网络流模板

——by ronaflx

目录

网络流 sap 模板

```
#include <iostream>
#include <cstring>
#include <cstdio>
using namespace std;

class sap
{
private:
    const static int V = 6000;
    const static int E = 1000001;
    const static int INF = 1000000000;
    int dis[V], numdis[V];

    struct edge
    {
        int v;
        int cap;
        edge *nxt;
    } pool[E], *g[V], *pp;

    void bfs(int i, int n)
    {
        int que[V], tail = 0;
        bool vst[V];
        memset(vst, 0, sizeof (vst));
        memset(numdis, 0, sizeof (numdis));
        for (int j = 0; j < n; j++)
            dis[j] = n;
        dis[i] = 0;
        numdis[0]++;
        vst[i] = 1;
        que[0] = i;
        for (int j = 0; j <= tail; j++)
            for (edge *ip = g[que[j % n]]; ip != NULL; ip = ip->nxt)
                if (pool[(ip - pool)^1].cap > 0 && !vst[ip->v])
                {
                    tail++;
                    vst[ip->v] = 1;
                    que[tail % n] = ip->v;
                    dis[ip->v] = dis[que[j % n]] + 1;
                    numdis[dis[ip->v]]++;
```

```
                }
    }

public:

    void firststart()
    {
        pp = pool;
        memset(g, 0, sizeof (g));
    }

    void addedge(int i, int j, int cap)
    {
        pp->v = j;
        pp->cap = cap;
        pp->nxt = g[i];
        g[i] = pp++;
    }

    int maxflowsap(int n, int s, int t)
    {
        bfs(t, n);
        int nowv = s, skip, pre[V];
        int ans = 0;
        edge * nowe[V], *pree[V];
        for (int j = 0; j < n; j++)
            nowe[j] = g[j];
        while (dis[s] < n)
        {
            skip = 0;
            while (nowe[nowv])
            {
                if (nowe[nowv]->cap > 0 && dis[nowv]==dis[nowe[nowv]->v]+1)
                {
                    pre[nowe[nowv]->v] = nowv;
                    pree[nowe[nowv]->v] = nowe[nowv];
                    nowv = nowe[nowv]->v;
                    if (nowv == t)
                    {
                        int minf = INF;
                        for (int tmpv = t; tmpv != s; tmpv = pre[tmpv])
                            minf = min(minf, pree[tmpv]->cap);
                        ans += minf;
                        for (int tmpv = t; tmpv != s; tmpv = pre[tmpv])
```

```
                        {
                            pree[tmpv]->cap -= minf;
                            pool[(pree[tmpv] - pool)^0x1].cap += minf;
                        }
                        nowv = s;
                        skip = 1;
                        break;
                    }
                    continue;
                }
                nowe[nowv] = nowe[nowv]->nxt;
            }
            if (!skip)
            {
                int mindis = INF;
                numdis[dis[nowv]]--;
                if (!numdis[dis[nowv]])
                    break;
                for (edge *ip = g[nowv]; ip != NULL; ip = ip->nxt)
                    if (ip->cap > 0)
                        mindis = min(mindis, dis[ip->v] + 1);
                if (mindis == INF)
                    dis[nowv] = n;
                else
                    dis[nowv] = mindis;
                numdis[dis[nowv]]++;
                nowe[nowv] = g[nowv];
                if (nowv != s)
                    nowv = pre[nowv];
            }
        }
        return ans;
    }
};
sap maxflow;
```

费用流模板

```
#include <iostream>
#include <cstring>
#include <cstdio>
#include <algorithm>
using namespace std;
```

```
class mincost
{
private:
    static const int V = 1000;
    static const int E = 1000001;
    static const int INF = 0x7ffffff;

    struct Edge
    {
        int v, cap, cost;
        Edge *next;
    } pool[E], *g[V], *pp, *pree[V];
    int T, S, dis[V], pre[V];
    int n, m;

    void SPFA()
    {
        bool vst[V] = {false};
        int cirq[V];
        int tail = 0, u;
        fill(dis, dis + T + 1, 0x7ffffff);
        cirq[0] = S;
        vst[S] = 1;
        dis[S] = 0;
        for (int j = 0; j <= tail; j++)
        {
            int v = cirq[j % n];
            for (Edge *i = g[v]; i != NULL; i = i->next)
            {
                if (!i->cap)
                    continue;
                u = i->v;
                if (i->cost + dis[v] < dis[u])
                {
                    dis[u] = i->cost + dis[v];
                    pree[u] = i;
                    pre[u] = v;
                    if (!vst[u])
                    {
                        tail++;
                        cirq[tail % n] = u;
                        vst[u] = true;
                    }
```

```
                }
            }
            vst[v] = false;
        }
    }

public:

    inline void addedge(int i, int j, int cap, int cost)
    {
        pp->cap = cap;
        pp->v = j;
        pp->cost = cost;
        pp->next = g[i];
        g[i] = pp++;
    }

    void initialize()
    {
        memset(g, 0, sizeof (g));
        pp = pool;
    }

    int mincost_maxflow(int n, int s, int t)
    {
        S = s;
        T = t;
        this->n = n;
        int flow = 0;
        while (true)
        {
            SPFA();
            if (dis[T] == INF)
                break;
            int minn = INF;
            for (int i = T; i != S; i = pre[i])
                minn = min(minn, pree[i]->cap);
            for (int i = T; i != S; i = pre[i])
            {
                pree[i]->cap -= minn;
                pool[(pree[i] - pool)^0x1].cap += minn;
                flow += minn * pree[i]->cost;
            }
        }
```

```
        return flow;
    }

};
```

网络流之上下界可行流

```
/*无源无汇上下界可行流:
设M(i)为流入结点 i 的下界总和减去流出 i 的下界总和。
设一附加源 S，汇 T，则可以令 C'(S, i) = M(i)。
如果 M(i)是负数，那么有 C'(i, T0) = -M(i)。
对于其他的边有 C'(u, v) = C(u, v) - B(u, v)*/
for(int i = 0;i < m;i++)
{
    scanf("%d %d %d %d",&vfrom,&vto,&low[i],&tmpcap);
    maxflow.addedge(vfrom,vto,tmpcap - low[i]);
    maxflow.addedge(vto,vfrom,0);
    b[vto] += low[i];
    b[vfrom] -= low[i];
}
for (int i = 1; i <= n; i++)
{
    if (b[i] > 0)
    {
        maxflow.addedge(s, i, b[i]);
        maxflow.addedge(i, s, 0);
        tmpans += b[i];
    }
    else
    {
        maxflow.addedge(i, t, -b[i]);
        maxflow.addedge(t, i, 0);
    }
}
/*有源有汇上下界可行流的解法:
如果从 s 到 t 有一个可行流 a，那么我们从 t 到 s 连一条，流量下界为 a 的边，上界 INF
那么就可以用无源无汇的模型了,二分 a 即可*/
```

网络流之关键割

```
/*关键割:
如果增加的该边的流量，那么网络流的流量会增加
注意一个初学者常犯的错误:
满流的边不一定就是关键割，所以 floodfill or dfs 的时候要遍历正向和反向边

算法:
开始给所有的点标记为 0
从源点做一次 floodfill,所有遇到的点标记为 1，包括正向边和负向边
再从汇点做一次 floodfill,所有遇到的点标记为 2,包括正向边和负向边
遍历正向边，如果他的一个端点为 1，另一个为 2，那么他就是在关键割中
代码：每个边给了一个编号， cnt 返回关键割中的编号关键的边*/

bool floodfill(int beg, int *reach, int clr)
{
    queue<int> q;
    q.push(beg);
    reach[beg] = clr;
    while (!q.empty())
    {
        beg = q.front();
        q.pop();
        for (Edges *i = vfrom[beg]; i != NULL; i = i->next)
        {
            if (!reach[i->vto] && i->ecap)
            {
                reach[i->vto] = clr;
                q.push(i->vto);
            }
        }
    }
    return reach[roof];
}
maxflow.maxflowsap(n, s, t);
memset(reach, 0, sizeof (reach));
floodfill(s, reach, 1);
floodfill(t, reach, 2);
cnt = INF;
for (int i = 0; i < n; i++)
{
    for (Edges *j = vfrom[i]; j != NULL; j = j->next)
    {
```

```
        if (reach[i] == 1 && reach[j->vto] == 2 && ((j - pool) & 1) == 0)
        {
            j->ecap++;
            int vst[V];
            memset(vst, 0, sizeof (vst));
            if (floodfill(s, vst, 1))
                cnt = min(cnt, j->id);
            j->ecap--;
        }

    }
}
```

网络流之混合图欧拉迹

```
/*算法:
    把该图的无向边随便定向，计算每个点的入度和出度。如果有某个点出入度之差为奇数，那么肯定不存在欧拉回路。因为欧拉回路要求每点入度 = 出度，也就是总度数为偶数，存在奇数度点必不能有欧拉回路。现在每个点入度和出度之差均为偶数。将这个偶数除以 2，得 x。有向边不能改变方向，直接删掉。另新建 s 和 t。对于入 > 出的点 u，连接边(u, t)、容量为 x，对于出 > 入的点 v，连接边(s, v)，容量为 x 之后，察看是否有满流的分配。有就是能有欧拉回路，没有就是没有。
代码: */
const int N = 201, E = 2000;
int indgr[N], outdgr[N], vfrom[E], vto[E], d[E];

bool check(int n)
{
    for (int i = 1; i <= n; i++)
        if ((indgr[i] - outdgr[i]) & 1)
            return false;
    return true;
}

int main()
{
    int csc, s, t, n, m, allt, alls;
    scanf("%d", &csc);
    while (csc--)
    {
        scanf("%d %d", &n, &m);
        s = 0, t = n + 1, allt = alls = 0;
        memset(indgr, 0, sizeof (indgr));
        memset(outdgr, 0, sizeof (outdgr));
```

```
        for (int i = 1; i <= m; i++)
        {
            scanf("%d %d %d", &vfrom[i], &vto[i], &d[i]);
            outdgr[vfrom[i]]++;
            indgr[vto[i]]++;
        }
        if (!check(n))
            printf("impossible\n");
        else
        {
            maxflow.firststart();
            for (int i = 1; i <= n; i++)
            {
                if (indgr[i] > outdgr[i])
                {
                    maxflow.addedge(i, t, (indgr[i] - outdgr[i]) / 2);
                    maxflow.addedge(t, i, 0);
                    allt += indgr[i] - outdgr[i];
                }
                else if (indgr[i] < outdgr[i])
                {
                    maxflow.addedge(s, i, (outdgr[i] - indgr[i]) / 2);
                    maxflow.addedge(i, s, 0);
                    alls += outdgr[i] - indgr[i];
                }
            }
            for (int i = 1; i <= m; i++)
            {
                if (d[i])
                    continue;
                maxflow.addedge(vfrom[i], vto[i], 1);
                maxflow.addedge(vto[i], vfrom[i], 0);
            }
            allt /= 2;
            alls /= 2;
            if (allt != alls || allt != maxflow.maxflowsap(t + 1, s, t))
                printf("impossible\n");
            else
                printf("possible\n");
        }
    }
    return 0;
}
```

网络流之点连通度

```
/*点联通度：在图中去掉多少个点，使图边的不联通
算法：
    拆点：把一个点 v 拆成一个 v" 一个 v',然后枚举源点和汇点做 n^2 网络流
    从 v" 到 v'连容量为 1 原图上的边<u,v> 在新图上构建一个 <u',v">容量为 INF
    注意：一定要源点和汇点都枚举，不然如果点割集在任意选取的源点和汇点上会得到错误的结果
    取每次网络流的最小值，如果最后的流量为 INF 那么原图为完全图 ans = n;
    复杂度 O(n^5)网络流，敢写就有奇迹！！！！！
代码：*/
int ans = INF;
for (int j = 0; j < n; j++)
{
    for (int k = j + 1; k < n; k++)
    {
        maxflow.firststart();
        for (int i = 0; i < n; i++)
        {
            maxflow.addedge(i * 2, i * 2 + 1, 1);
            maxflow.addedge(i * 2 + 1, i * 2, 0);
        }
        for (int i = 0; i < m; i++)
        {
            maxflow.addedge(vfrom[i] * 2 + 1, vto[i] * 2, INF);
            maxflow.addedge(vto[i] * 2, vfrom[i] * 2 + 1, 0);
            maxflow.addedge(vto[i] * 2 + 1, vfrom[i] * 2, INF);
            maxflow.addedge(vfrom[i] * 2, vto[i] * 2 + 1, 0);
        }
        s = j * 2 + 1, t = k * 2;
        ans = min(ans, maxflow.maxflowsap(n * 2, s, t));
    }
}
if (ans == INF)
    printf("%d\n", n);
else
    printf("%d\n", ans);
```

网络流之二分图最小点权覆盖集最大点权独立集

```
/*点覆盖集：无向图G的一个点集，使得该图中所有边都至少有一个端点在该点集中。
点独立集：无向图G的一个点集，使得该点集中任意两个点之间没有边
最大点权独立集:
      等价于 sum - 最小点权覆盖集
限制:
      网络流的解法只能用于二分图
模型转换:
      二分图最小点权覆盖集转换为网络流的最小割
算法:
      sum = val(点集X) + val(点集Y)
      所有的v属于X连<s,v>点权为val(v)所有的v属于Y连<v,t> 点权为val(v),原图的<v,u>连<v,u>cap = INF
      求解最小割 = 最小点权覆盖集
      原理：网络流中的任意一个割都对应一个点权覆盖集，假设有没被覆盖的边e<u,v>，
              则点<s,u>与<v,t>没被割那么必有从s到t的流量不为0的增广路与割的定义矛盾
      量化关系：最大流的最小割定理
代码: HOJ matrix I
染色,标号，网络流*/
scanf("%d %d", &n, &m);
memset(check, false, sizeof (check));
ans = 0;
for (int i = 0; i < n; i++)
{
    for (int j = 0; j < m; j++)
    {
        scanf("%d", &matrix[i][j]);
        ans += matrix[i][j];
    }
}
int a = 1;
for (int i = 0; i < n; i++)
{
    for (int j = 0; j < m; j++)
        num[i][j] = a++;
}
pair<int, int> beg, temp;
beg.first = beg.second = 0;
q.push(beg);
color[0][0] = 1;
```

```cpp
check[0][0] = true;
s = 0;
t = n * m + 1;
while (!q.empty())
{
    beg = q.front();
    q.pop();
    for (int i = 0; i < 4; i++)
    {
        temp.first = beg.first + dx[i];
        temp.second = beg.second + dy[i];
        if (judge(temp.first, temp.second))
        {
            color[temp.first][temp.second] = !color[beg.first][beg.second];
            check[temp.first][temp.second] = true;
            q.push(temp);
        }
    }
}
for (int i = 0; i < n; i++)
{
    for (int j = 0; j < m; j++)
    {
        if (color[i][j])
        {
            addedge(s, num[i][j], matrix[i][j]);
            addedge(num[i][j], s, 0);
            for (int k = 0; k < 4; k++)
            {
                temp.first = i + dx[k];
                temp.second = j + dy[k];
                if (temp.first >= 0 && temp.second >= 0 && temp.first < n && temp.second < m)
                {
                    addedge(num[i][j], num[temp.first][temp.second], INF);
                    addedge(num[temp.first][temp.second], num[i][j], 0);
                }
            }
        }
        else
        {
            addedge(num[i][j], t, matrix[i][j]);
            addedge(t, num[i][j], 0);
        }
```

```
    }
}
printf("%d\n", ans - maxflowsap(t + 1, s, t));
/*最小点权覆盖的输出:
    最小割中的所有割边的有权点就是最小点覆盖集合的点集*/
POJ destroying the graph
void dfs(int x, int color)
{
    chk[x] = color;
    for (edge * i = g[x]; i != NULL; i = i->nxt)
    {
        if (!chk[i->v] && i->cap > 0)
            dfs(i->v, color);
    }
}

void output(int n)
{
    memset(chk, 0, sizeof (chk));
    memset(ans, 0, sizeof (ans));
    int cnt = 0;
    dfs(0, 1);
    dfs(2 * n + 1, 2);
    for (edge *j = g[0]; j != NULL; j = j->nxt)
        if (chk[j->v] != 0 && chk[j->v] != chk[0])
            ans[j->v] = 1;
    for (edge *j = g[2 * n + 1]; j != NULL; j = j->nxt)
        if (chk[j->v] != 0 && chk[j->v] != chk[2 * n + 1])
            ans[j->v] = 1;
    for (int i = 1; i <= n; i++)
    {
        if (ans[i])
            cnt++;
        if (ans[i + n])
            cnt++;
    }
    printf("%d\n", cnt);
    for (int i = 1; i <= n; i++)
    {
        if (ans[i])
            printf("%d -\n", i);
        if (ans[i + n])
            printf("%d +\n", i);
```

```
    }
}
```

二分图最大点权团 = 原图的补图的最大点权独立集(补图中认为 x 集合（y 集合）的元素之间都有边,原图反之)

网络流之最大权闭合图

```
/*最大权闭合图：（闭合图是一个子图）
    在一个图中,我们选取一些点构成集合,记为 V,且集合中的出边(即集合中的点的向外连出的弧),所指向的终点(弧头)也在 V 中，则我们称 V 为闭合图。最大权闭合图即在所有闭合图中，集合中点的权值之和最大的 V，我们称 V 为最大权闭合图。
建图都比较隐蔽，依靠的定理是最大流的最小割定理
建图:
    原图 G(E,V) 新图 G(EN,VE)
    在原图点集的基础上增加源 s 和汇 t ;将原图每条有向边 <u,v> ∈ E 替换为容量为 c(u,v) = ∞ 的有向边 <u,v> ∈ EN ;
    增加连接源 s 到原图每个正权点 v ( wv > 0) 的有向边 s, v ∈ EN , 容量为 c( s, v) = wv ;
    增加连接原图每个负权点 v ( wv < 0) 到汇 t 的有向边 v, t ∈ EN , 容量为 c(v, t ) = - wv 。
    其中，正无限 ∞ 定义为任意一个大于 ∑ wv 的整数。v∈V
    在该问题中，最小割是一个简单割（割中的每一个边都于 s 相连或者与 t 相连）,割开后的集合 V1 是就成了一个闭合图
    按照闭合图权的定义:权值为闭合图中正值点的权和减去负值点的权值和的绝对值 所以求闭合图 V1 的权 = sum（V(val > 0)） - cut
    取最小割即可
输出最大权闭合图:
    从 s 做一次 DFS,所有 reach 过的点，便是最大权闭合图中的点
代码: */
bool chk[V];
void dfs(int x, int n)
{
        chk[x] = true;
        for (edge * i = g[x]; i != NULL; i = i->nxt)
        {
                if (!chk[i->v] && i->cap > 0)
                        dfs(i->v, n);
        }
}
int find_cut(int n)
{
        memset(chk, false, sizeof (chk));
        dfs(0, n);
```

```
        int ans = 0;
        for (int i = 1; i <= n; i++)
                ans += chk[i];
        return ans;
}
```