steiner 树	10
1.10 流和匹配的建模技巧	11
1.10.1 二分图相关	11
1.10.2 网络流相关	11
1.11 三四元环计数	11
1.12 支配树	12
1.13 图论计数	13
1.13.1 Prufer 序列	13
1.13.2 无标号树计数	13
1.13.3 有标号 DAG 计数	14
1.13.4 有标号连通简单图计数	14
1.13.5 生成树计数	14
1.13.6 BEST 定理	14
2 树论	15
2.1 快速 LCA	15
2.2 虚树	15
2.3 长链剖分	15
2.4 静态点分治	15
2.5 点分树	15
2.6 动态 dp	15
2.7 树上背包	15
3 数论	16
4 数学	17
5 字符串	18
6 数据结构	19
7 计算几何	20
8 杂项	21

0 Header

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     st[++top] = u;
1003     for (int v : G[u]) {
1004         if (v == f) continue;
1005         if (!dfn[v])
1006             tarjan(v, u), ckmin(low[u], low[v]);
1007         else
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 }

```

1.5 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 算法

```

1000 namespace Dinic {
1001 |   int N, S, T;
1002 |   struct edge {
1003 |   |   int to, nxt, cap;
1004 |   } e[maxm << 1];
1005 |   int head[maxn], cur[maxn], tot = 1;
1006 |   int d[maxn];
1007 |   void addedge(int u, int v, int c) {
1008 |   |   e[++tot] = (edge){v, head[u], c}, head[u] = tot;
1009 |   |   e[++tot] = (edge){u, head[v], 0}, head[v] = tot;
1010 |   }
1011 |   bool bfs(int S, int T) {
1012 |   |   queue<int> q;
1013 |   |   for (int i = 1; i <= N; i++) d[i] = 0;
1014 |   |   d[S] = 1;
1015 |   |   q.push(S);
1016 |   |   while (!q.empty()) {
1017 |   |   |   int u = q.front();
1018 |   |   |   q.pop();
1019 |   |   |   for (int i = head[u]; i; i = e[i].nxt) {
1020 |   |   |   |   int v = e[i].to;
1021 |   |   |   |   if (e[i].cap && !d[v]) {
1022 |   |   |   |   |   d[v] = d[u] + 1, q.push(v);
1023 |   |   |   |   |   if (v == T) return true;
1024 |   |   |   |   }
1025 |   |   |   }
1026 |   |   }
1027 |   |   return false;
1028 |   }
1029 |   int dfs(int u, int f) {
1030 |   |   if (u == T) return f;
1031 |   |   int r = f;
1032 |   |   for (int& i = cur[u]; i && r; i = e[i].nxt) {
1033 |   |   |   int v = e[i].to;
1034 |   |   |   if (e[i].cap && d[v] == d[u] + 1) {
1035 |   |   |   |   int x = dfs(v, min(e[i].cap, r));
1036 |   |   |   |   if (!x) d[v] = 0;
1037 |   |   |   |   e[i].cap -= x, e[i ^ 1].cap += x;
1038 |   |   |   |   r -= x;
1039 |   |   |   }
1040 |   |   }
1041 |   |   return f - r;
1042 |   }
1043 |   ll work(int _N, int _S, int _T) {
1044 |   |   N = _N, S = _S, T = _T;
1045 |   |   ll ans = 0;
1046 |   |   while (bfs(S, T)) {
1047 |   |   |   for (int i = 1; i <= N; i++) cur[i] = head[i];
1048 |   |   |
1049 |   |   |   ans += 1ll * dfs(S, INF);
1050 |   |   }
1051 |   |   return ans;
1052 |   }

```

```
1053 } // namespace Dinic
```

ISAP 算法

```
1000 namespace ISAP {
1001     int N, S, T;
1002     struct edge {
1003         int to, nxt, cap;
1004     } e[maxm << 1];
1005     int head[maxn], cur[maxn], gap[maxn], dis[maxn], tot = 1;
1006     void addedge(int u, int v, int w) {
1007         e[++tot] = {v, head[u], w}, head[u] = tot;
1008         e[++tot] = {u, head[v], 0}, head[v] = tot;
1009     }
1010     int ISAP(int u, int lim) {
1011         if (u == T) return lim;
1012         int res = 0;
1013         for (int& i = cur[u]; i; i = e[i].nxt) {
1014             int v = e[i].to;
1015             if (e[i].cap && dis[u] == dis[v] + 1) {
1016                 ll det = ISAP(v, min(lim, e[i].cap));
1017                 e[i].cap -= det, e[i ^ 1].cap += det;
1018                 lim -= det, res += det;
1019                 if (!lim) return res;
1020             }
1021         }
1022         cur[u] = head[u];
1023         if (!--gap[dis[u]]) dis[S] = N + 1;
1024         gap[++dis[u]]++;
1025         return res;
1026     }
1027     ll work(int _N, int _S, int _T) {
1028         S = _S, T = _T, N = _N;
1029         ll res = 0;
1030         while (dis[S] <= N) res += 1ll * ISAP(S, INF);
1031         return res;
1032     }
1033 } // namespace ISAP
```

HLPP 算法

```
1000 namespace HLPP { // by ProjectEMmm
1001     int N, S, T;
1002     struct edge {
1003         int to, nxt, cap;
1004     } e[maxm << 1];
1005     int head[maxn], tot = 1;
1006
1007     int d[maxn], num[maxn];
1008     stack<int> lib[maxn];
1009     ll ex[maxn];
1010     int level = 0;
1011     void addedge(int u, int v, int c) {
1012         e[++tot] = {v, head[u], c}, head[u] = tot;
1013         e[++tot] = {u, head[v], 0}, head[v] = tot;
1014     }
1015     int Push(int u) {
1016         bool init = (u == S);
1017         for (int i = head[u]; i; i = e[i].nxt) {
1018             const int &v = e[i].to, &c = e[i].cap;
1019             if (!c || init == false && d[u] != d[v] + 1) continue;
1020             ll k = init ? c : min((ll)c, ex[u]);
1021             if (v != S && v != T && !ex[v] && d[v] < INF)
1022                 lib[d[v]].push(v), level = max(level, d[v]);
1023             ex[u] -= k, ex[v] += k, e[i].cap -= k, e[i ^ 1].cap += k;
1024             if (!ex[u]) return 0;
1025         }
1026         return 1;
1027     }
```

```

1028 void Relabel(int x) {
1029     d[x] = INF;
1030     for (int i = head[x]; i; i = e[i].nxt)
1031         if (e[i].cap) d[x] = min(d[x], d[e[i].to]);
1032     if (++d[x] < N) {
1033         lib[d[x]].push(x);
1034         level = max(level, d[x]);
1035         ++num[d[x]];
1036     }
1037 }
1038 bool BFS() {
1039     for (int i = 1; i <= N; ++i) {
1040         d[i] = INF;
1041         num[i] = 0;
1042     }
1043     queue<int> q;
1044     q.push(T), d[T] = 0;
1045     while (!q.empty()) {
1046         int u = q.front();
1047         q.pop();
1048         num[d[u]]++;
1049         for (int i = head[u]; i; i = e[i].nxt) {
1050             const int& v = e[i].to;
1051             if (e[i ^ 1].cap && d[v] > d[u] + 1) d[v] = d[u] + 1, q.push(v);
1052         }
1053     }
1054     return d[S] != INF;
1055 }
1056 int Select() {
1057     while (lib[level].size() == 0 && level > -1) level--;
1058     return level == -1 ? 0 : lib[level].top();
1059 }
1060 ll work(int _N, int _S, int _T) {
1061     N = _N, S = _S, T = _T;
1062     if (!BFS()) return 0;
1063     d[S] = N;
1064     Push(S);
1065     int x;
1066     while (x = Select()) {
1067         lib[level].pop();
1068         if (!Push(x)) continue;
1069         if (!--num[d[x]])
1070             for (int i = 1; i <= N; ++i)
1071                 if (i != S && i != T && d[i] > d[x] && d[i] < N + 1)
1072                     d[i] = N + 1;
1073         Relabel(x);
1074     }
1075     return ex[T];
1076 }
1077 } // namespace HLPP

```

1.7 最小费用最大流

```

1000 namespace MCMF {
1001     using pr = pair<ll, int>;
1002     int N, S, T;
1003     struct edge {
1004         int to, nxt, cap, w;
1005     } e[maxm << 1];
1006     int head[maxn], tot = 1;
1007     void addedge(int x, int y, int cap, int w) {
1008         e[++tot] = {y, head[x], cap, w}, head[x] = tot;
1009         e[++tot] = {x, head[y], 0, -w}, head[y] = tot;
1010     }
1011     ll d[maxn], dis[maxn];
1012     int vis[maxn], fr[maxn];
1013     bool spfa() {

```



```

1014 | queue<int> Q;
1015 | fill(d + 1, d + N + 1, 1e18); // CHECK
1016 | for (d[S] = 0, Q.push(S); !Q.empty();) {
1017 |     int x = Q.front();
1018 |     Q.pop();
1019 |     vis[x] = 0;
1020 |     for (int i = head[x]; i; i = e[i].nxt)
1021 |         if (e[i].cap && d[e[i].to] > d[x] + e[i].w) {
1022 |             d[e[i].to] = d[x] + e[i].w;
1023 |             fr[e[i].to] = i;
1024 |             if (!vis[e[i].to]) vis[e[i].to] = 1, Q.push(e[i].to);
1025 |         }
1026 |     }
1027 |     return d[T] < 1e17; // 如果只是最小费用流, 当d < 0继续增广
1028 | }
1029 | bool dijkstra() { // 正常题目不需要 dijk
1030 |     priority_queue<pr, vector<pr>, greater<pr>> Q;
1031 |     for (int i = 1; i <= N; ++i)
1032 |         dis[i] = d[i], d[i] = 1e18, vis[i] = fr[i] = 0; // CHECK
1033 |     Q.emplace(d[S] = 0, S);
1034 |     while (!Q.empty()) {
1035 |         int x = Q.top().second;
1036 |         Q.pop();
1037 |         if (vis[x]) continue;
1038 |         vis[x] = 1;
1039 |         for (int i = head[x]; i; i = e[i].nxt) {
1040 |             const ll v = e[i].w + dis[x] - dis[e[i].to];
1041 |             if (e[i].cap && d[e[i].to] > d[x] + v) {
1042 |                 fr[e[i].to] = i;
1043 |                 Q.emplace(d[e[i].to] = d[x] + v, e[i].to);
1044 |             }
1045 |         }
1046 |     }
1047 |     for (int i = 1; i <= N; ++i) d[i] += dis[i]; // CHECK
1048 |     return d[T] < 1e17;
1049 | }
1050 | std::pair<ll, ll> work(int _N, int _S, int _T) {
1051 |     N = _N, S = _S, T = _T;
1052 |     spfa(); // 如果初始有负权且要 dijk
1053 |     ll f = 0, c = 0;
1054 |     for (; dijkstra(); ) { // 正常可以用 spfa
1055 |         ll fl = 1e18;
1056 |         for (int i = fr[T]; i; i = fr[e[i ^ 1].to])
1057 |             fl = min((ll)e[i].cap, fl);
1058 |         for (int i = fr[T]; i; i = fr[e[i ^ 1].to])
1059 |             e[i].cap -= fl, e[i ^ 1].cap += fl;
1060 |         f += fl, c += fl * d[T];
1061 |     }
1062 |     return make_pair(f, c);
1063 | }
1064 | } // namespace MCMF

```

1.8 匹配

1.8.1 二分图最大匹配-Hungary

```

1000 | // 匈牙利, 左到右单向边, 0 (M !match!)
1001 | int vis[maxn], match[maxn];
1002 | bool dfs(int u) {
1003 |     for (int v : G[u]) {
1004 |         if (vis[v]) continue;
1005 |         vis[v] = 1;
1006 |         if (!match[v] || dfs(match[v])) return match[v] = u, 1;
1007 |     }
1008 |     return 0;
1009 | }
1010 | int work() {

```

```

1011 |   for (int i = 1; i <= nl; i++)
1012 |   |   if (dfs(i)) fill(vis + 1, vis + nr + 1, 0);
1013 | }
1014 | // 匈牙利, 左到右单向边, bitset, O(n^2|matchl|w)
1015 | bitset<N> G[N], unvis;
1016 | int match[N];
1017 | bool dfs(int u) {
1018 | |   for (auto s = G[u];;) {
1019 | |   |   s &= unvis;
1020 | |   |   int v = s._Find_first();
1021 | |   |   if (v == N) return 0;
1022 | |   |   unvis.reset(v);
1023 | |   |   if (!match[v] || dfs(match[v])) return match[v] = u, 1;
1024 | |   }
1025 | |   return 0;
1026 | }
1027 | int work() {
1028 | |   unvis.set();
1029 | |   for (int i = 1; i <= nl; i++)
1030 | |   |   if (dfs(i)) unvis.set();
1031 | }

```

1.8.2 二分图最大匹配-HK

```

1000 | // HK, 左到右单向边, O(M \sqrt{|matchl|})
1001 | int matchl[maxn], matchr[maxn], a[maxn], p[maxn];
1002 | int HK() {
1003 | |   while (true) {
1004 | |   |   for (int i = 1; i <= nl; i++) a[i] = p[i] = 0;
1005 | |   |   queue<int> Q;
1006 | |   |   while (!Q.empty()) Q.pop();
1007 | |   |   for (int i = 1; i <= nl; i++)
1008 | |   |   |   if (!matchl[i]) a[i] = p[i] = i, Q.push(i);
1009 | |   |   int succ = 0;
1010 | |   |   while (!Q.empty()) {
1011 | |   |   |   int u = Q.front();
1012 | |   |   |   Q.pop();
1013 | |   |   |   if (matchl[a[u]]) continue;
1014 | |   |   |   for (int v : G[u]) {
1015 | |   |   |   |   if (!matchr[v]) {
1016 | |   |   |   |   |   for (succ = 1; v; u = p[u])
1017 | |   |   |   |   |   |   matchr[v] = u, swap(matchl[u], v);
1018 | |   |   |   |   |   break;
1019 | |   |   |   |   }
1020 | |   |   |   |   if (!p[matchr[v]])
1021 | |   |   |   |   |   Q.push(matchr[v]), p[matchr[v]] = u, a[matchr[v]] = a[u];
1022 | |   |   |   }
1023 | |   |   }
1024 | |   |   if (!succ) break;
1025 | |   }
1026 | }

```

1.8.3 二分图最大权匹配-KM

```

1000 | // KM 二分图最大权匹配 复杂度O(n^3)
1001 | namespace KM {
1002 | |   int nl, nr;
1003 | |   ll e[maxn][maxn], lw[maxn], rw[maxn], mnw[maxn];
1004 | |   int lpr[maxn], rpr[maxn], vis[maxn], fa[maxn];
1005 | |   void addedge(int x, int y, ll w) {
1006 | |   |   ckmax(e[x][y], w), ckmax(lw[x], w);
1007 | |   }
1008 | |   void work(int x) {
1009 | |   |   int xx = x;
1010 | |   |   for (int i = 1; i <= nr; i++) vis[i] = 0, mnw[i] = 1e18;

```

```

1011 | while (true) {
1012 |     for (int i = 1; i <= nr; i++)
1013 |         if (!vis[i] && mnw[i] >= lw[x] + rw[i] - e[x][i])
1014 |             ckmin(mnw[i], lw[x] + rw[i] - e[x][i]), fa[i] = x;
1015 |     ll mn = 1e18;
1016 |     int y = -1;
1017 |     for (int i = 1; i <= nr; i++)
1018 |         if (!vis[i] && mn >= mnw[i]) ckmin(mn, mnw[i]), y = i;
1019 |     lw[xx] -= mn;
1020 |     for (int i = 1; i <= nr; i++)
1021 |         if (vis[i])
1022 |             rw[i] += mn, lw[rpr[i]] -= mn;
1023 |         else
1024 |             mnw[i] -= mn;
1025 |     if (rpr[y])
1026 |         x = rpr[y], vis[y] = 1;
1027 |     else {
1028 |         while (y) rpr[y] = fa[y], swap(y, lpr[fa[y]]);
1029 |         return;
1030 |     }
1031 | }
1032 | }
1033 | void init(int _nl, int _nr) {
1034 |     nl = _nl, nr = _nr;
1035 |     if (nl > nr) nr = nl;
1036 |     for (int i = 1; i <= nl; i++) lw[i] = -1e18;
1037 |     for (int i = 1; i <= nl; i++)
1038 |         for (int j = 1; j <= nr; j++) e[i][j] = 0; // or -1e18
1039 | }
1040 | ll work() {
1041 |     for (int i = 1; i <= nl; i++) work(i);
1042 |     ll tot = 0;
1043 |     for (int i = 1; i <= nl; i++) tot += e[i][lpr[i]];
1044 |     return tot;
1045 | }
1046 | } // namespace KM

```

1.8.4 一般图最大匹配-带花树

```

1000 | namespace blossom {
1001 |     vector<int> G[maxn];
1002 |     int f[maxn];
1003 |     int n, match[maxn];
1004 |     int getfa(int x) {
1005 |         return f[x] == x ? x : f[x] = getfa(f[x]);
1006 |     }
1007 |     void addedge(int x, int y) {
1008 |         G[x].push_back(y), G[y].push_back(x);
1009 |     }
1010 |     int pre[maxn], mk[maxn];
1011 |     int vis[maxn], T;
1012 |     queue<int> q;
1013 |     int LCA(int x, int y) {
1014 |         T++;
1015 |         for (; x = pre[match[x]], swap(x, y))
1016 |             if (vis[x = getfa(x)] == T)
1017 |                 return x;
1018 |         else
1019 |             vis[x] = x ? T : 0;
1020 |     }
1021 |     void flower(int x, int y, int z) {
1022 |         while (getfa(x) != z) {
1023 |             pre[x] = y;
1024 |             y = match[x];
1025 |             f[x] = f[y] = z;
1026 |             x = pre[y];
1027 |             if (mk[y] == 2) q.push(y), mk[y] = 1;

```

```

1028 |     }
1029 | }
1030 | void aug(int s) {
1031 |     for (int i = 1; i <= n; i++) pre[i] = mk[i] = vis[i] = 0, f[i] = i;
1032 |     q = {};
1033 |     mk[s] = 1;
1034 |     q.push(s);
1035 |     while (q.size()) {
1036 |         int x = q.front();
1037 |         q.pop();
1038 |         for (int v : G[x]) {
1039 |             int y = v, z;
1040 |             if (mk[y] == 2) continue;
1041 |             if (mk[y] == 1)
1042 |                 z = LCA(x, y), flower(x, y, z), flower(y, x, z);
1043 |             else if (!match[y]) {
1044 |                 for (pre[y] = x; y;)
1045 |                     x = pre[y], match[y] = x, swap(y, match[x]);
1046 |                 return;
1047 |             } else
1048 |                 pre[y] = x, mk[y] = 2, q.push(match[y]), mk[match[y]] = 1;
1049 |         }
1050 |     }
1051 | }
1052 | int work() {
1053 |     for (int i = 1; i <= n; i++) if (!match[i]) aug(i);
1054 |     int res = 0;
1055 |     for (int i = 1; i <= n; i++) res += match[i] > i;
1056 |     return res;
1057 | }
1058 | } // namespace blossom

```

1.8.5 一般图最大权匹配

待补充

1.9 最短路相关

1.9.1 差分约束

x_i 向 x_j 连一条权值为 c 的有向边表示 $x_j - x_i \leq c$ 。
用 BF 判断是否存在负环, 存在即无解。

1.9.2 最小环

记原图中 u, v 之间边的边权为 $val(u, v)$ 。

我们注意到 Floyd 算法有一个性质: 在最外层循环到点 k 时 (尚未开始第 k 次循环), 最短路数组 dis 中, $dis_{u,v}$ 表示的是从 u 到 v 且仅经过编号在 $[1, k]$ 区间中的点的最短路。

由最小环的定义可知其至少有三个顶点, 设其中编号最大的顶点为 w , 环上与 w 相邻两侧的两个点为 u, v , 则在最外层循环枚举到 $k = w$ 时, 该环的长度即为 $dis_{u,v} + val(v, w) + val(w, u)$ 。

故在循环时对于每个 k 枚举满足 $i < k, j < k$ 的 (i, j) , 更新答案即可。

1.9.3 Steiner 树

状态设计: $dp(i, S)$ 以 i 为根, 树中关键点集合为 S 的最小值。

1. 树根度数不为 1, 考虑拆分成两个子集 $T, S - T$:

$$dp(i, S) \leftarrow dp(i, S - T) + dp(i, T)$$

2. 树根度数为 1:

$$dp(i, S) \leftarrow dp(j, S) + w(i, j)$$

相当于超级源到每个顶点距离为 $dp(i, S)$, 求到每个顶点的最短路, dij 即可。

1.10 流和匹配的建模技巧

1.10.1 二分图相关

- 二分图最小点覆盖：等于最大匹配 $|match|$ 。从每一个非匹配点出发，沿着非匹配边正向进行遍历，沿着匹配边反向进行遍历到的点进行标记。选取左部点中没有被标记过的点，右部点中被标记过的点，则这些点可以形成该二分图的最小点覆盖。
- 二分图最大独立集：等于 $n - |match|$ ，考虑最小点覆盖给所有边都至少有一边有点，取反后必然为最大独立集。
- 二分图最小边覆盖：等于 $n - |match|$ ，考虑最坏情况每个顶点都要一条边，一个匹配能减小 1 的贡献。
- 最大团：等于补图的最大独立集。
- 最小路径覆盖：对于每条有向边 (u, v) ，拆成 $u \rightarrow v + n$ ， u 为进入 u ， $v + n$ 为从 v 离开，则答案为 $n - |match|$ 。
- Hall Theorem: 对于左部顶点集 X ， $\forall S \subseteq X, |N(S)| \geq |S| \iff$ 存在完美匹配。

1.10.2 网络流相关

- 二分图最大权独立集：考虑连边 (S, x, w_x) ，原图边 (x, y, ∞) ， (y, T, w_y) ，变为最小割。
- 最大权闭合子图：正权 w_u 连 (S, u, w_u) ，负权 w_v 连 $(v, T, -w_v)$ ，原图边连 ∞ 。此时最小割之后源点 S 能到达的点即为最大权闭合子图，答案即为正权和 $-\text{mincut}$ 。
- 无源汇上下界可行流：建源汇 S, T ， $l(u, v), r(u, v)$ 分别为流量上下界。记 $d(i) = \sum l(u, i) - \sum l(i, v)$ 。
 - 原边 (u, v) 连 $(u, v, r(u, v) - l(u, v))$ 。
 - 对于每个点 u ，若 $d_u > 0$ ，连 (S, u, d_u) 。
 - 若 $d_u < 0$ ，连 $(u, T, -d_u)$ 。

若 S 的出边全部流满则存在解。

- 有源汇上下界可行流：原图源汇连边 $(T \rightarrow S, (0, \infty))$ ，则转化为无源汇。
- 有源汇上下界最大流：从 T 到 S 连一条下界为 0，上界为 $+\infty$ 的边，转化为无源汇网络。按照无源汇上下界可行流的做法求一次无源汇上下界超级源 SS 到超级汇 TT 的最大流。删去所有附加边，在上一步的残量网络基础上，求一次 S 到 T 的最大流。两者之和即为答案。
- 有源汇上下界最小流：从 T 到 S 连一条下界为 0，上界为 $+\infty$ 的边，转化为无源汇网络。按照无源汇上下界可行流的做法求一次无源汇上下界超级源 SS 到超级汇 TT 的最大流。删去所有附加边，在上一步的残量网络基础上，求一次 T 到 S 的最大流。两者之差即为答案。
- 最小费用可行流：同有源汇上下界可行流，在超级源汇跑最小费用最大流，答案为费用 + 下界流量的费用。
- 平面图最小割 = 对偶图最短路

1.11 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 | }

```

1.12 支配树

```

1000 namespace Dom_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 Dom_DAG
1035 namespace Dom {
1036 |   vector<int> G[maxn], rG[maxn];
1037 |   int dfn[maxn], id[maxn], anc[maxn], cnt;
1038 |   void dfs(int x) {
1039 |   |   id[dfn[x] = ++cnt] = x;
1040 |   |   for (int v : G[x])
1041 |   |   |   if (!dfn[v]) {
1042 |   |   |   |   Dom_DAG::G[x].push_back(v);
1043 |   |   |   |   Dom_DAG::deg[v]++;
1044 |   |   |   |   anc[v] = x;
1045 |   |   |   |   dfs(v);
1046 |   |   |   }
1047 |   |   }
1048 |   int fa[maxn], mn[maxn];
1049 |   int find(int x) {
1050 |   |   if (x == fa[x]) return x;
1051 |   |   int tmp = fa[x];
1052 |   |   fa[x] = find(fa[x]);
1053 |   |   ckmin(mn[x], mn[tmp]);
1054 |   |   return fa[x];
1055 |   }
1056 |   int semi[maxn];
1057 |   void work() {
1058 |   |   dfs(1);

```

```

1059 |     for (int i = 1; i <= n; i++) fa[i] = i, mn[i] = 1e9, semi[i] = i;
1060 |     for (int w = n; w >= 2; w--) {
1061 |         int x = id[w];
1062 |         int cur = 1e9;
1063 |         if (w > cnt) continue;
1064 |         for (int v : rG[x]) {
1065 |             if (!dfn[v]) continue;
1066 |             if (dfn[v] < dfn[x])
1067 |                 ckmin(cur, dfn[v]);
1068 |             else
1069 |                 find(v), ckmin(cur, mn[v]);
1070 |         }
1071 |         semi[x] = id[cur];
1072 |         mn[x] = cur;
1073 |         fa[x] = anc[x];
1074 |         Dom_DAG::G[semi[x]].push_back(x);
1075 |         Dom_DAG::deg[x]++;
1076 |     }
1077 | }
1078 | void addedge(int x, int y) {
1079 |     G[x].push_back(y), rG[y].push_back(x);
1080 | }
1081 | } // namespace Dom

```

1.13 图论计数

1.13.1 Prufer 序列

有标号无根树和其 prufer 编码一一对应, 一颗 n 个点的树, 其 prufer 编码长度为 $n - 2$, 且度数为 d_i 的点在 prufer 编码中出现 $d_i - 1$ 次.

由树得到序列: 总共需要 $n - 2$ 步, 第 i 步在当前的树中寻找具有最小标号的叶子节点, 将与其相连的点的标号设为 Prufer 序列的第 i 个元素 p_i , 并将此叶子节点从树中删除, 直到最后得到一个长度为 $n - 2$ 的 Prufer 序列和一个只有两个节点的树.

由序列得到树: 先将所有点的度赋初值为 1, 然后加上它的编号在 Prufer 序列中出现的次数, 得到每个点的度; 执行 $n - 2$ 步, 第 i 步选取具有最小标号的度为 1 的点 u 与 $v = p_i$ 相连, 得到树中的一条边, 并将 u 和 v 的度减一. 最后再把剩下的两个度为 1 的点连边, 加入到树中.

推论:

- n 个点完全图, 要求每个点度数依次为 d_1, d_2, \dots, d_n , 这样生成树的棵数为: $\frac{(n-2)!}{\prod (d_i - 1)!}$
- 左边有 n_1 个点, 右边有 n_2 个点的完全二分图的生成树棵数为 $n_1^{n_2-1} \times n_2^{n_1-1}$
- m 个连通块, 每个连通块有 c_i 个点, 把他们全部连通的生成树方案数: $(\sum c_i)^{m-2} \prod c_i$

1.13.2 无标号树计数

(1) 有根树计数:

$$f_n = \frac{\sum_{i=1}^{n-1} f_{n-i} \sum_{d|i} f_d \cdot d}{n-1}$$

记 $g_i = \sum_{d|i} f_d \cdot d$ 即可做到 $\Theta(n^2)$ 。

(2) 无根树计数:

当 n 是奇数时

如果根不是重心, 必然存在恰好一个子树, 它的大小超过 $\left\lfloor \frac{n}{2} \right\rfloor$ (设它的大小为 k) 减去这种情况即可。

因此答案为

$$f_n - \sum_{k=\left\lfloor \frac{n}{2} \right\rfloor + 1}^{n-1} f_k \cdot f_{n-k}$$

当 n 是偶数时

有可能存在两个重心, 且其中一个为根 (即存在一棵子树大小恰为 $\frac{n}{2}$), 额外减去 $\left(f_{\frac{n}{2}}\right)$ 即可

1.13.3 有标号 DAG 计数

$$F_i = \sum_{j=1}^i \binom{i}{j} (-1)^{j+1} 2^{j(i-j)} F_{i-j}$$

想法是按照拓扑序分层，每次剥开所有入度为零的点。

1.13.4 有标号连通简单图计数

记 $g(n) = 2^{\binom{n}{2}}$ 为有标号简单图数量， $c(n)$ 为有标号简单连通图数量，那么枚举 1 所在连通块大小，有

$$g(n) = \sum_{i=1}^n \binom{n-1}{i-1} c(i) g(n-i)$$

易递推求 $c(n)$ 。多项式做法考虑 exp 组合意义即可。

1.13.5 生成树计数

Kirchhoff Matrix $T = Deg - A$, Deg 是度数对角阵, A 是邻接矩阵.

无向图度数矩阵是每个点度数; 有向图度数矩阵是每个点入度.

邻接矩阵 $A[u][v]$ 表示 $u \rightarrow v$ 边个数, 重边按照边数计算, 自环不计入度数.

无向图生成树计数: $c = |K|$ 的任意 1 个 $n-1$ 阶主子式 |

有向图外向树计数: $c = |$ 去掉根所在的那阶得到的主子式 |

若求边权和则邻接矩阵可以设为 $(1 + wx)$, 相当于一次项的系数。

1.13.6 BEST 定理

设 G 是有向欧拉图, k 为任意顶点, 那么 G 的不同欧拉回路总数 $ec(G)$ 是

$$ec(G) = t^{\text{root}}(k) \prod_{v \in V} (\deg(v) - 1)!$$

$t^{\text{root}}(k)$ 为以 k 为根的外向树个数。

2 树论

2.1 快速 LCA

查询 $[dfn_u + 1, dfn_v]$ 深度最小节点的父亲
 可以简化为在 ST 表的最底层记录父亲，比较时取时间戳较小的结点。
 取决于 st 表实现可以做到 $O(n)$ or $O(n \log n)$

```

1000 int getmin(int x, int y) {
1001 |     return dfn[x] < dfn[y] ? x : y;
1002 }
1003
1004 void dfs(int u, int f) {
1005 |     dfn[u] = ++tim;
1006 |     a[dfn[u]] = f;
1007 |     for (int v : e[u])
1008 |         if (v != f) dfs(v, u);
1009 }
1010 int lca(int u, int v) {
1011 |     if (u == v) return u;
1012 |     if ((u = dfn[u]) > (v = dfn[v])) swap(u, v);
1013 |     return RMQ(dfn[u] + 1, dfn[v]);
1014 }

```

2.2 虚树

2.3 长链剖分

2.4 静态点分治

2.5 点分树

2.6 动态 dp

2.7 树上背包

3 数论

Delete This

4 数学

Delete This

--

5 字符串

Delete This

6 数据结构

Delete This

7 计算几何

Delete This



8 杂项

Delete This

--