二分图问题和缩点缩边问题优先选择如下模板

二分图最大匹配_匈牙利算法

```cpp
#include <algorithm>
#include <iostream>
#include <queue>
#include <tuple>
#include <vector>
using namespace std;

// sqrt(n)*m/10  ~  nlog(n)
// 最小点覆盖问题是指，在一张无向图中选择最少的顶点，满足每条边至少有一个端点被选。
// 二分图中，最小点覆盖中的顶点数量等于最大匹配中的边数量
// 最小点覆盖与最大独立集的大小之和等于顶点数目。
struct BipartiteGraph
{
    int n1, n2;                       // number of vertices in X and Y, resp.
    std::vector<std::vector<int>> g; // edges from X to Y
    std::vector<int> ma, mb;          // matches from X to Y and from Y to X, resp.
匹配从X到Y和从Y到X
    std::vector<int> dist;            // distance from unsaturated vertices in X.
到X中不饱和顶点的距离

    BipartiteGraph(int n1, int n2)
        : n1(n1), n2(n2), g(n1), ma(n1, -1), mb(n2, -1) {}
    // 添加一条从X中的u 到Y中的v的边
    // Add an edge from u in X to v in Y.
    void add_edge(int u, int v) { g[u].emplace_back(v); }

    // Build the level graph.
    // 构建层级图
    bool bfs()
    {
        dist.assign(n1, -1);
        std::queue<int> q;
        for (int u = 0; u < n1; ++u)
        {
            if (ma[u] == -1)
            {
                dist[u] = 0;
                q.emplace(u);
            }
        }
        // 为所有可到达的顶点构建层级图。
        // Build the level graph for all reachable vertices.
        bool succ = false;
        while (!q.empty())
        {
```

```
            int u = q.front();
            q.pop();
            for (int v : g[u])
            {
                if (mb[v] == -1)
                {
                    succ = true;
                }
                else if (dist[mb[v]] == -1)
                {
                    dist[mb[v]] = dist[u] + 1;
                    q.emplace(mb[v]);
                }
            }
        }
        return succ;
    }
    // 找到一个从u开始的增广路径
    // Find an augmenting path starting at u.
    bool dfs(int u)
    {
        for (int v : g[u])
        {
            if (mb[v] == -1 || (dist[mb[v]] == dist[u] + 1 && dfs(mb[v])))
            {
                ma[u] = v;
                mb[v] = u;
                return true;
            }
        }
        // 将此点标记为访问一次后不可达
        dist[u] = -1; // Mark this point as inreachable after one visit.
        return false;
    }

    // Hopcroft-Karp maximum matching algorithm.
    std::vector<std::pair<int, int>> hopcroft_karp_maximum_matching()
    {
        // Build the level graph and then find a blocking flow.
        // 构建层级图，然后找出一条阻塞流
        while (bfs())
        {
            for (int u = 0; u < n1; ++u)
            {
                if (ma[u] == -1)
                {
                    dfs(u);
                }
            }
        }
        // 收集成功的配对
        std::vector<std::pair<int, int>> matches;
        matches.reserve(n1);
        for (int u = 0; u < n1; ++u)
```

```
        {
            if (ma[u] != -1)
            {
                matches.emplace_back(u, ma[u]);
            }
        }
        return matches;
    }
};

/*
input:
4 4 7
0 0
1 0
1 1
2 1
2 2
3 2
3 3

左边的点: 0, 1, 2, 3
右边的点: 0, 1, 2, 3

output:
4
0 0
1 1
2 2
3 3

*/


int main()
{
    std::ios::sync_with_stdio(false), std::cin.tie(nullptr);
    int n1, n2, m;
    std::cin >> n1 >> n2 >> m;
    BipartiteGraph gr(n1, n2);
    for (int i = 0; i < m; ++i)
    {
        int u, v;
        std::cin >> u >> v;
        gr.add_edge(u, v);
    }
    auto res = gr.hopcroft_karp_maximum_matching();
    std::cout << res.size() << '\n';
    for (int i = 0; i < res.size(); ++i)
    {
        std::cout << res[i].first << ' ' << res[i].second << '\n';
    }
```

```
    return 0;
}
```

二分图最大权匹配_模板二

```
#include <bits/stdc++.h>
using namespace std;


// O(n^3)  n = 400时，大概为 1200ms
// 一定要注意值是否会超出 int 范围

// #define int long long
struct MaxCostMatch
{
    vector<int> ansl, ansr, pre;
    vector<int> lx, ly;
    vector<vector<int>> ver;
    int n;
    MaxCostMatch(int n) : n(n)
    {
        ver.resize(n + 1, vector<int>(n + 1));
        ansl.resize(n + 1, -1);
        ansr.resize(n + 1, -1);
        lx.resize(n + 1);
        ly.resize(n + 1, -1E18);
        pre.resize(n + 1);
    }
    void add(int x, int y, int w)
    {
        ver[x][y] = w;
    }
    void bfs(int x)
    {
        vector<bool> visl(n + 1), visr(n + 1);
        vector<int> slack(n + 1, 1E18);
        queue<int> q;
        function<bool(int)> check = [&](int x)
        {
            visr[x] = 1;
            if (~ansr[x])
            {
                q.push(ansr[x]);
                visl[ansr[x]] = 1;
                return false;
            }
            while (~x)
            {
                ansr[x] = pre[x];
                swap(x, ansl[pre[x]]);
```

```
        }
        return true;
    };
    q.push(x);
    visl[x] = 1;
    while (1)
    {
        while (!q.empty())
        {
            int x = q.front();
            q.pop();
            for (int y = 1; y <= n; ++y)
            {
                if (visr[y])
                    continue;
                int del = lx[x] + ly[y] - ver[x][y];
                if (del < slack[y])
                {
                    pre[y] = x;
                    slack[y] = del;
                    if (!slack[y] && check(y))
                        return;
                }
            }
        }
        int val = 1E18;
        for (int i = 1; i <= n; ++i)
        {
            if (!visr[i])
            {
                val = min(val, slack[i]);
            }
        }
        for (int i = 1; i <= n; ++i)
        {
            if (visl[i])
                lx[i] -= val;
            if (visr[i])
            {
                ly[i] += val;
            }
            else
            {
                slack[i] -= val;
            }
        }
        for (int i = 1; i <= n; ++i)
        {
            if (!visr[i] && !slack[i] && check(i))
            {
                return;
            }
        }
    }
```

```
    }
    int work()
    {
        for (int i = 1; i <= n; ++i)
        {
            for (int j = 1; j <= n; ++j)
            {
                ly[i] = max(ly[i], ver[j][i]);
            }
        }
        for (int i = 1; i <= n; ++i)
            bfs(i);
        int res = 0;
        for (int i = 1; i <= n; ++i)
        {
            res += ver[i][ansl[i]];
        }
        return res;
    }
    void getMatch(int x, int y)
    { // 获取方案 (0代表无匹配)
        // 输出左端点分别匹配的右端点编号
        for (int i = 1; i <= x; ++i)
        {
            cout << (ver[i][ansl[i]] ? ansl[i] : 0) << " ";
        }
        cout << endl;

        // 输出右端点匹配的相应的左端点的编号
        for (int i = 1; i <= y; ++i)
        {
            cout << (ver[i][ansr[i]] ? ansr[i] : 0) << " ";
        }
        cout << endl;
    }
};



signed main()
{
    int n1, n2, m;
    cin >> n1 >> n2 >> m;
    MaxCostMatch match(max(n1, n2));
    for (int i = 1; i <= m; i++)
    {
        int x, y, w;
        cin >> x >> y >> w;
        match.add(x, y, w);
    }
    cout << match.work() << '\n';
    match.getMatch(n1, n2);
```

```
}
```

二分图最大权匹配

```
#include <bits/stdc++.h>
using namespace std;


// O(n^3)  n = 400时，大概为 1500ms
template <typename T>
struct hungarian
{ // km
    int n;
    vector<int> matchx; // 左集合对应的匹配点
    vector<int> matchy; // 右集合对应的匹配点
    vector<int> pre;    // 连接右集合的左点
    vector<bool> visx;  // 拜访数组 左
    vector<bool> visy;  // 拜访数组 右
    vector<T> lx;
    vector<T> ly;
    vector<vector<T>> g;
    vector<T> slack;
    T inf;
    T res;
    queue<int> q;
    int org_n;
    int org_m;

    hungarian(int _n, int _m)
    {
        org_n = _n;
        org_m = _m;
        n = max(_n, _m);
        inf = numeric_limits<T>::max();
        res = 0;
        g = vector<vector<T>>(n, vector<T>(n));
        matchx = vector<int>(n, -1);
        matchy = vector<int>(n, -1);
        pre = vector<int>(n);
        visx = vector<bool>(n);
        visy = vector<bool>(n);
        lx = vector<T>(n, -inf);
        ly = vector<T>(n);
        slack = vector<T>(n);
    }

    void addEdge(int u, int v, int w)
    {
        g[u][v] = max(w, 0); // 负值还不如不匹配 因此设为0不影响
```

```
}

bool check(int v)
{
    visy[v] = true;
    if (matchy[v] != -1)
    {
        q.push(matchy[v]);
        visx[matchy[v]] = true; // in S
        return false;
    }
    // 找到新的未匹配点 更新匹配点 pre 数组记录着"非匹配边"上与之相连的点
    while (v != -1)
    {
        matchy[v] = pre[v];
        swap(v, matchx[pre[v]]);
    }
    return true;
}

void bfs(int i)
{
    while (!q.empty())
    {
        q.pop();
    }
    q.push(i);
    visx[i] = true;
    while (true)
    {
        while (!q.empty())
        {
            int u = q.front();
            q.pop();
            for (int v = 0; v < n; v++)
            {
                if (!visy[v])
                {
                    T delta = lx[u] + ly[v] - g[u][v];
                    if (slack[v] >= delta)
                    {
                        pre[v] = u;
                        if (delta)
                        {
                            slack[v] = delta;
                        }
                        else if (check(v))
                        { // delta=0 代表有机会加入相等子图 找增广路
                          // 找到就return 重建交错树
                            return;
                        }
                    }
                }
            }
```

```
            }
            // 没有增广路 修改顶标
            T a = inf;
            for (int j = 0; j < n; j++)
            {
                if (!visy[j])
                {
                    a = min(a, slack[j]);
                }
            }
            for (int j = 0; j < n; j++)
            {
                if (visx[j])
                { // S
                    lx[j] -= a;
                }
                if (visy[j])
                { // T
                    ly[j] += a;
                }
                else
                { // T'
                    slack[j] -= a;
                }
            }
            for (int j = 0; j < n; j++)
            {
                if (!visy[j] && slack[j] == 0 && check(j))
                {
                    return;
                }
            }
        }
    }

    void solve()
    {
        // 初始顶标
        for (int i = 0; i < n; i++)
        {
            for (int j = 0; j < n; j++)
            {
                lx[i] = max(lx[i], g[i][j]);
            }
        }

        for (int i = 0; i < n; i++)
        {
            fill(slack.begin(), slack.end(), inf);
            fill(visx.begin(), visx.end(), false);
            fill(visy.begin(), visy.end(), false);
            bfs(i);
        }
```

```
        // custom
        for (int i = 0; i < n; i++)
        {
            if (g[i][matchx[i]] > 0)
            {
                res += g[i][matchx[i]];
            }
            else
            {
                matchx[i] = -1;
            }
        }
        cout << res << "\n";

        // 左集合对应的匹配点
        // 如要右集合则改为输出  matchy
        for (int i = 0; i < org_n; i++)
        {
            cout << matchx[i] + 1 << " ";
        }
        cout << "\n";
    }
};


int main()
{
    ios::sync_with_stdio(false), cin.tie(nullptr);
    int n, m, e;
    cin >> n >> m >> e;

    hungarian<long long> solver(n, m);

    int u, v, w;
    for (int i = 0; i < e; i++)
    {
        cin >> u >> v >> w;
        u--, v--;
        solver.addEdge(u, v, w);
    }
    solver.solve();
}
```

SPFA (负权图)

```
#include <bits/stdc++.h>
using namespace std;
const int INF = 0x3f3f3f3f;
const int N = 1e6 + 5, M = 2e6 + 5;
// 单源最短路（负权边，负环），最坏 O(nm)
```

```
int n, m;
int pre[N];
void print_path(int s, int t)
{
    if (s == t)
    {
        cout << s << " ";
        return;
    }
    print_path(s, pre[t]);
    cout << t << " " ;
}
int head[N], cnt;
struct
{
    int to, next, w;
}edge[M];
void init()
{
    for (int i = 0; i < N; i++) head[i] = -1;
    for (int i = 0; i < M; i++) edge[i].next = -1;
    cnt = 0;
}
void addedge(int u, int v, int w)
{
    edge[cnt].to = v;
    edge[cnt].w = w;
    edge[cnt].next = head[u];
    head[u] = cnt++;
}
int dis[N];
bool inq[N];
int Neg[N];
int spfa(int s)
{
    memset(Neg, 0, sizeof(Neg));
    Neg[s] = 1;
    for (int i = 1; i < n; i++)
    {
        dis[i] = INF;
        inq[i] = false;
    }
    dis[s] = 0;
    queue<int> Q;
    Q.push(s);
    inq[s] = true;
    while (!Q.empty())
    {
        int u = Q.front();
        Q.pop();
        inq[u] = false;
        for (int i =  head[u]; ~i; i = edge[i].next)
        {
            int v = edge[i].to, w = edge[i].w;
```

```
            if (dis[u] + w < dis[v])
            {
                dis[v] = dis[u] + w;
                pre[v] = u;
                if (!inq[v])
                {
                    inq[v] = true;
                    Q.push(v);
                    Neg[v]++;
                    if (Neg[v] > n) return 1; // 在不经过负环的情况下，最短路最多有 n
- 1 条边
                }
            }
        }
    }
    return 0;
}
signed main()
{
    ios::sync_with_stdio(false);
    cin.tie(0), cout.tie(0);
    while (cin >> n >> m)
    {
        init();
        if (n == 0 && m == 0) return 0;
        while (m--)
        {
            int u, v, w;
            cin >> u >> v >> w;
            addedge(u, v, w);
            addedge(v, u, w);
        }
        spfa(1);
        cout << dis[n] << endl;
    }
    return 0;
}
```

```
#include <bits/stdc++.h>
using namespace std;
// 割边缩点
struct EDCC{
    int n, now, cnt;
    vector<vector<int>> ver;
    vector<int> dfn, low, col, S;
    set<array<int, 2>> bridge;
    EDCC(int n) : n(n), ver(n + 1), low(n + 1){
        dfn.resize(n + 1, -1);
        col.resize(n + 1, -1);
        now = cnt = 0;
    }
    void add(int x, int y){
```

```
        ver[x].push_back(y);
        ver[y].push_back(x);
    }
    void tarjan(int x, int fa)
    {
        dfn[x] = low[x] = now++;
        S.push_back(x);
        for (auto y : ver[x]){
            if (y == fa) continue;
            if (dfn[y] == -1){
                bridge.insert({x, y}); // 储存割边
                tarjan(y, x);
                low[x] = min(low[x], low[y]);
            }
            else if (col[y] == -1 && dfn[y] < dfn[x]){
                bridge.insert({x, y});
                low[x] = min(low[x], dfn[y]);
            }
        }
        if (dfn[x] == low[x]){
            int pre;
            cnt++;
            do{
                pre = S.back();
                col[pre] = cnt;
                S.pop_back();
            }while (pre != x);
        }
    }
    auto work(){ // [cnt 新图的顶点数量， bridge 全部割边]
        for (int i = 1; i <= n; i++) { // 避免图不连通
            if (dfn[i] == -1) {
                tarjan(i, 0);
            }
        }
        vector<int> siz(cnt + 1); // siz 每个边双中点的数量
        vector<vector<int>> adj(cnt + 1); // adj 新图
        for (int i = 1; i <= n; i++) {
            siz[col[i]]++;
            for (auto j : ver[i]) {
                int x = col[i], y = col[j];
                if (x != y) {
                    adj[x].push_back(y);
                }
            }
        }
        return tuple{cnt, adj, col, siz};
    }
};
```

```
#include <bits/stdc++.h>
```

```
using namespace std;
// 有向图缩点
struct SCC
{
    int n, now, cnt;
    vector<vector<int>> ver;
    vector<int> dfn, low, col, S;

    SCC(int n) : n(n), ver(n + 1), low(n + 1)
    {
        dfn.resize(n + 1, -1);
        col.resize(n + 1, -1);
        now = cnt = 0;
    }

    void add(int x, int y)
    {
        ver[x].push_back(y);
    }
    void tarjan(int x)
    {
        dfn[x] = low[x] = now++;
        S.push_back(x);
        for (auto y : ver[x])
        {
            if (dfn[y] == -1)
            {
                tarjan(y);
                low[x] = min(low[x], low[y]);
            }
            else if (col[y] == -1)
            {
                low[x] = min(low[x], dfn[y]);
            }
        }
        if (dfn[x] == low[x])
        {
            int pre;
            cnt++;
            do{
                pre = S.back();
                col[pre] = cnt;
                S.pop_back();
            }while (pre != x);
        }
    }
    tuple<int, vector<vector<int>>, vector<int>, vector<int>> work()
    {
        for (int i = 1; i <= n; i++)
        {
            if (dfn[i] == -1)
            {
                tarjan(i);
            }
```

```
        }

        vector<int> siz(cnt + 1);
        vector<vector<int>> adj(cnt + 1);
        for (int i = 1 ; i <= n; i++)
        {
            siz[col[i]]++;
            for (auto j : ver[i])
            {
                int x = col[i], y = col[j];
                if (x != y)
                {
                    adj[x].push_back(y);
                }
            }
        }
        // 新图的顶点数， 新图的边， 节点属于哪个个SCC， SCC的大小
        // C++17及以上可用 : auto {} = scc.work();
        // C++11 / 14 可用：result = scc.work(); vector<int> cnt = get<0>(result);
vector<vector<int>> adj = get<1>(result); vector<int> col = get<2>(result);
vector<int> siz = get<3>(result);
        return {cnt, adj, col, siz};
    }
};
```

```
#include <bits/stdc++.h>
using namespace std;
// 割点缩边
struct V_DCC
{
    int n;
    vector<vector<int>> ver, col;
    vector<int> dfn, low, S;
    int now, cnt;
    vector<bool> point; // 记录是否为割点
    V_DCC(int n) : n(n)
    {
        ver.resize(n + 1);
        dfn.resize(n + 1);
        low.resize(n + 1);
        col.resize(2 * n + 1);
        point.resize(n + 1);
        S.clear();
        cnt = now = 0;
    }
    void add(int x, int y)
    {
        if (x == y)
            return; // 手动去除重边
        ver[x].push_back(y);
        ver[y].push_back(x);
    }
```

```
    void tarjan(int x, int root)
    {
        low[x] = dfn[x] = now++;
        S.push_back(x);
        if (x == root && !ver[x].size())
        { // 特判孤立点
            ++cnt;
            col[cnt].push_back(x);
            return;
        }
        int flag = 0;
        for (auto y : ver[x])
        {
            if (!dfn[y])
            {
                tarjan(y, root);
                low[x] = min(low[x], low[y]);
                if (dfn[x] <= low[y])
                {
                    flag++;
                    if (x != root || flag > 1)
                    {
                        point[x] = true; // 标记为割点
                    }
                    int pre = 0;
                    cnt++;
                    do
                    {
                        pre = S.back();
                        col[cnt].push_back(pre);
                        S.pop_back();
                    } while (pre != y);
                    col[cnt].push_back(x);
                }
            }
            else
            {
                low[x] = min(low[x], dfn[y]);
            }
        }
    }
    pair<int, vector<vector<int>>> rebuild()
    { // [新图的顶点数量, 新图]
        work();
        vector<vector<int>> adj(cnt + 1);
        for (int i = 1; i <= cnt; i++)
        {
            if (!col[i].size())
            { // 注意，孤立点也是 V-DCC
                continue;
            }
            for (auto j : col[i])
            {
                if (point[j])
```

```
                { // 如果 j 是割点
                    adj[i].push_back(point[j]);
                    adj[point[j]].push_back(i);
                }
            }
        }
        return {cnt, adj};
    }
    void work()
    {
        for (int i = 1; i <= n; ++i)
        { // 避免图不连通
            if (!dfn[i])
            {
                tarjan(i, i);
            }
        }
    }
};
```

```
template <class T>
struct Fenwick
{
    int n;
    vector<T> t;
    Fenwick(T n) : n(n) { t.assign(n + 1, T{}); }
    void add(int x, const T &v)
    {
        for (int i = x; i <= n; i += i & -i)
        {
            t[i] += v;
        }
    }
    T sum(int x)
    {
        assert(x >= 0);
        int res = 0;
        for (int i = x; i; i -= i & -i)
        {
            res += t[i];
        }
        return res;
    }
    T range(int l, int r)
    {
        return sum(r) - sum(l - 1);
    }
    int select(const T &k)
    {
        int x = 0;
        T cur{};
        for (int i = 1 << __lg(n); i; i /= 2)
```

```
        {
            if (x + i <= n && cur + t[x + i] <= k)
            {
                x += i;
                cur += t[x];
            }
        }
        return x;
    }
};
```