

网络流模板

——by ronaflx

目录

网络流模板	1
——by ronaflx	1
网络流 sap 模板	2
费用流模板	4
网络流之上下界可行流	7
网络流之关键割	8
网络流之混合图欧拉迹	9
网络流之点连通度	11
网络流之二分图最小点权覆盖集最大点权独立集	12
网络流之最大权闭合图	15

网络流 sap 模板

```
001 #include <iostream>
002 #include <cstring>
003 #include <cstdio>
004 using namespace std;
005
006 class sap
007 {
008 private:
009     const static int V = 6000;
010     const static int E = 1000001;
011     const static int INF = 1000000000;
012     int dis[V], numdis[V];
013
014     struct edge
015     {
016         int v;
017         int cap;
018         edge *nxt;
019     } pool[E], *g[V], *pp;
020
021     void bfs(int i, int n)
022     {
023         int que[V], tail = 0;
024         bool vst[V];
025         memset(vst, 0, sizeof (vst));
026         memset(numdis, 0, sizeof (numdis));
027         for (int j = 0; j < n; j++)
028             dis[j] = n;
029         dis[i] = 0;
030         numdis[0]++;
031         vst[i] = 1;
032         que[0] = i;
033         for (int j = 0; j <= tail; j++)
034             for (edge *ip = g[que[j % n]]; ip != NULL; ip = ip->nxt)
035                 if (pool[(ip - pool)^1].cap > 0 && !vst[ip->v])
036                 {
037                     tail++;
038                     vst[ip->v] = 1;
039                     que[tail % n] = ip->v;
040                     dis[ip->v] = dis[que[j % n]] + 1;
041                     numdis[dis[ip->v]]++;
```

```

042         }
043     }
044
045 public:
046
047     void firststart()
048     {
049         pp = pool;
050         memset(g, 0, sizeof (g));
051     }
052
053     void addedge(int i, int j, int cap)
054     {
055         pp->v = j;
056         pp->cap = cap;
057         pp->nxt = g[i];
058         g[i] = pp++;
059     }
060
061     int maxflowsap(int n, int s, int t)
062     {
063         bfs(t, n);
064         int nowv = s, skip, pre[V];
065         int ans = 0;
066         edge * nowe[V], *pree[V];
067         for (int j = 0; j < n; j++)
068             nowe[j] = g[j];
069         while (dis[s] < n)
070         {
071             skip = 0;
072             while (nowe[nowv])
073             {
074                 if (nowe[nowv]->cap > 0 && dis[nowv]==dis[nowe[nowv]->v]+1)
075                 {
076                     pre[nowe[nowv]->v] = nowv;
077                     pree[nowe[nowv]->v] = nowe[nowv];
078                     nowv = nowe[nowv]->v;
079                     if (nowv == t)
080                     {
081                         int minf = INF;
082                         for (int tmpv = t; tmpv != s; tmpv = pre[tmpv])
083                             minf = min(minf, pree[tmpv]->cap);
084                         ans += minf;
085                         for (int tmpv = t; tmpv != s; tmpv = pre[tmpv])

```

```

086             {
087                 pree[tmpv]->cap -= minf;
088                 pool[(pree[tmpv] - pool)^0x1].cap += minf;
089             }
090             nowv = s;
091             skip = 1;
092             break;
093         }
094         continue;
095     }
096     nowe[nowv] = nowe[nowv]->nxt;
097 }
098 if (!skip)
099 {
100     int mindis = INF;
101     numdis[dis[nowv]]--;
102     if (!numdis[dis[nowv]])
103         break;
104     for (edge *ip = g[nowv]; ip != NULL; ip = ip->nxt)
105         if (ip->cap > 0)
106             mindis = min(mindis, dis[ip->v] + 1);
107     if (mindis == INF)
108         dis[nowv] = n;
109     else
110         dis[nowv] = mindis;
111     numdis[dis[nowv]]++;
112     nowe[nowv] = g[nowv];
113     if (nowv != s)
114         nowv = pre[nowv];
115 }
116 }
117 return ans;
118 }
119 };
120 sap maxflow;

```

费用流模板

```

001 #include <iostream>
002 #include <cstring>
003 #include <cstdio>
004 #include <algorithm>
005 using namespace std;

```

```

006
007 class mincost
008 {
009 private:
010     static const int V = 1000;
011     static const int E = 1000001;
012     static const int INF = 0x7fffffff;
013
014     struct Edge
015     {
016         int v, cap, cost;
017         Edge *next;
018     } pool[E], *g[V], *pp, *pree[V];
019     int T, S, dis[V], pre[V];
020     int n, m;
021
022     void SPFA()
023     {
024         bool vst[V] = {false};
025         int cirq[V];
026         int tail = 0, u;
027         fill(dis, dis + T + 1, 0x7fffffff);
028         cirq[0] = S;
029         vst[S] = 1;
030         dis[S] = 0;
031         for (int j = 0; j <= tail; j++)
032         {
033             int v = cirq[j % n];
034             for (Edge *i = g[v]; i != NULL; i = i->next)
035             {
036                 if (!i->cap)
037                     continue;
038                 u = i->v;
039                 if (i->cost + dis[v] < dis[u])
040                 {
041                     dis[u] = i->cost + dis[v];
042                     pree[u] = i;
043                     pre[u] = v;
044                     if (!vst[u])
045                     {
046                         tail++;
047                         cirq[tail % n] = u;
048                         vst[u] = true;
049                     }

```

```

050         }
051     }
052     vst[v] = false;
053 }
054 }
055
056 public:
057
058     inline void addedge(int i, int j, int cap, int cost)
059     {
060         pp->cap = cap;
061         pp->v = j;
062         pp->cost = cost;
063         pp->next = g[i];
064         g[i] = pp++;
065     }
066
067     void initialize()
068     {
069         memset(g, 0, sizeof (g));
070         pp = pool;
071     }
072
073     int mincost_maxflow(int n, int s, int t)
074     {
075         S = s;
076         T = t;
077         this->n = n;
078         int flow = 0;
079         while (true)
080         {
081             SPFA();
082             if (dis[T] == INF)
083                 break;
084             int minn = INF;
085             for (int i = T; i != S; i = pre[i])
086                 minn = min(minn, pree[i]->cap);
087             for (int i = T; i != S; i = pre[i])
088             {
089                 pree[i]->cap -= minn;
090                 pool[(pree[i] - pool)^0x1].cap += minn;
091                 flow += minn * pree[i]->cost;
092             }
093         }

```

```

094         return flow;
095     }
096
097 };

```

网络流之上下界可行流

```

01  /*无源无汇上下界可行流:
02  设  $M(i)$  为流入结点  $i$  的下界总和减去流出  $i$  的下界总和。
03  设一附加源  $S$ , 汇  $T$ , 则可以令  $C'(S, i) = M(i)$ 。
04  如果  $M(i)$  是负数, 那么有  $C'(i, T) = -M(i)$ 。
05  对于其他的边有  $C'(u, v) = C(u, v) - B(u, v) */$ 
06  for(int i = 0; i < m; i++)
07  {
08      scanf("%d %d %d %d", &vfrom, &vto, &low[i], &tmpcap);
09      maxflow.addedge(vfrom, vto, tmpcap - low[i]);
10      maxflow.addedge(vto, vfrom, 0);
11      b[vto] += low[i];
12      b[vfrom] -= low[i];
13  }
14  for (int i = 1; i <= n; i++)
15  {
16      if (b[i] > 0)
17      {
18          maxflow.addedge(s, i, b[i]);
19          maxflow.addedge(i, s, 0);
20          tmpans += b[i];
21      }
22      else
23      {
24          maxflow.addedge(i, t, -b[i]);
25          maxflow.addedge(t, i, 0);
26      }
27  }
28  /*有源有汇上下界可行流的解法:
29  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  $INF$ 
30  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

```
01 /*关键割:
02 如果增加的该边的流量, 那么网络流的流量会增加
03 注意一个初学者常犯的错误:
04 满流的边不一定是关键割, 所以 floodfill or dfs 的时候要遍历正向和反向边
05
06 算法:
07 开始给所有的点标记为 0
08 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
09 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
10 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
11 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*/
12
13 bool floodfill(int beg, int *reach, int clr)
14 {
15     queue<int> q;
16     q.push(beg);
17     reach[beg] = clr;
18     while (!q.empty())
19     {
20         beg = q.front();
21         q.pop();
22         for (Edges *i = vfrom[beg]; i != NULL; i = i->next)
23         {
24             if (!reach[i->vto] && i->ecap)
25             {
26                 reach[i->vto] = clr;
27                 q.push(i->vto);
28             }
29         }
30     }
31     return reach[roof];
32 }
33 maxflow.maxflowsap(n, s, t);
34 memset(reach, 0, sizeof (reach));
35 floodfill(s, reach, 1);
36 floodfill(t, reach, 2);
37 cnt = INF;
38 for (int i = 0; i < n; i++)
39 {
40     for (Edges *j = vfrom[i]; j != NULL; j = j->next)
41     {
```



```

42         if (reach[i] == 1 && reach[j->vto] == 2 && ((j - pool) & 1) == 0)
43         {
44             j->ecap++;
45             int vst[V];
46             memset(vst, 0, sizeof (vst));
47             if (floodfill(s, vst, 1))
48                 cnt = min(cnt, j->id);
49             j->ecap--;
50         }
51
52     }
53 }

```

网络流之混合图欧拉迹

01 /*算法:

02 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。

03 代码: */

```

04 const int N = 201, E = 2000;
05 int indgr[N], outdgr[N], vfrom[E], vto[E], d[E];
06
07 bool check(int n)
08 {
09     for (int i = 1; i <= n; i++)
10         if ((indgr[i] - outdgr[i]) & 1)
11             return false;
12     return true;
13 }
14
15 int main()
16 {
17     int csc, s, t, n, m, allt, alls;
18     scanf("%d", &csc);
19     while (csc--)
20     {
21         scanf("%d %d", &n, &m);
22         s = 0, t = n + 1, allt = alls = 0;
23         memset(indgr, 0, sizeof (indgr));
24         memset(outdgr, 0, sizeof (outdgr));

```

```

25     for (int i = 1; i <= m; i++)
26     {
27         scanf("%d %d %d", &vfrom[i], &vto[i], &d[i]);
28         outdgr[vfrom[i]]++;
29         indgr[vto[i]]++;
30     }
31     if (!check(n))
32         printf("impossible\n");
33     else
34     {
35         maxflow.firststart();
36         for (int i = 1; i <= n; i++)
37         {
38             if (indgr[i] > outdgr[i])
39             {
40                 maxflow.addedge(i, t, (indgr[i] - outdgr[i]) / 2);
41                 maxflow.addedge(t, i, 0);
42                 allt += indgr[i] - outdgr[i];
43             }
44             else if (indgr[i] < outdgr[i])
45             {
46                 maxflow.addedge(s, i, (outdgr[i] - indgr[i]) / 2);
47                 maxflow.addedge(i, s, 0);
48                 alls += outdgr[i] - indgr[i];
49             }
50         }
51         for (int i = 1; i <= m; i++)
52         {
53             if (d[i])
54                 continue;
55             maxflow.addedge(vfrom[i], vto[i], 1);
56             maxflow.addedge(vto[i], vfrom[i], 0);
57         }
58         allt /= 2;
59         alls /= 2;
60         if (allt != alls || allt != maxflow.maxflowsap(t + 1, s, t))
61             printf("impossible\n");
62         else
63             printf("possible\n");
64     }
65 }
66 return 0;
67 }

```

网络流之点连通度

01 /*点联通度：在图中去掉多少个点，使图边的不联通

02 算法：

03 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

04 从 v'' 到 v' 连容量为 1 原图上的边 $\langle u, v \rangle$ 在新图上构建一个 $\langle u', v'' \rangle$ 容量为 INF

05 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

06 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 $ans = n$;

07 复杂度 $O(n^5)$ 网络流，敢写就有奇迹！！！！

08 代码：*/

09 **int** ans = INF;

10 **for** (**int** j = 0; j < n; j++)

11 {

12 **for** (**int** k = j + 1; k < n; k++)

13 {

14 maxflow.firststart();

15 **for** (**int** i = 0; i < n; i++)

16 {

17 maxflow.addedge(i * 2, i * 2 + 1, 1);

18 maxflow.addedge(i * 2 + 1, i * 2, 0);

19 }

20 **for** (**int** i = 0; i < m; i++)

21 {

22 maxflow.addedge(vfrom[i] * 2 + 1, vto[i] * 2, INF);

23 maxflow.addedge(vto[i] * 2, vfrom[i] * 2 + 1, 0);

24 maxflow.addedge(vto[i] * 2 + 1, vfrom[i] * 2, INF);

25 maxflow.addedge(vfrom[i] * 2, vto[i] * 2 + 1, 0);

26 }

27 s = j * 2 + 1, t = k * 2;

28 ans = min(ans, maxflow.maxflowsap(n * 2, s, t));

29 }

30 }

31 **if** (ans == INF)

32 printf("%d\n", n);

33 **else**

34 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

```
001 /*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。
002 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
003 最大点权独立集：
004     等价于 sum - 最小点权覆盖集
005 限制：
006     网络流的解法只能用于二分图
007 模型转换：
008     二分图最小点权覆盖集转换为网络流的最小割
009 算法：
010     sum = val(点集X) + val(点集Y)
011     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
012     求解最小割 = 最小点权覆盖集
013     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>,
014         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
015     量化关系：最大流的最小割定理
016 代码：HOJ matrix I
017 染色,标号,网络流*/
018 scanf("%d %d", &n, &m);
019 memset(check, false, sizeof (check));
020 ans = 0;
021 for (int i = 0; i < n; i++)
022 {
023     for (int j = 0; j < m; j++)
024     {
025         scanf("%d", &matrix[i][j]);
026         ans += matrix[i][j];
027     }
028 }
029 int a = 1;
030 for (int i = 0; i < n; i++)
031 {
032     for (int j = 0; j < m; j++)
033         num[i][j] = a++;
034 }
035 pair<int, int> beg, temp;
036 beg.first = beg.second = 0;
037 q.push(beg);
038 color[0][0] = 1;
```

```

039 check[0][0] = true;
040 s = 0;
041 t = n * m + 1;
042 while (!q.empty())
043 {
044     beg = q.front();
045     q.pop();
046     for (int i = 0; i < 4; i++)
047     {
048         temp.first = beg.first + dx[i];
049         temp.second = beg.second + dy[i];
050         if (judge(temp.first, temp.second))
051         {
052             color[temp.first][temp.second] = !color[beg.first][beg.second];
053             check[temp.first][temp.second] = true;
054             q.push(temp);
055         }
056     }
057 }
058 for (int i = 0; i < n; i++)
059 {
060     for (int j = 0; j < m; j++)
061     {
062         if (color[i][j])
063         {
064             addedge(s, num[i][j], matrix[i][j]);
065             addedge(num[i][j], s, 0);
066             for (int k = 0; k < 4; k++)
067             {
068                 temp.first = i + dx[k];
069                 temp.second = j + dy[k];
070                 if (temp.first >= 0 && temp.second >= 0 && temp.first < n &&
temp.second < m)
071                 {
072                     addedge(num[i][j], num[temp.first][temp.second], INF);
073                     addedge(num[temp.first][temp.second], num[i][j], 0);
074                 }
075             }
076         }
077         else
078         {
079             addedge(num[i][j], t, matrix[i][j]);
080             addedge(t, num[i][j], 0);
081         }

```

```

082     }
083 }
084 printf("%d\n", ans - maxflowsap(t + 1, s, t));
085 /*最小点权覆盖的输出:
086     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*/
087 POJ destroying the graph
088 void dfs(int x, int color)
089 {
090     chk[x] = color;
091     for (edge * i = g[x]; i != NULL; i = i->nxt)
092     {
093         if (!chk[i->v] && i->cap > 0)
094             dfs(i->v, color);
095     }
096 }
097
098 void output(int n)
099 {
100     memset(chk, 0, sizeof (chk));
101     memset(ans, 0, sizeof (ans));
102     int cnt = 0;
103     dfs(0, 1);
104     dfs(2 * n + 1, 2);
105     for (edge *j = g[0]; j != NULL; j = j->nxt)
106         if (chk[j->v] != 0 && chk[j->v] != chk[0])
107             ans[j->v] = 1;
108     for (edge *j = g[2 * n + 1]; j != NULL; j = j->nxt)
109         if (chk[j->v] != 0 && chk[j->v] != chk[2 * n + 1])
110             ans[j->v] = 1;
111     for (int i = 1; i <= n; i++)
112     {
113         if (ans[i])
114             cnt++;
115         if (ans[i + n])
116             cnt++;
117     }
118     printf("%d\n", cnt);
119     for (int i = 1; i <= n; i++)
120     {
121         if (ans[i])
122             printf("%d -\n", i);
123         if (ans[i + n])
124             printf("%d +\n", i);

```

125 }

126 }

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

01 /*最大权闭合图: (闭合图是一个子图)

02 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。

03 建图都比较隐蔽,依靠的定理是最大流的最小割定理

04 建图:

05 原图 $G(E, V)$ 新图 $G(EN, VE)$

06 在原图点集的基础上增加源 s 和汇 t ;将原图每条有向边 $\langle u, v \rangle \in E$ 替换为容量为 $c(u, v) = \infty$ 的有向边 $\langle u, v \rangle \in EN$;

07 增加连接源 s 到原图每个正权点 v ($w_v > 0$) 的有向边 $s, v \in EN$, 容量为 $c(s, v) = w_v$;

08 增加连接原图每个负权点 v ($w_v < 0$) 到汇 t 的有向边 $v, t \in EN$, 容量为 $c(v, t) = -w_v$ 。

09 其中,正无限 ∞ 定义为任意一个大于 $\sum w_v$ 的整数。 $v \in V$

10 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

11 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 = $\text{sum}(V(\text{val} > 0)) - \text{cut}$

12 取最小割即可

13 输出最大权闭合图:

14 从 s 做一次 DFS,所有 reach 过的点,便是最大权闭合图中的点

15 代码: */

16 bool chk[V];

17 void dfs(int x, int n)

18 {

19 chk[x] = true;

20 for (edge * i = g[x]; i != NULL; i = i->nxt)

21 {

22 if (!chk[i->v] && i->cap > 0)

23 dfs(i->v, n);

24 }

25 }

26 int find_cut(int n)

27 {

28 memset(chk, false, sizeof(chk));

29 dfs(0, n);

```
30     int ans = 0;
31     for (int i = 1; i <= n; i++)
32         ans += chk[i];
33     return ans;
34 }
```