求给定无向图的最大团

课程 算法设计与分析 ，教师 王丽珍

学院 信息学院 ，专业 计算机科学与技术

学号 20201050331 ，姓名 黄珀芝

摘 要

本次研究的算法问题为——给定无向图，求解最大团的问题，并且最好能够求出多种可行解。简单来讲，团就是一个无向图 G = < V , E >的一个子集，一个团内的顶点都与团内其余任意顶点有链接，而最大团其实就是顶点数目最多的团。

本次采用回溯算法来解决最大团问题，它是一种系统地搜索问题的解的方法。回溯算法的基本思想是：从一条路往前走，能进则进，不能进则退回来，换一条路再试。它在包含问题的所有解的解空间树中，按照深度优先（DFS）的策略，从根结点出发搜索解空间树。因为它适用于解一些组合数较大的问题,因此通过分析用回溯算法解决最大团问题能得到问题的最优解。

采用C语言实现算法，通过测试分析，程序运行结果正确，能够得出多种解。

关键词：最大团，迭代回溯算法，解空间树

**目 录**

[1 问题描述 1](#_Toc280776659)

[2 问题分析 2](#_Toc280776660)

[3 算法设计 3](#_Toc280776661)

[4 算法实现 4](#_Toc280776662)

[5 测试分析 8](#_Toc280776663)

[结 论 9](#_Toc280776664)

# 1 问题描述

团就是最大完全子图。给定无向图G=(V,E)。如果UV，且对任意u，vU 有(u，v) E，则称U 是G 的完全子图。G 的完全子图U是G的团当且仅当U不包含在G 的更大的完全子图中，即U就是最大完全子图。G 的最大团是指G中所含顶点数最多的团。通俗点讲就是在一个无向图中找出一个点数最多的完全图。

BA在用来求问题的所有解时，要回溯到根，且根结点的所有子树都已被搜索遍才结束。而BA在用来求问题的任一解时，只要搜索到问题的一个解即可结束。这种以深度优先的方式系统地搜索问题的解的算法称为[回溯法](https://baike.baidu.com/item/%E5%9B%9E%E6%BA%AF%E6%B3%95/86074" \t "https://baike.baidu.com/item/%E6%9C%80%E5%A4%A7%E5%9B%A2%E9%97%AE%E9%A2%98/_blank)，它适用于解一些[组合数](https://baike.baidu.com/item/%E7%BB%84%E5%90%88%E6%95%B0/2153250" \t "https://baike.baidu.com/item/%E6%9C%80%E5%A4%A7%E5%9B%A2%E9%97%AE%E9%A2%98/_blank)较大的问题，因此适用于解决最大团问题。

显然，我们可以将无向图的<G>看作解空间树，其解空间树为子集树，可使用子集选取的思想，设当前拓展节点z位于解空间树的第 i层，在进入其左子树前，必须确认从顶点i到已选入的顶点集中每个顶点都有边相连；在进入右子树前，必须确认还有足够多的可选择顶点使得算法有可能在右子树中找到更大的团。

# 2 问题分析

既然已经确定问题解的方向与算法，我们可以开始分析构建。显然，其解空间树为子集树，我们使用了子集选取的思想，由此开始分析回溯时的约束函数与限界函数：算法约束函数(constraint())：准备深刻左子树时，判断当前节点是否能够加入团顶点集合；数组限界函数(bound())：准备深刻右子树时，判断是否仍有可能产生最优解。app在具体实现时，咱们使用临界矩阵表示图G，用整型数组x[]来表示是否选择进入最大团。

# 3 算法设计

我设计用回溯算法解决最大团问题得到问题的最优解。回溯法是一个既带有系统性又带有跳跃性的的搜索算法，它是一种系统地搜索问题的解的方法。它适用于解一些组合数较大的问题,通过分析用回溯算法解决最大团问题能得到问题的最优解。

解向量：X{x1,x2,…,xn}(xi∈{0,1})

解空间结构：子集树。

其中回溯法是采用迭代回溯，先秒速出需要实现各种功能的函数。

迭代的基本框架：

|  |  |  |
| --- | --- | --- |
| 约束函数constraint() | bool constraint(int k) {  for(int j=1; j<k; j++) {  if(x[j] && a[j][k]==0) {  return false;  }  }  return true;  } | 对于待加入节点 k k k，若是存在某个已经在当前最大团集合的节点与其没有链接，那么就没法构成团，即返回false，不然返回true |
| 限界函数bound() | bool bound(int k) {  if(cn+n-k+1 > bestn) {  return true;  }  return false;  } | 对于带考虑节点 k k k，若是最大团集合内当前节点数cn加上剩下未考虑的节点数n-k+1的值严格优于当前最大团集合的节点最优值，则返回true，不然返回false |
| 迭代函数MCP() | void MCP() {  int k=1;  while(1) {  while(k<=n && constraint(k)) {  cn++;  x[k++]=1;    }  if(k>=n) { //记录解  if(cn>bestn) bestx.clear(); //一组更大的解，清空解向量集合  if(cn>=bestn){ //相等顶点数的解(更大的解也会落入此)  vector<int> chose;  for(int i=1;i<=n;i++){ //获得哪些点被选入团  if(x[i])chose.push\_back(i);  }  bestx.push\_back(chose); //加入到解向量  bestn = cn; //更新最优值  }  k=n; //在判断bound前，提早回溯  while(k && !x[k]) { //找到最后一个加入的节点  k--;  }  cn--; //移出团  x[k++]=0; //标记不选  } else {;  x[k++]=0;  }  while(!bound(k)) {  k--;  while(k && !x[k]) {  k--;  }  if(k==0)return;  cn--;  x[k++]=0;  }  }  } | 在记录解以后，判断bound()以前，修改掉其k值，从而避免刚刚记录bestn就卡住了；一种可行的想法是：在记录了解以后，先提早找到最后一个加入团的顶点，将其回溯，即将其从团集合中剔除，而后才进入bound()的判断，这样既能够达到预期的效果。 |
| bound() | bool bound(int k) {  if(cn+n-k+1 >= bestn) { //添加等于的状况  return true;  }  return false;  } | 试图获得多组解 |

4 算法实现

#include <iostream>

#include <vector>

using namespace std;

#define MAX 2000

int n,m;

int a[MAX+1][MAX+1];

int cn=0,bestn=0;

int x[MAX+1];//bestx[MAX+1];

vector< vector<int> > bestx;

bool constraint(int k) {

for(int j=1; j<k; j++) {

if(x[j] && a[j][k]==0) {

return false;

}

}

return true;

}

bool bound(int k) {

if(cn+n-k+1 >= bestn) {

return true;

}

return false;

}

void output() {

for(int i=0;i<bestx.size();i++){

for(int j=0;j<bestx[i].size();j++){

cout<<bestx[i][j]<<" ";

}

cout<<endl;

}

}

void MCP() {

int k=1;

while(1) {

while(k<=n && constraint(k)) {

cn++;

//cout<<"第"<<k<<"个知足约束条件cn="<<cn<<endl;

x[k++]=1;

}

if(k>=n) {

if(cn>bestn) bestx.clear();

if(cn>=bestn){

vector<int> chose;

for(int i=1;i<=n;i++){

if(x[i])chose.push\_back(i);

}

cout<<endl;

bestx.push\_back(chose);

bestn = cn;

}

k=n;

while(k && !x[k]) {

k--;

}

cn--;

x[k++]=0;

} else {

//cout<<"第"<<k<<"个不知足约束条件"<<endl;

x[k++]=0;

}

while(!bound(k)) {

//cout<<"第"<<k<<"个节点不知足限界cn="<<cn<<endl;

k--;

while(k && !x[k]) {

k--;

}

if(k==0)return;

cn--;

//cout<<"由于不知足约束也不知足限界而回溯至第"<<k<<"个节点cn="<<cn<<endl;

x[k++]=0;

}

}

}

int main() {

cout<<"请输入该图的顶点数V与边数E："<<endl;

cin>>n>>m;

int u,v;

cout<<"请依次输入边的顶点u和v："<<endl;

for(int i=1; i<=m; i++) {

cin>>u>>v;

a[u][v]=1;

a[v][u]=1;

}

MCP();

cout<<"此无向图的最大团的解有以下："<<endl;

output();

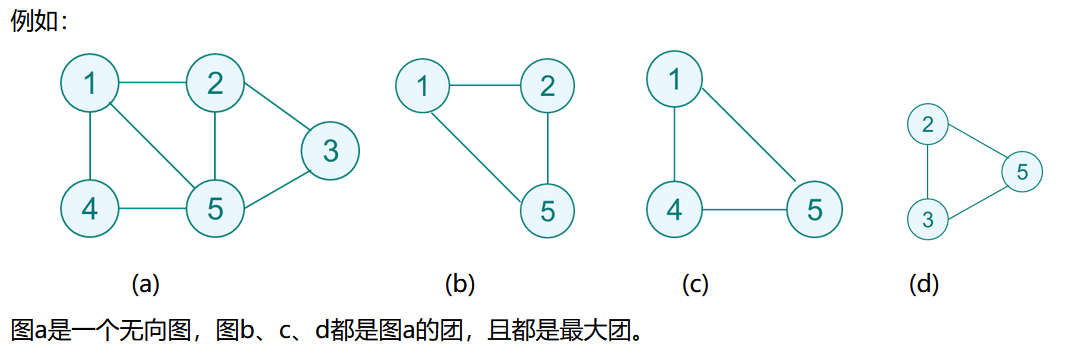
}

//注释掉的为调试信息。

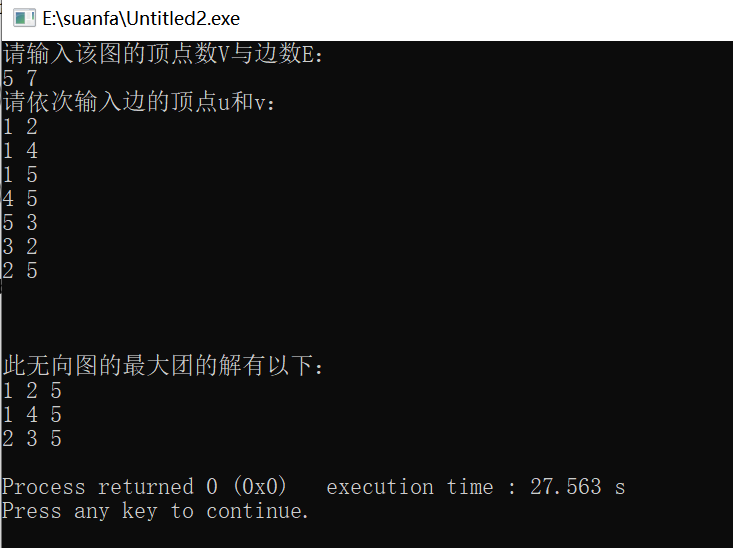
5 测试分析

1、程序测试

通过运行，先测试一个简单的无向图：

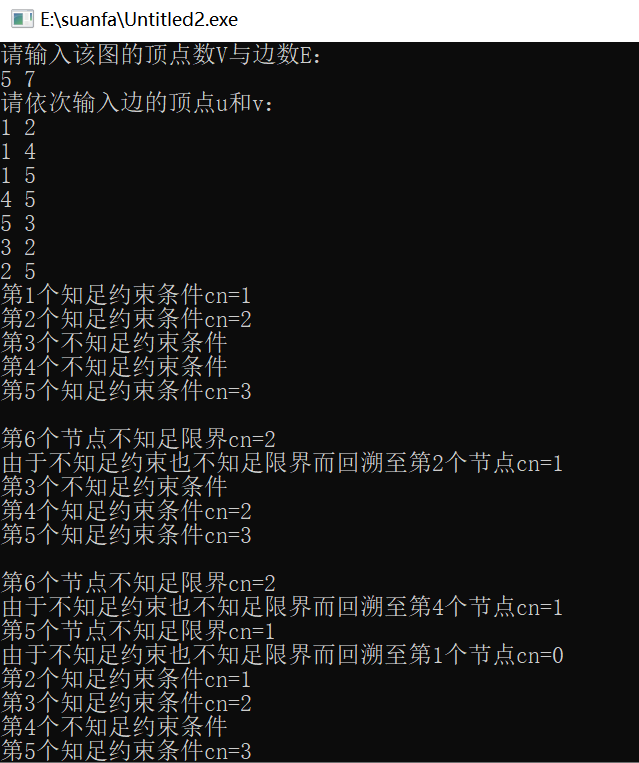
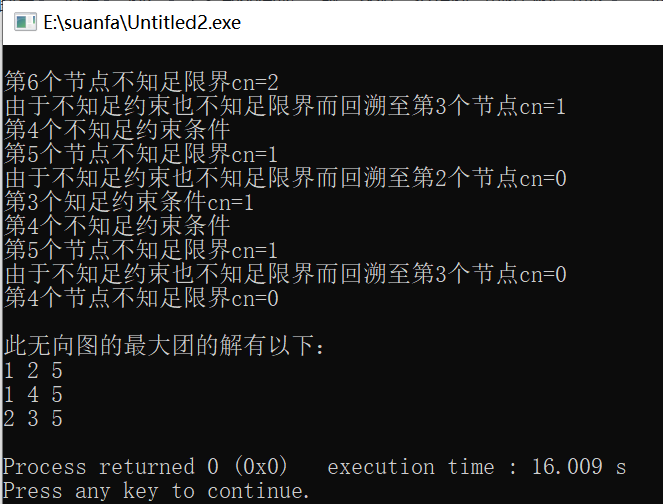


运行结果：



可以看出运行得出的结果正确。

查看调试步骤：



在程序里面我定义了max=2000，原则上就是可以处理顶点数<=2000和边数<=2000的无向图。

2、时间复杂度分析

有个顶点须要考虑，每一个节点都都有深刻左右子树两种选择，而解空间树高为，故至多有种可能状况，所以T ( n ) = n \*T(n)=n\*2n，故时间复杂度为： O ( n ) O(n\*2n)。

3、设计实现中出现的问题

（1）每次求解相关无向图都需要手动输入顶点数和边数，以及每一条边的两个顶点，很繁琐。顶点数与边数不多时还好，但是顶点数与边数多起来之后，手动输入不仅慢，且准确性低，一旦输错并没有提示，而是需要结束程序重新输入。（这是可以改进算法的地方，当发现中途输错了之后，可以通过按什么键来返回更改哪一行的边的顶点数）；

（2）通过调试分析，发现此程序中的数组变化是一个难点，其中的数值变化是整个课程设计的关键，或多或少有点繁琐，不容易掌握清楚；

（3）对一些特殊情况没有考虑到，从而造成设计出来的程序不全面，如在输入图中顶点信息时，没有考虑到输入的顶点不存在的情况；

（5）如果数据之间的类型不同，会引发数据间交换紊乱。有些语法错误在调试中引发乱码，而且会频繁出现，因此需要多调试，增加了时间成本。

结 论

本课程设计研究回溯算法来解决最大团问题，因为递归回溯算法已经很多人尝试过，因此我尝试的是迭代回溯。解决了最大团问题的方法只适合某些特殊的图，活则消耗时间长，求解效率低等问题，不必再担心得到最优解难问题。根据算法的设计结果，采用C语言实现算法，通过测试分析，程序运行结果正确，运用程序可以通过输入顶点个数、边的个数，每条边的信息得到结果，方便快捷，运行效率较高。

同时，在课程设计过程中还存在很多问题：

对数据结构和C语言的生疏；此程序中的数值变化或多或少有点繁琐，不容易掌握清楚；对一些特殊情况没有考虑到，从而造成设计出来的程序不全面，如在输入图中顶点信息时，没有考虑到输入的顶点不存在的情况；如果数据之间的类型不同，会引发数据间交换紊乱；有些语法错误在调试中引发乱码，而且会频繁出现；在回溯算法的效率上没有更多的关注，设定的顶点和边数量给定不能更改，同时在输入界面上没美化，信息提示不够完善等问题，这些隐在的问题都会使用户在使用时出现错误而使程序无法继续运行。因此要多调试，思以解决这些问题。