匹配与流

增广路

注：在每个testcase前需将nc置零，用到前几个点就`addv`几个点。

注2：费用流均为多路增广，返回的`pair<ll,ll>`中`first`为最大流，`second`为最小费用。

Hungrian

二分图最大匹配

时间复杂度：$O(nm)$

```
typedef pair<int, int> pii;

namespace matching {

vector<int> g[N];
int lnk[N]; bool vis[N];

bool dfs(int u) {
    vis[u] = 1;
    for (int v : g[u]) {
        if (lnk[v] == u || vis[lnk[v]]) continue;
        if (lnk[v] && !dfs(lnk[v])) continue;
        lnk[v] = u;
        return true;
    }
    return false;
}

vector<pii> match(int nl, int nr, const vector<pii>& es) {
    //  initialize
    for (int i = 1; i <= nl; ++i)
        g[i].clear();
    for (pii p : es)
        g[p.first].push_back(p.second);
    fill_n(lnk + 1, nr, 0);
    //  do matching
    for (int i = 1; i <= nl; ++i) {
        fill_n(vis + 1, nl, false);
        dfs(i);
    }
    //  get result
    vector<pii> res;
    for (int i = 1; i <= nr; ++i)
        if (lnk[i]) res.emplace_back(lnk[i], i);
    return res;
}

}
```

Hopcroft-Karp

二分图最大匹配

```
vector<int> g[N]; int n1, n2;
int lnk[N], dis[N], dm; bool vis[N];
void clr_hk(int n, int m) {
    n1 = n; n2 = m;
    for (int i = 1; i <= n1 + n2; ++i)
        g[i].clear();
}

bool bfs_hk() {
    queue<int> q; dm = INT_MAX;
    fill(dis + 1, dis + n1 + n2 + 1, -1);
    for (int i = 1; i <= n1; ++i)
        if (!lnk[i]) q.push(i), dis[i] = 0;
    while (!q.empty()) {
        int u = q.front(); q.pop();
        if (dis[u] > dm) break;
        for (int v : g[u]) {
            if (dis[v] != -1) continue;
            dis[v] = dis[u] + 1;
            if (!lnk[v]) dm = dis[v];
            else dis[lnk[v]] = dis[v] + 1, q.push(lnk[v]);
        }
    }
    return dm != INT_MAX;
}

int dfs_hk(int u) {
    for (int v : g[u]) {
        if (vis[v] || dis[v] != dis[u] + 1) continue;
        vis[v] = 1;
        if (lnk[v] && dis[v] == dm) continue;
        if (lnk[v] && !dfs_hk(lnk[v])) continue;
        lnk[v] = u; lnk[u] = v; return 1;
    }
    return 0;
}

int hk() {
    fill (lnk + 1, lnk + n1 + n2 + 1, 0);
    int res = 0;
    while (bfs_hk()) {
        fill (vis + 1, vis + n1 + n2 + 1, 0);
        for (int i = 1; i <= n1; ++i)
            if (!lnk[i] && dfs_hk(i)) res++;
    }
    return res;
}
```

Kuhn-Munkres

二分图最大权匹配

```
namespace matching {

vector<int> g[N];
int lnk[N], pre[N];
bool vis[N];

ll a[N], b[N], w[N][N], sl[N];

vector<pair<pii, ll>> match(int nl, int nr, const vector<pair<pii, ll>>& es) {
    nr = max(nl, nr);
    fill_n(lnk + 1, nr, 0);
    for (int i = 1; i <= nl; ++i)
        fill_n(w[i] + 1, nr, 0);
    for (pair<pii, ll> e : es)
        w[e.first.first][e.first.second] = max(w[e.first.first][e.first.second],
e.second);
    for (int i = 1; i <= nl; ++i)
        a[i] = *max_element(w[i] + 1, w[i] + nr + 1);
    fill_n(b + 1, nr, 0);
    for (int i = 1, j, u, vt = 0; i <= nl; ++i) {
        fill_n(vis + 1, nr, 0);
        fill_n(sl + 1, nr, inf);
        fill_n(pre + 1, nr, 0);
        lnk[0] = i;
        for (j = 0; u = lnk[j]; j = vt) {
            ll d = inf; vis[j] = 1;
            for (int v = 1; v <= nr; ++v) {
                ll t = a[u] + b[v] - w[u][v];
                if (vis[v]) continue;
                if (sl[v] > t) sl[v] = t, pre[v] = j;
                if (sl[v] < d) d = sl[v], vt = v;
            }
            for (int v = 0; v <= nr; ++v) {
                if (vis[v]) a[lnk[v]] -= d, b[v] += d;
                else sl[v] -= d;
            }
        }
        for (; j; j = pre[j]) lnk[j] = lnk[pre[j]];
    }
    vector<pair<pii, ll>> res;
    for (int i = 1; i <= nr; ++i)
        res.emplace_back(pii(lnk[i], i), a[lnk[i]] + b[i]);
    return res;
}

}
```

Dinic

最大流

```
const ll inf = LLONG_MAX;
struct edge { int v, p; ll c, f; };
vector<edge> g[N];
int cur[N], dis[N], nc;

int addv(int cnt) {
    while (cnt--) {
        int p = ++nc;
        g[p].resize(0);
    }
    return nc;
}

void adde(int u, int v, ll c) {
    g[u].push_back({ v, g[v].size(), c, 0 });
    g[v].push_back({ u, g[u].size() - 1, c, c });
}

bool bfs(int s, int t) {
    fill_n(dis + 1, nc, INT_MAX); queue<int> q;
    q.push(s); dis[s] = 0;
    while (!q.empty()) {
        int u = q.front(); q.pop();
        for (edge e : g[u]) {
            if (dis[e.v] != INT_MAX || e.c == e.f) continue;
            dis[e.v] = dis[u] + 1; q.push(e.v);
        }
    }
    return dis[t] != INT_MAX;
}

ll dfs(int u, int t, ll df) {
    if (u == t) return df; ll sf = 0;
    for (int& i = cur[u]; i != g[u].size(); ++i) {
        edge& e = g[u][i]; int v = e.v;
        if (dis[v] != dis[u] + 1 || e.c == e.f) continue;
        ll f = dfs(v, t, min(df, e.c - e.f));
        sf += f; e.f += f; g[v][e.p].f -= f; df -= f;
        if (!df) break;
    }
    return sf;
}

ll dinic(int s, int t, ll f = inf) {
    ll sf = 0;
    while(bfs(s, t)) {
        fill_n(cur + 1, nc, 0);
        ll df = dfs(s, t, f);
        sf += df; f -= df;
    }
    return sf;
}
```

ISAP

最大流

```
typedef long long ll;
const ll inf = LLONG_MAX;

struct edge { int p, v; ll c, f; };
vector<edge> g[N];
int cur[N], dis[N], gap[N], nc;

void fclr() { nc = 0; }

int addv(int cnt = 1) {
    while(cnt--) g[++nc].clear();
    return nc;
}

void adde(int u, int v, ll c) {
    g[u].push_back({ g[v].size(), v, c, 0 });
    g[v].push_back({ g[u].size() - 1, u, c, c });
}

ll dfs_isap(int s, int t, int u, ll f) {
    if (u == t) return f;
    ll sf = 0;
    for (int& i = cur[u]; i != g[u].size(); ++i) {
        edge& e = g[u][i]; int v = e.v;
        if (e.c == e.f || dis[u] != dis[v] + 1) continue;
        ll df = dfs_isap(s, t, v, min(f, e.c - e.f));
        e.f += df; sf += df; g[v][e.p].f -= df; f -= df;
        if (!f) return sf;
    }
    if (!--gap[dis[u]]) dis[s] = nc + 1;
    gap[++dis[u]]++; cur[u] = 0;
    return sf;
}

ll isap(int s, int t, ll f = inf) {
    ll sf = 0; queue<int> q;
    fill_n(dis + 1, nc, -1);
    fill_n(cur + 1, nc, 0);
    fill_n(gap + 1, nc, 0);
    gap[0] = 1; dis[t] = 0; q.push(t);
    while(!q.empty()) {
        int u = q.front(); q.pop();
        for (edge e : g[u]) {
            int v = e.v;
            if (g[v][e.p].c == g[v][e.p].f || dis[v] != -1) continue;
            dis[v] = dis[u] + 1; gap[dis[v]]++; q.push(v);
        }
    }
    while(dis[s] < nc) sf += dfs_isap(s, t, s, f);
    return sf;
}
```

MCMF-SPFA

最小费用最大流

```
typedef long long ll;

const ll inf = LLONG_MAX;
struct edge { int v, p; ll c, f, w; };
vector<edge> g[N];
int cur[N], nc;

int addv(int cnt) {
    while (cnt--) {
        int p = ++nc;
        g[p].resize(0);
    }
    return nc;
}

void adde(int u, int v, ll c, ll w) {
    g[u].push_back({ v, g[v].size(), c, 0, w });
    g[v].push_back({ u, g[u].size() - 1, c, c, -w });
}


bool inq[N]; ll dis[N];
bool spfa_mcmf(int s, int t) {
    fill_n(dis + 1, nc, LLONG_MAX);
    queue<int> q; q.push(s); inq[s] = 1; dis[s] = 0;
    while (!q.empty()) {
        int u = q.front(); q.pop(); inq[u] = 0;
        for (edge e : g[u]) {
            int v = e.v; ll dv = e.w + dis[u];
            if (dis[v] <= dv || e.c == e.f) continue;
            dis[v] = dv; if (!inq[v]) q.push(v), inq[v] = 1;
        }
    }
    return dis[t] != LLONG_MAX;
}

ll dfs_mcmf(int u, int t, ll f) {
    if (u == t) return f; ll sf = 0; inq[u] = 1;
    for (int& i = cur[u]; i != g[u].size(); ++i) {
        edge& e = g[u][i]; int v = e.v;
        if (e.c == e.f || dis[u] + e.w != dis[v] || inq[v]) continue;
        ll df = dfs_mcmf(v, t, min(f, e.c - e.f));
        e.f += df; sf += df; f -= df; g[v][e.p].f -= df;
        if (!f) break;
    }
    inq[u] = 0; return sf;
}

pair<ll, ll> mcmf(int s, int t, ll mf = inf) {
    ll sf = 0, sc = 0;
    while (spfa_mcmf(s, t)) {
        fill_n(cur + 1, nc, 0);
        ll df = dfs_mcmf(s, t, mf);
```

```
        mf -= df; sf += df;
        sc += dis[t] * df;
    }
    return { sf, sc };
}
```

MCMF-Dijkstra

最小费用最大流（原始对偶）

```
typedef long long ll;

const ll inf = LLONG_MAX;
struct edge { int v, p; ll c, f, w; };
vector<edge> g[N];
int cur[N], nc;

int addv(int cnt) {
    while (cnt--) {
        int p = ++nc;
        g[p].resize(0);
    }
    return nc;
}

void adde(int u, int v, ll c, ll w) {
    g[u].push_back({ v, g[v].size(), c, 0, w });
    g[v].push_back({ u, g[u].size() - 1, c, c, -w });
}

bool inq[N]; ll dis[N], pot[N];
bool dijkstra(int s, int t) {
    fill_n(dis + 1, nc, LLONG_MAX);
    typedef pair<ll, int> pli;
    priority_queue<pli, vector<pli>, greater<pli>> pq;
    pq.push({ dis[s] = 0, s });
    while (!pq.empty()) {
        pli p = pq.top(); pq.pop();
        ll du = p.first; int u = p.second;
        if (dis[u] < du) continue;
        for (edge e : g[u]) {
            int v = e.v; if (e.c == e.f) continue;
            ll dv = du + e.w + pot[u] - pot[v];
            if (dis[v] > dv) pq.push({ dis[v] = dv, v });
        }
    }
    return dis[t] != inf;
}

ll dfs_mcmf(int u, int t, ll f) {
    if (u == t) return f; ll sf = 0; inq[u] = 1;
    for (int& i = cur[u]; i != g[u].size(); ++i) {
        edge& e = g[u][i]; int v = e.v;
        if (e.c==e.f||dis[u]+e.w+pot[u]-pot[v]!=dis[v]||inq[v])continue;
        ll df = dfs_mcmf(v, t, min(f, e.c - e.f));
        e.f += df; sf += df; f -= df; g[v][e.p].f -= df;
        if (!f) break;
```

```
    }
    inq[u] = 0; return sf;
}

pair<ll, ll> mcmf(int s, int t, ll mf = inf) {
    ll sf = 0, sc = 0;
    fill_n(pot + 1, nc, 0);
    while (dijkstra(s, t)) {
        fill_n(cur + 1, nc, 0);
        ll df = dfs_mcmf(s, t, mf);
        mf -= df; sf += df;
        sc += (dis[t] + pot[t]) * df;
        for (int i = 1; i <= nc; ++i) pot[i] += dis[i];
    }
    return { sf, sc };
}
```

Edmonds(Blossom)

其他

HLPP

```
typedef long long ll;

const ll inf = LLONG_MAX;
struct edge { int v, p; ll c, f; };
vector<edge> g[N];
int h[N], gap[N << 1], nc; ll ef[N]; bool inq[N];
typedef priority_queue<int, vector<int>, function<bool(int,int)>> pq;

int addv(int cnt) {
    while (cnt--) {
        int p = ++nc; g[p].resize(0);
        ef[p] = h[p] = gap[p] = inq[p] = 0;
    }
    return nc;
}

void adde(int u, int v, ll c) {
    g[u].push_back({ v, g[v].size(), c, 0 });
    g[v].push_back({ u, g[u].size() - 1, c, c });
}

bool bfs(int s, int t) {
    fill_n(h + 1, nc, INT_MAX);
    queue<int> q; q.push(t); h[t] = 0;
    while (!q.empty()) {
        int u = q.front(); q.pop();
        for (edge e : g[u])
            if (e.f && h[e.v] == INT_MAX)
                h[e.v] = h[u] + 1, q.push(e.v);
    }
    return h[s] != INT_MAX;
}
```

```
void push(pq& q, int u, bool flg = 0) {
    for (edge& e : g[u]) {
        if (e.c == e.f || h[e.v] == INT_MAX || (!flg && h[e.v] + 1 != h[u]))
continue;
        ll df = min(ef[u], e.c - e.f);
        e.f += df; g[e.v][e.p].f -= df;
        ef[u] -= df; ef[e.v] += df;
        if (!inq[e.v]) q.push(e.v), inq[e.v] = 1;
        if (!ef[u]) break;
    }
}

ll hlpp(int s, int t, ll mf = inf) {
    if (!bfs(s, t)) return 0; h[s] = nc; inq[s] = inq[t] = 1;
    pq q([&](int i1, int i2) { return h[i1] < h[i2]; });
    ef[s] = mf; for (int i = 1; i <= nc; ++i)
        if (h[i] != INT_MAX) gap[h[i]]++;
    push(q, s, 1); while (!q.empty()) {
        int u = q.top(); q.pop(); inq[u] = 0;
        push(q, u); if (!ef[u]) continue;
        if (!--gap[h[u]]) for (int i = 1; i <= nc; ++i)
            if (i!=s&&i!=t&&h[u]<h[i])h[i]=max(h[i],nc+1);
        h[u] = 2 * nc;
        for (edge e : g[u]) if (e.c != e.f && h[e.v] != INT_MAX)
            h[u] = min(h[u], h[e.v] + 1);
        ++gap[h[u]]; q.push(u); inq[u] = 1;
    }
    return mf - ef[s];
}
```

带上下界的网络流

带上下界的最大流

```
ll df[N];
void adde_c(int u, int v, ll cl, ll cr) {
    adde(u, v, cr - cl);
    df[u] -= cl;
    df[v] += cl;
}

bool feasible_flow() {
    int s = addv(1), t = addv(1);
    ll se = 0;
    for (int i = 1; i <= nc - 2; ++i) {
        if (!df[i]) continue;
        if (df[i] > 0) adde(s, i, df[i]), se += df[i];
        else adde(i, t, -df[i]);
    }
    se -= isap(s, t, se); nc -= 2;
    for (int i = 1; i <= nc; ++i)
        while (g[i].back().v > nc) g[i].pop_back();
    return !se;
}

ll max_flow(int s, int t) {
    adde(t, s, inf);
```

```
    if (!feasible_flow()) return -1;
    return isap(s, t);
}

ll min_flow(int s, int t) {
    adde(t, s, inf);
    int i1 = g[t].size() - 1, i2 = g[s].size() - 1;
    if (!feasible_flow()) return -1;
    ll sf = g[t][i1].f;
    g[t][i1].c = g[t][i1].f = 0;
    return sf - isap(t, s);
}
```