

As an ACMer, not only try, but also create.

公告

昵称: wide sea
园龄: 7个月
粉丝: 1
关注: 1
+ 加关注

日历

<	2012年8月						>
日	一	二	三	四	五	六	
29	30	31	1	2	3	4	
5	6	7	8	9	10	11	
12	13	14	15	16	17	18	
19	20	21	22	23	24	25	
26	27	28	29	30	31	1	
2	3	4	5	6	7	8	

统计

随笔 - 30
文章 - 0
评论 - 0
引用 - 0

导航

博客园
首页
发新随笔
发新文章
联系
订阅 XML
管理

搜索

ACM暑假集训之二分匹配

点集、边集与匹配

点集中包含点支配集和点覆盖集，边集包含边覆盖集和边独立集（匹配）。

<注：参考《图论算法理论、实现及应用》第七章——支配集、覆盖集、独立集与匹配>

匹配问题：

- 1 关键词：
 - 1.1 二部图的最大匹配、完美匹配、完备匹配、最佳匹配。
 - 1.2 交错轨、可增广轨
 - 1.3 网络流、匈牙利算法

2 求解算法；

2.1 网络流算

法：http://www.cnblogs.com/tiantianhaoweidao/archive/2012/08/16/2643018.html

需要构建二部图，建图的方式可通过拆点的方式建立，再自己设定源点S和汇点T，然后让每一条的边权设为1，最后求S à T 的最大流（推荐 SAP + gap 数组 优化算法）。

这种求解的方式可以求边权不为1的最佳匹配，只需参考最小费用最大流求解思维。

2.2 匈牙利算法：

2.2.1 DFS增广：

采用DFS思想搜索可增广路并求最大匹配。

优点：实现简洁，理解容易；

适用：稠密图，由于边多，DFS找增广路很快。

复杂度：O(N³)/O(NM)。

View Code

```
1 #include<cstdio> // DFS
```

常用链接

我的随笔
我的评论
我的参与
最新评论
我的标签

我的标签

map(1)
set(1)
八数码(1)
二部图(1)
高斯消元(1)
割顶集(1)
双连通(1)

随笔分类

纯搜索(7)
动态规划(1)
二部图匹配(8)
矩阵-高斯消元(1)
算法回顾(5)
图的连通(6)
网络流(3)
最短路
最小生成树

随笔档案

2013年1月 (13)
2012年12月 (1)
2012年11月 (4)
2012年10月 (6)
2012年9月 (1)
2012年8月 (3)
2012年7月 (2)

最新评论

阅读排行榜

1. ACM暑假集训之网络流初章(297)
2. ACM暑假集训之最小生成树(145)
3. Tempter of the Bone_ZOJ 2110(121)
4. poj 2516 Minimum Cost 最小费用最大流(77)
5. Red and Black_poj 1979(70)

```

2 #include<cstring>
3 #include<algorithm>
4 using namespace std;
5 const int MAXN = 305;
6 int g[MAXN][MAXN];
7 int cx[MAXN], cy[MAXN], nx, ny;
8 bool vis[MAXN];
9 int dfs( int u )
10 {
11     int v;
12     for( v = 1; v <= ny; v++ )
13     {
14         if( g[u][v] && !vis[v] )
15         {
16             vis[v] = true;
17             if( !cy[v] || dfs( cy[v] ) )
18             {
19                 cx[u] = v;
20                 cy[v] = u;
21                 return 1;
22             }
23         }
24     }
25     return 0;
26 }
27 int MaxMatch()
28 {
29     int u, cnt;
30     memset( cx, 0, sizeof(cx) );
31     memset( cy, 0, sizeof(cy) );
32     cnt = 0;
33     for( u = 1; u <= nx; u++ )
34     {
35         if( !cx[u] )
36         {
37             memset( vis, false, sizeof(vis) );
38             cnt += dfs( u );
39         }
40     }
41     return cnt;
42 }
43 int main()
44 {
45     int n, m, i, u, v;
46     scanf( "%d%d", &n, &m );
47     nx = n;
48     ny = m;
49     memset( g, false, sizeof(g) );
50     for( i = 1; i <= m; i++ )
51     {
52         scanf( "%d%d", &u, &v );
53         g[u][v] = true;
54     }
55     printf( "%d\n", MaxMatch() );
56     return 0;

```

评论排行榜

1. poj 1422 最大匹配基础应用(O)
2. poj 1325 最小点覆盖集(O)
3. poj 1466 最大独立点数(O)
4. poj 3041 二分匹配基础(O)
5. poj 1469 二分匹配基础(O)

推荐排行榜

57 }



2.2.2 BFS增广

采用BFS思想搜索可增广路并求最大匹配。

适用：稀疏二部图，边少，增广路短。

复杂度： $O(N^3)/O(NM)$ 。

View Code



```

1 #include<cstdio>
2 #include<cstring>
3 #include<algorithm>
4 using namespace std;
5 const int MAXN = 305;
6 int cx[MAXN], cy[MAXN], nx, ny;
7 bool g[MAXN][MAXN];
8 int pre[MAXN];
9 int queue[MAXN*MAXN], front, rear;
10 int MaxMatch()
11 {
12     int u, v, y, cnt;
13     memset( cx, 0, sizeof(cx) );
14     memset( cy, 0, sizeof(cy) );
15     cnt = 0;
16     for( u = 1; u <= nx; u++ )
17     {
18         if( cx[u] ) continue;
19         for( v = 1; v <= ny; v++ ) pre[v] = -2;
20         front = rear = 0;
21         for( v = 1; v <= ny; v++ ) if( g[u][v] )
22         {
23             pre[v] = -1;    queue[rear++] = v;
24         }
25         while( front != rear )
26         {
27             y = queue[front];
28             if( !cy[v] ) break;
29             front++;
30             for( v = 1; v <= ny; v++ )
31             {
32                 if( pre[v] == -2 && g[cy[y]][v] )
33                 {
34                     pre[v] = y;
35                     queue[rear++] = v;
36                 }
37             }
38         }
39         if( front == rear ) continue;
40         while( pre[y] > -1 )

```

```
41     {
42         cx[cy[pre[y]]] = y;
43         cy[y] = cy[pre[y]];
44         y = pre[y];
45     }
46     cy[y] = u;
47     cx[u] = y;
48     cnt++;
49 }
50 return cnt;
51 }
52 int main()
53 {
54     int n, m, u, v, i;
55     scanf( "%d%d", &n, &m );
56     memset( g, false, sizeof(g) );
57     nx = ny = n;
58     for( i = 1; i <= m ; i++ )
59     {
60         scanf( "%d%d", &u, &v );
61         g[u][v] = true;
62     }
63     printf( "%d\n", MaxMatch() );
64     return 0;
65 }
```



2.2.3 HK算法

BFS构建层级 + DFS找增路路，效率高，很强大，可以处理上万顶点的二部图，使用邻接表。（测试数据：poj 1469 407ms）

View Code



```
1 #include<cstdio>
2 #include<cstring>
3 #include<algorithm>
4 #include<queue>
5 using namespace std;
6 #define MAXN 500
7 struct Edge{                                // 使用邻接表用来处理多点多边的情况，如使
8     int v, next;                             用使用矩阵建议不采用HK算法，可用dfs或者bfs
9 }Arc[MAXN*MAXN];
10 int tot, head[MAXN];
11 int cx[MAXN*2], cy[MAXN*2], nx, ny; // cx[],cy[]记录匹配定点，nx,ny为二部
12 int rk[MAXN*2];                            图分别为上下部点个数
13 queue<int>Q;                                // 各顶点层级记录数组，注：y部层级记录在 i+nx
14 void init()
15 {
16     tot = 0;
```

```

17     memset( head, -1, sizeof(head) );
18 }
19 inline void add_edge( int u, int v )
20 {
21     Arc[tot].v = v;
22     Arc[tot].next = head[u];
23     head[u] = tot++;
24 }
25 bool bfs()                // 层级构建
26 {
27     bool flag;
28     int i, u, v;
29     flag = false;
30     while( !Q.empty() )Q.pop();
31     for( i = 1; i <= nx; i++ )if( !cx[i] )
32         Q.push( i );
33     for( i = 0; i <= nx + ny; i++ )
34         rk[i] = 0;
35     while( !Q.empty() )
36     {
37         u = Q.front();    Q.pop();
38         for( i = head[u]; i != -1; i = Arc[i].next )
39         {
40             v = Arc[i].v;
41             if( !rk[v+nx] )
42             {
43                 rk[v+nx] = rk[u] + 1;
44                 if( cy[v] )
45                 {
46                     rk[cy[v]] = rk[v+nx] + 1;
47                     Q.push( cy[v] );
48                 }
49                 else flag = true;
50             }
51         }
52     }
53     return flag;
54 }
55 bool dfs( int u )
56 {
57     int i, v;
58     for( i = head[u]; i != -1; i = Arc[i].next )
59     {
60         v = Arc[i].v;
61         if( rk[v+nx] == rk[u] + 1 )
62         {
63             rk[v+nx] = 0;
64             if( !cy[v] || dfs( cy[v] ) )
65             {
66                 cx[u] = v;
67                 cy[v] = u;
68                 return 1;
69             }
70         }
71     }

```

```

72     return 0;
73 }
74 int HK()
75 {
76     int u, ans;
77     ans = 0;
78     memset( cx, 0, sizeof(int) * (nx + 1) );
79     memset( cy, 0, sizeof(int) * (ny + 1) );
80     while( bfs() ) // HK 的优化
81         for( u = 1; u <= nx; u++ )
82             if( !cx[u] )
83                 ans += dfs( u );
84     return ans;
85 }
86 int main()
87 {
88     int i, u, v, cas, num, P, N;
89     scanf( "%d", &cas );
90     while( cas-- )
91     {
92         scanf( "%d%d", &P, &N );
93         nx = P;
94         ny = N;
95         init();
96         for( u = 1; u <= nx; u++ )
97         {
98             scanf( "%d", &num );
99             for( i = 1; i <= num; i++ )
100             {
101                 scanf( "%d", &v );
102                 add_edge( u, v );
103             }
104         }
105         if( HK() == P )printf( "YES\n" );
106         else printf( "NO\n" );
107     }
108     return 0;
109 }

```



也可以使用矩阵（不推荐）

View Code



```

1 #include<cstdio>
2 #include<cstring>
3 #include<algorithm>
4 #include<queue>
5 using namespace std;
6 #define MAXN 305
7 int xs[MAXN], ys[MAXN];
8 int P, N, nx, ny;
9 bool g[MAXN][MAXN];
10 int rk[MAXN*2];

```

```

11 queue<int>Q;
12 bool bfs()
13 {
14     bool flag;
15     int i, u, v;
16     flag = false;
17     while( !Q.empty() )Q.pop();
18     for( i = 1; i <= nx; i++ )if( !xs[i] )
19         Q.push( i );
20     for( i = 0; i <= nx + ny; i++ )
21         rk[i] = 0;
22     while( !Q.empty() )
23     {
24         u = Q.front();    Q.pop();
25         for( v = 1; v <= ny; v++ )if( g[u][v] )
26         {
27             if( !rk[v+nx] )
28             {
29                 rk[v+nx] = rk[u] + 1;
30                 if( ys[v] )
31                 {
32                     rk[ys[v]] = rk[v+nx] + 1;
33                     Q.push( ys[v] );
34                 }
35                 else flag = true;
36             }
37         }
38     }
39     return flag;
40 }
41 int dfs( int u )
42 {
43     int v;
44     for( v = 1; v <= ny; v++ )
45     {
46         if( g[u][v] && rk[v+nx] == rk[u] + 1 )
47         {
48             rk[v+nx] = 0;
49             if( !ys[v] || dfs( ys[v] ) )
50             {
51                 xs[u] = v;
52                 ys[v] = u;
53                 return 1;
54             }
55         }
56     }
57     return 0;
58 }
59 int MaxMatch()
60 {
61     int u, tot = 0;
62     memset( xs, 0, sizeof(xs) );
63     memset( ys, 0, sizeof(ys) );
64     while( bfs() )
65         for( u = 1; u <= nx; u++ )

```

```

66         if( !xs[u] )
67             tot += dfs( u );
68         return tot;
69     }
70 int main()
71 {
72     int num, u, j, v, cas;
73     scanf( "%d", &cas );
74     while( cas-- )
75     {
76         scanf( "%d%d", &P, &N );
77         nx = P;
78         ny = N;
79         memset( g, false, sizeof(g) );
80         for( u = 1; u <= nx; u++ )
81         {
82             scanf( "%d", &num );
83             for( j = 1; j <= num; j++ )
84             {
85                 scanf( "%d", &v );
86                 g[u][v] = true;
87             }
88         }
89         if( MaxMatch() == P )printf( "YES\n" );
90         else printf( "NO\n" );
91     }
92     return 0;
93 }

```



<注：参考《图论算法理论、实现及应用》第七章——支配集、覆盖集、独立集与匹配>

点

- 1 点支配集: u 与 v 之间存在边, 则可称 u 支配 v , 或者 v 支配 u 。(邻接顶点)
- 1.1 极小支配集: 集合中各个点两两不互相支配, 但整个集合能支配这个无向图的所有定点。集合中加入无向图任意一点, 该集合就不会是极小支配集。
- 1.2 最小支配集: 在一个无向图中所有极小支配集顶点个数最小的集合。
- 1.3 点支配数: 最小支配集所含的点数, 记作 γ_0 。
- 1.4 性质: 无孤立点图 G , 存在一个支配集, 它的补集也是支配集; 极小支配集的补集也是支配集。
- 1.5 应用例子: 在 n 个地区寻找能所有地区连通的最小支配集。
- 2 点覆盖集: 覆盖的含义是顶点“覆盖”边。
- 2.1 极小覆盖集: 该集合内去掉任意一点, 都无法覆盖所有边。
- 2.2 最小覆盖集: 顶点数最小的极小覆盖集。

2.3 点覆盖数：最小覆盖集的顶点数 α_0 。

2.4 应用例子：配置小区（每条路）消防措施（点）。

3 点独立集：集合中任意两点都不相邻。

3.1 极大点独立集：集合内加入任意一点不再是独立集。

3.2 最大点独立集：顶点数极大点独立集合。

3.3 点独立数：最大点独立集的顶点数 β_0 。

3.4 应用例子：化学药品和合理存放（图的点着色问题），设计路口交通信号灯（图的点着色问题）。

4 三者间的联系：（在无孤立点的G图中）

4.1，G的极大点独立集就是G的极小支配集；逆命题不成立，极小点独立集未必是极大独立集。

4.2 一个独立集是极大独立集，当且仅当它是一个支配集。

4.3 极大（最大）点独立集的补集是极小（最小）点覆盖集， $\alpha_0 + \beta_0 = n$ （n为顶点数）。

5 点支配集、点覆盖集、点独立集的求解：

通过逻辑运算（公式详细参考其他资料），操作所需的时间复杂度至少为 $O(2^n)$ ，目前尚无有效的精确算法，事实上，极小点支配集、极小点覆盖集合极大点独立集问题都是NP（非确定多项式Non-Deterministic Polynomial）问题，但有些问题可转化为（二部图）最大匹配来求解。

<注：参考《图论算法理论、实现及应用》第七章——支配集、覆盖集、独立集与匹配>

边

1 边覆盖集：覆盖的含义是边“覆盖”顶点。

1.1 极小边覆盖：去掉集合内任意一点都无法覆盖所有顶点。

1.2 最小边覆盖：边数最小的极小边覆盖。

1.3 边覆盖数：最小边覆盖的边数 α_1 。

1.4 应用例子：安排ACM竞赛题目讲解（一个人至少讲一题）。

2 边独立集（匹配）：集合任意两条边无公共顶点。

2.1 极大匹配：集合任意加入一条边都不再匹配。

2.2 最大匹配：边数最多的极大匹配。

2.3 边独立数：最大匹配的边数 β_1 。

2.4 盖点与未盖点：顶点是否是匹配中其中一条边的端点。

2.5 应用例子：飞行员搭配问题1（正副驾驶员的最佳分配）。

3 最大边独立集（最大匹配）与最小边覆盖集之间的联系

- 3.1 可通过最大匹配构造最小边覆盖，可通过最小边构造最大匹配
- 3.2 G 中边覆盖数 α_1 与匹配数 β_1 ，满足：边覆盖数 + 边独立数 = n ；即 $\alpha_1 + \beta_1 = n$ 。

分类: 算法回顾

绿色通道:

好文要顶

关注我

收藏该文

与我联系

wide sea

关注 - 1

粉丝 - 1

+ 加关注

0

0

推荐

反对

(请您对文章做出评价)

« 博主上一篇: ACM暑假集训之网络流初章

» 博主下一篇: Tempter of the Bone_ZOJ 2110

posted on 2012-08-21 10:30 wide sea 阅读(46) 评论(0) 编辑 收藏

[刷新评论](#) [刷新页面](#) [返回顶部](#)

注册用户登录后才能发表评论，请 [登录](#) 或 [注册](#)，[访问](#)网站首页。

[博客园首页](#) [博问](#) [新闻](#) [闪存](#) [程序员招聘](#) [知识库](#)

免费咨询

网站建设

一对一服务 省时 省力 省心

全国统一业务咨询电话:

400 188 6666

www.zhubajie.com

- 最新IT新闻:
- LG LTE智能手机全球销量突破1000万大关
 - 周鸿祎: 我不做教父 我不用微信
 - 放翁: 开放2013
 - 王垠: 漫谈 Linux, Windows 和 Mac
 - 王垠: 你好, 世界。
- » 更多新闻...

- 最新知识库文章:
- Facebook如何实现PB级别数据库自动化备份
 - 源代码管理十诫
 - 如何成为强大的程序员?
 - Xen 虚拟机架构

- NoSQL的现状
- » 更多知识库文章...

[\(永久免费\) 中小企业OA办公系统](#)
免实施，免维护，注册即用，无人数限制，44万家中小企业的选择！
www.jinggoal.com


AdChoices 

 www.cnblogs.com

Copyright © wide sea Powered by: 博客园 模板提供: 沪江博客