以下算法可把 G 中任一匹配 M 扩充为最大匹配，此算法是 Edmonds 于1965年提出的，被称为匈牙利算法，其步 骤如下：

（1）首先用（*）标记 X 中所有的非 M-顶点，然后交替进行步骤（2），（3）。

（2）选取一个刚标记（用（*）或在步骤（3）中用（y_i）标记）过的 X 中顶点，例如顶点 x_i，然后用（x_i）去标记 Y 中顶点 y，如果 x_i 与 y 为同一非匹配边的两端点，且在本步骤中 y 尚未被标记过。重复步骤（2），直至对刚标记过的 X 中顶点全部完成一遍上述过程。

（3）选取一个刚标记（在步骤（2）中用（x_i）标记）过的 Y 中结点，例如 y_i，用（y_i）去标记 X 中结点 x，如果 y_i 与 x 为同一匹配边的两端点，且在本步骤中 x 尚未被标记过。重复步骤（3），直至对刚标记过的 Y 中结点全部完成一遍上述过程。

（2），（3）交替执行，直到下述情况之一出现为止：

（Ⅰ）标记到一个 Y 中顶点 y，它不是 M-顶点。这时从 y 出发循标记回溯，直到（*）标记的 X 中顶点 x，我们求得一条交替链。设其长度为2k+1，显然其中 k 条是匹配边，k+1 条是非匹配边。

（Ⅱ）步骤（2）或（3）找不到可标记结点，而又不是情况 （Ⅰ）。

（4） 当 （2），（3）步骤中断于情况（Ⅰ），则将交替链中非匹配边改为匹配边，原匹配 边改为非匹配边（从而得到一个比原匹配多一条边的新匹配），回到步骤（1），同时消除一切现有标记。

（5）对一切可能，（2）和（3）步骤均中断于情况（Ⅱ），或步骤（1）无可标记结点，算法终止（算法找不到 交替链）。

我们不打算证明算法的正确性，只用一个例子 跟踪一下算法的执行，来直观地说明这一点。

例9.3 用匈牙利算法求图9.3的 一个最大匹配。

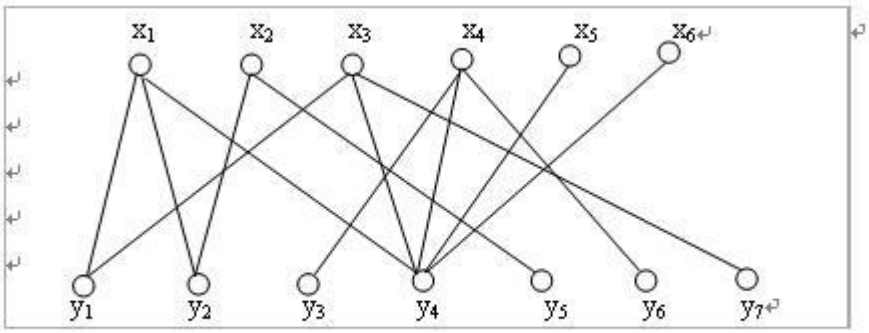

图 9.3

（1）置 M =Æ，对 x_1-x_6标记（*）。

（2）找到交替链（x_1, y_1）（由标记（x_1），（*）回溯得），置 M = {(x_1, y_1)}。

（3）找到交替链（x_2, y_2）（由标记（x_2），（*）回溯得），置 M = {(x_1, y_1), (x_2, y_2),}。

（4）找到交替链（x_3, y_1, x_1, y_4） （如图9.4所示。图中虚线表示非匹配边，细实线表示交替链中非匹配边，粗实线表示匹配边），因而得 M = {(x_2, y_2), (x_3, y_1),(x_1, y_4)}。

（5）找到交替链（x_4, y_3）（由标记（x_4），（*）回溯得），置 M = {(x_2, y_2), (x_3, y_1),(x_1, y_4), (x_4, y_3)}。

（6）找到交替链（x_5, y_4, x_1, y_1, x_3, y_7）（如图9.5所示，图中各种线段的意义同上），因

而得

$$M = \{(x_2, y_2), (x_4, y_3),(x_5, y_4), (x_1, y_1), (x_3, y_7)\}$$

即为最大匹配（如图9.6所示）。

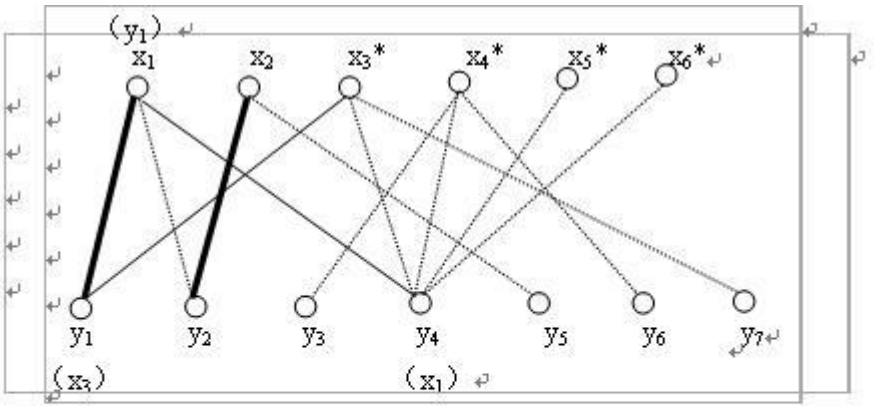

图 9.4

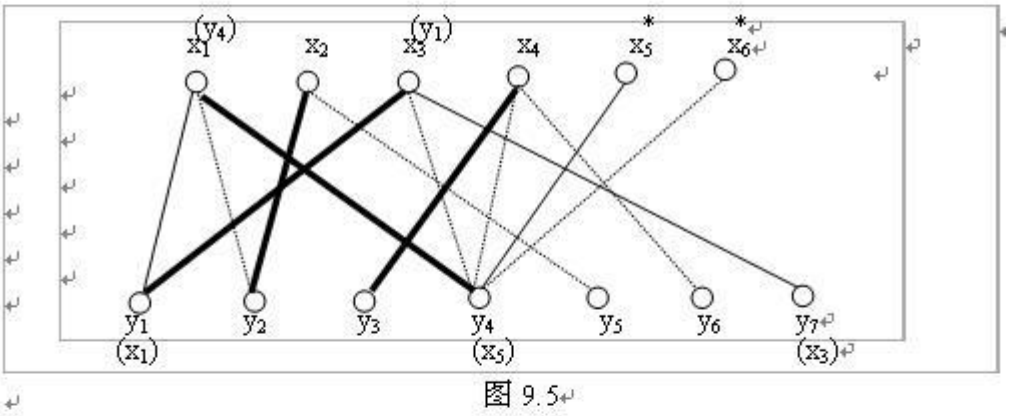

图 9.5

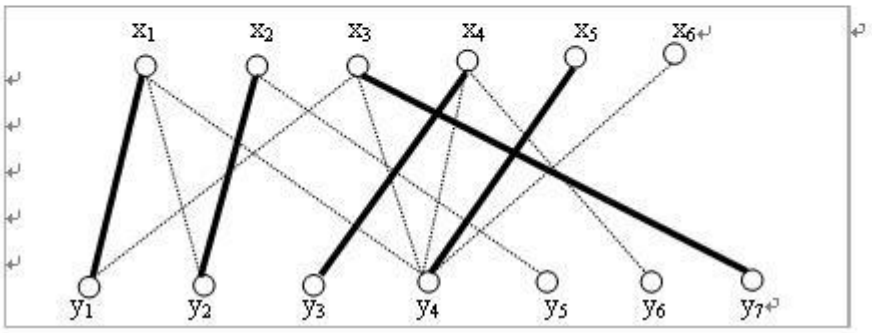

图9.6

代码实现：

实现 一般多用 dfs 实现， 简单明了

bfs 过程：

```
#include<stdio.h>
#include<string.h>
main()
{
 bool map[100][300];
 int i,i1,i2,num,num1,que[300],cou,stu,match1[100],match2[300],pque,p1,now,prev[300],n;
 scanf("%d",&n);
 for(i=0;i<n;i++)
 {
  scanf("%d%d",&cou,&stu);
  memset(map,0,sizeof(map));
  for(i1=0;i1<cou;i1++)
  {
   scanf("%d",&num);
   for(i2=0;i2<num;i2++)
   {
    scanf("%d",&num1);
    map[i1][num1-1]=true;
   }
  }
  num=0;
  memset(match1,int(-1),sizeof(match1));
  memset(match2,int(-1),sizeof(match2));
  for(i1=0;i1<cou;i1++)
  {
   p1=0;
   pque=0;
   for(i2=0;i2<stu;i2++)
   {
    if(map[i1][i2])
    {
     prev[i2]=-1;
     que[pque++]=i2;
    }
    else
     prev[i2]=-2;
   }
   while(p1<pque)
   {
    now=que[p1];
    if(match2[now]==-1)
```

```
        break;
      p1++;
      for(i2=0;i2<stu;i2++)
      {
       if(prev[i2]==-2&&map[match2[now]][i2])
       {
        prev[i2]=now;
        que[pque++]=i2;
       }
      }
     }
     if(p1==pque)
      continue;
     while(prev[now]>=0)
     {
      match1[match2[prev[now]]]=now;
      match2[now]=match2[prev[now]];
      now=prev[now];
     }
     match2[now]=i1;
     match1[i1]=now;
     num++;
    }
    if(num==cou)
     printf("YES\n");
    else
     printf("NO\n");
  }
}
```

dfs 实现过程：

```
#include<stdio.h>
#include<string.h>
#define MAX 100

bool map[MAX][MAX],searched[MAX];
int prev[MAX],m,n;

bool dfs(int data)
{
  int i,temp;
  for(i=0;i<m;i++)
  {
    if(map[data][i]&&!searched[i])
```

```
    {
      searched[i]=true;
      temp=prev[i];
      prev[i]=data;
      if(temp==-1||dfs(temp))
        return true;
      prev[i]=temp;
    }
  }
  return false;
}

main()
{
  int num,i,k,temp1,temp2,job;
  while(scanf("%d",&n)!=EOF&&n!=0)
  {
    scanf("%d%d",&m,&k);
    memset(map,0,sizeof(map));
    memset(prev,int(-1),sizeof(prev));
    memset(searched,0,sizeof(searched));
    for(i=0;i<k;i++)
    {
      scanf("%d%d%d",&job,&temp1,&temp2);
      if(temp1!=0&&temp2!=0)
        map[temp1][temp2]=true;
    }
    num=0;
    for(i=0;i<n;i++)
    {
      memset(searched,0,sizeof(searched));
      dfs(i);
    }
    for(i=0;i<m;i++)
    {
      if(prev[i]!=-1)
        num++;
    }
    printf("%d\n",num);
  }
}
```