ACM暑假集训之二分匹配

点集、边集与匹配$\alpha_0\beta_0\alpha_1\beta_1\gamma_0$

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，再自已设定源点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：$O(N^3)/O(NM)$。

View Code

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

找找看

谷歌搜索

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)

```
#include<cstring>
#include<algorithm>
using namespace std;
const int MAXN = 305;
int g[MAXN][MAXN];
int cx[MAXN], cy[MAXN], nx, ny;
bool vis[MAXN];
int dfs( int u )
{
    int v;
    for( v = 1; v <= ny; v++ )
    {
        if( g[u][v] && !vis[v] )
        {
            vis[v] = true;
            if( !cy[v] || dfs( cy[v] ) )
            {
                cx[u] = v;
                cy[v] = u;
                return 1;
            }
        }
    }
    return 0;
}
int MaxMatch()
{
    int u, cnt;
    memset( cx, 0, sizeof(cx) );
    memset( cy, 0, sizeof(cy) );
    cnt = 0;
    for( u = 1; u <= nx; u++ )
    {
        if( !cx[u] )
        {
            memset( vis, false, sizeof(vis) );
            cnt += dfs( u );
        }
    }
    return cnt;
}
int main()
{
    int n, m, i, u, v;
    scanf( "%d%d", &n, &m );
    nx = n;
    ny = m;
    memset( g, false, sizeof(g) );
    for( i = 1; i <= m; i++ )
    {
        scanf( "%d%d", &u, &v );
        g[u][v] = true;
    }
    printf( "%d\n", MaxMatch() );
    return 0;
```

评论排行榜

1. poj 1422 最大匹配基础应用(0)
2. poj 1325 最小点覆盖集(0)
3. poj 1466 最大独立点数(0)
4. poj 3041 二分匹配基础(0)
5. poj 1469 二分匹配基础(0)

推荐排行榜

```
}
```

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

```
#include<cstdio>
#include<cstring>
#include<algorithm>
using namespace std;
const int MAXN = 305;
int cx[MAXN], cy[MAXN], nx, ny;
bool g[MAXN][MAXN];
int pre[MAXN];
int queue[MAXN*MAXN], front, rear;
int MaxMatch()
{
    int u, v, y, cnt;
    memset( cx, 0, sizeof(cx) );
    memset( cy, 0, sizeof(cy) );
    cnt = 0;
    for( u = 1; u <= nx; u++ )
    {
        if( cx[u] )continue;
        for( v = 1; v <= ny; v++ )pre[v] = -2;
        front = rear = 0;
        for( v = 1; v <= ny; v++ )if( g[u][v] )
        {
            pre[v] = -1;    queue[rear++] = v;
        }
        while( front != rear )
        {
            y = queue[front];
            if( !cy[v] )break;
            front++;
            for( v = 1; v <= ny; v++ )
            {
                if( pre[v] == -2 && g[cy[y]][v] )
                {
                    pre[v] = y;
                    queue[rear++] = v;
                }
            }
        }
        if( front == rear ) continue;
        while( pre[y] > -1 )
```

```
        {
            cx[cy[pre[y]]] = y;
            cy[y] = cy[pre[y]];
            y = pre[y];
        }
        cy[y] = u;
        cx[u] = y;
        cnt++;
    }
    return cnt;
}
int main()
{
    int n, m, u, v, i;
    scanf( "%d%d", &n, &m );
    memset( g, false, sizeof(g) );
    nx = ny = n;
    for( i = 1; i <= m ; i++ )
    {
        scanf( "%d%d", &u, &v );
        g[u][v] = true;
    }
    printf( "%d\n", MaxMatch() );
    return 0;
}
```

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

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

```
    memset( head, -1, sizeof(head) );
}
inline void add_edge( int u, int v )
{
    Arc[tot].v = v;
    Arc[tot].next = head[u];
    head[u] = tot++;
}
bool bfs()              // 层级构建
{
    bool flag;
    int i, u, v;
    flag = false;
    while( !Q.empty() )Q.pop();
    for( i = 1; i <= nx; i++ )if( !cx[i] )
        Q.push( i );
    for( i = 0; i <= nx + ny; i++ )
        rk[i] = 0;
    while( !Q.empty() )
    {
        u = Q.front();    Q.pop();
        for( i = head[u]; i != -1; i = Arc[i].next )
        {
            v = Arc[i].v;
            if( !rk[v+nx] )
            {
                rk[v+nx] = rk[u] + 1;
                if( cy[v] )
                {
                    rk[cy[v]] = rk[v+nx] + 1;
                    Q.push( cy[v] );
                }
                else flag = true;
            }
        }
    }
    return flag;
}
bool dfs( int u )
{
    int i, v;
    for( i = head[u]; i != -1; i = Arc[i].next )
    {
        v = Arc[i].v;
        if( rk[v+nx] == rk[u] + 1 )
        {
            rk[v+nx] = 0;
            if( !cy[v] || dfs( cy[v] ) )
            {
                cx[u] = v;
                cy[v] = u;
                return 1;
            }
        }
    }
```

```
    return 0;
}
int HK()
{
    int u, ans;
    ans = 0;
    memset( cx, 0, sizeof(int) * (nx + 1) );
    memset( cy, 0, sizeof(int) * (ny + 1) );
    while( bfs() )                              // HK 的优化
        for( u = 1; u <= nx; u++ )
            if( !cx[u] )
                ans += dfs( u );
    return ans;
}
int main()
{
    int i, u, v, cas, num, P, N;
    scanf( "%d", &cas );
    while( cas-- )
    {
        scanf( "%d%d", &P, &N );
        nx = P;
        ny = N;
        init();
        for( u = 1; u <= nx; u++ )
        {
            scanf( "%d", &num );
            for( i = 1; i <= num; i++ )
            {
                scanf( "%d", &v );
                add_edge( u, v );
            }
        }
        if( HK() == P )printf( "YES\n" );
        else printf( "NO\n" );
    }
    return 0;
}
```

也可以使用矩阵（不推荐）

View Code

```
#include<cstdio>
#include<cstring>
#include<algorithm>
#include<queue>
using namespace std;
#define MAXN 305
int xs[MAXN], ys[MAXN];
int P, N, nx, ny;
bool g[MAXN][MAXN];
int rk[MAXN*2];
```

```
queue<int>Q;
bool bfs()
{
    bool flag;
    int i, u, v;
    flag = false;
    while( !Q.empty() )Q.pop();
    for( i = 1; i <= nx; i++ )if( !xs[i] )
        Q.push( i );
    for( i = 0; i <= nx + ny; i++ )
        rk[i] = 0;
    while( !Q.empty() )
    {
        u = Q.front();    Q.pop();
        for( v = 1; v <= ny; v++ )if( g[u][v] )
        {
            if( !rk[v+nx] )
            {
                rk[v+nx] = rk[u] + 1;
                if( ys[v] )
                {
                    rk[ys[v]] = rk[v+nx] + 1;
                    Q.push( ys[v] );
                }
                else flag = true;
            }
        }
    }
    return flag;
}
int dfs( int u )
{
    int v;
    for(  v = 1; v <= ny; v++ )
    {
        if( g[u][v] && rk[v+nx] == rk[u] + 1 )
        {
            rk[v+nx] = 0;
            if( !ys[v] || dfs( ys[v] ) )
            {
                xs[u] = v;
                ys[v] = u;
                return 1;
            }
        }
    }
    return 0;
}
int MaxMatch()
{
    int u, tot = 0;
    memset( xs, 0, sizeof(xs) );
    memset( ys, 0, sizeof(ys) );
    while( bfs() )
       for( u = 1; u <= nx; u++ )
```

```c
        if( !xs[u] )
            tot += dfs( u );
    return tot;
}
int main()
{
    int num, u, j, v, cas;
    scanf( "%d", &cas );
    while( cas-- )
    {
        scanf( "%d%d", &P, &N );
        nx = P;
        ny = N;
        memset( g, false, sizeof(g) );
        for( u = 1; u <= nx; u++ )
        {
            scanf( "%d", &num );
            for( j = 1; j <= num; j++ )
            {
                scanf( "%d", &v );
                g[u][v] = true;
            }
        }
        if( MaxMatch() == P )printf( "YES\n" );
        else printf( "NO\n");
    }
    return 0;
}
```

<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

« 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

.

· NoSQL的现状

» 更多知识库文章...