用c语言实现有限大电路图的精确求解及拓展

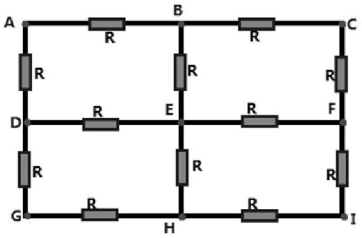
张超、谢佳豪

一、引言

两幅电路图：

图1：求E,G之间的电阻

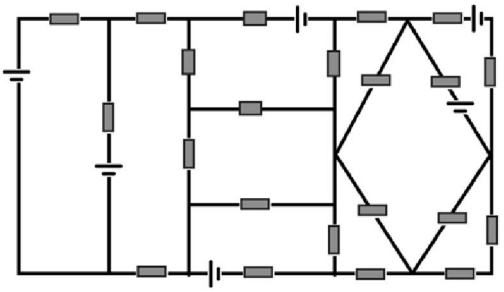
对称性极好，有简便方法



- 图2：求解各路径电流

对称性差，基本只能用基尔霍夫定律求解

而现实生活中更多的问题是对非对称电路图的求解，也正是基于此，我们便想通过编一个通用的C语言程序对该类问题进行求解。



各处阻值不等

二、算法描述

对于一个已知的电路（先讨论没有电源的情况），求解步骤如下：

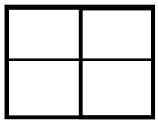
- 存储电路图
- 寻找电路图中的回路
- 根据电路回路构造出方程组
- 用crammer法则求解方程组

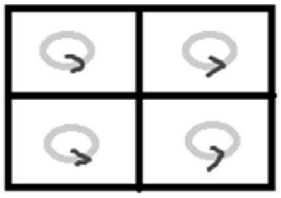
存储电路图

- **利用“图”的结构存储**；把电路图简单看做很多个点，每两个点之间有一条线，线伴有一个权值，即为电阻（详细参见“数据结构与数据库”）
- **对每个点标记**；如a与b点之间电阻为m；具体可根据用户需要存储电路图。

寻找回路（关键点）

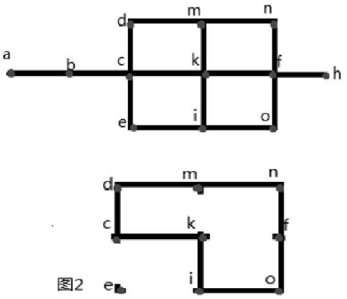
- **代码存储的是零散的点与线，无法识别回路和孔洞；**现实运算中我们只需要根据每个最小孔洞列出回路方程即可。例如对于"田"字方格，孔洞如图；
- 考虑到如果根据所有的回路可以**画出原来的电路图**，（如图）说明找到的所有回路基本保留了电路图所有的信息。





算法1：

任意选取**两个点**，找出这两个点之间**所有**的的**连通路径**，显然找到的只是半回路，组合起来的回路不能保证方程独立。(失败)



算法2：

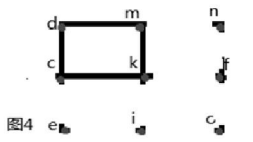
1去除所有**孤立的边**，如图中ab，bc，fh均是孤立边，要求与它们相关的电阻比较简单，可以额外考虑。

2.再在非孤立的边构成的图中**任意**寻找一条**闭合回路**，不妨寻找为cdmnfoikc(图2)

3.这时存在这样一种**公共点**，既有走过了的边又有没走过的边，如m,c,i,k,f;通过这些点**任意**选取**两个点**，如m,f

4、在已经**画好的图**中（图2）寻找通过m,f的**半回路**：如mnf;然后在**未画好的图**中（图3）寻找**半回路**；如f,k,m。但是当经过了**三个公共点**(k也是公共点)，需**更换**两个公共点，如m,k。对应回路mdck和km拼接起来得到第二条回路（图4）；重复上述过程**直到没有公共点**即找完所有的回路。

- 回路：cdmnfoikc　mdckm　foikf　iecki



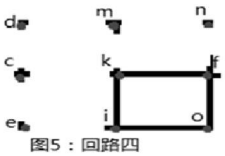
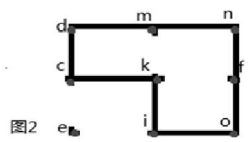
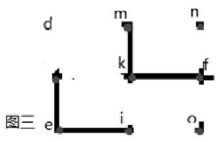


图5：回路四



已画好的图



未画好的图

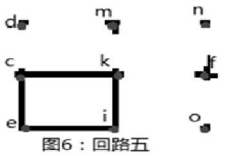
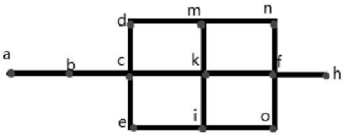
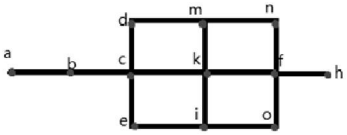


图6：回路五

根据回路构造方程组



- **在用户输入想要求电阻的两点添加一个电源；**可知这两点和这个电源就是一个很好的**半回路**，然后在原电路图中任意找到一条连接两点的**半回路**，拼接起来即是一条**回路**；
- **根据回路判断通过每条边的电流；**对找到的回路标记1,2,3,4,5；构造一个二维数组（用来判断通过每条边的电流），根据回路，如iecki,就在ie边对应的数组行里添加一个+4（第4个回路，选取i到e为正，反之为负），遍历完所有的回路这样每条边都标记上了走过的电流。



- **根据路径电流和回路构造矩阵；**之后构造一个5*6（每一列代表一个未知数电流）的数组（矩阵）（有5个回路），遍历每个回路，[illegible]系数存在每一行里）；如mdckm，[illegible]和-2；就在第2行（回路2）第[illegible]存入cd阻值的负值；如此重复，访[illegible]第5行第6列存入阻值u（内部生成）；[illegible]

过程完全仿造手工操作过程，太繁琐了，略去

方程组求解

-

三、程序内容（可见附件程序）

```
#include <stdio.h>
#include <stdlib.h>
#include<string.h>
#define max_n 20
#define max_float 1000.0

typedef float adjmatrix[max_n][max_n];
typedef struct{
    int vexnum;             //顶点数,
    int arcnum;             //边数
    char vexs[max_n];       //顶点标记
    adjmatrix arcs;         //邻接矩阵
    char adjarc[2][max_n];  //存储边的两个顶点，用int存储，即我用0,1,2,3来存储；可能有没考虑周到的地方
    int state[max_n];       //存储边的状态
    float resist[max_n];    //边的阻值
    char circle[max_n][max_n];   //回路,第一位都用来存该回路的长度，第length位是没用的,同adjarc的存储方式
    int cirnum;             //回路数
}mgraph;
typedef struct{
    float mat[max_n][max_n];  //方阵
    int dim;                  //方阵的行数
}matrix;


void findcircle(mgraph &p);
//寻找回路的初级函数
void printatoi(char *aa,int length);
//打印字符串
void printgraph(mgraph p);
```

break

```
void printgraph(mgraph p);
//打印整个存储结构
int graphfind(mgraph p,int vi,int vj);
//寻找vi vj对应的电阻下标
int firstadjvex(mgraph p,int vi);
//找到vi的第一个邻接点
int nextadjvex(mgraph p,int vi,int vj);
//找到vi的所有邻接点中在vj的下一个；
int degree(mgraph p,int vi);
//对vi度的判断
void vexjudge(mgraph &p);
//对度为1的顶点进行判断并将相关的边的state存为-1
void circle4(mgraph &p,int vi,int vj,char *a,int &length,char *ac);
//a先申请空间，初始好，a的第一个空间放1表示还未构成回路，2表示已经构成会路
//转存时注意第一个2不存入circle   寻找已有回路中的半回路 length用来记录a的长度a[length]一定是没用的空间
//第一次调用这个函数时a的第一个空间应该初始好，length初始为1,ac用来做参照   判断是否访问过
//这个专门用来找干路半回路的
void circle3(mgraph &p,int vi,int vj,char *a,int &length,char *ac);
//a先申请空间，初始好，a的第一个空间放1表示还未构成回路，2表示已经构成会路
//转存时注意第一个2不存入circle   寻找未构成回路中的半回路 length用来记录a的长度a[length]一定是没用的空间
//第一次调用这个函数时a的第一个空间应该初始好，length初始为1,ac用来做参照
void circle2(mgraph &p,int vi,int vj,char *a,int &length,char *ac);
//a先申请空间，初始好，a的第一个空间放1表示还未构成回路，2表示已经构成会路
//转存时注意第一个2不存入circle   寻找已有回路中的半回路 length用来记录a的长度a[length]一定是没用的空间
//第一次调用这个函数时a的第一个空间应该初始好，length初始为1,ac用来做参照   判断是否访问过
void circle1(mgraph &p,int vi,int vj,char *a,int &length);
//a先申请空间，初始好，a的第一个空间放1表示还未构成回路，2表示已经构成会路
//转存时注意第一个2不存入circle   寻找第一个回路 length用来记录a的长度a[length]一定是没用的空间
//第一次调用这个函数时a的第一个空间应该初始好，length初始为1
```

break

```
void initation(mgraph &p);
//对图初始化
void create1(mgraph &p);
//创建
void state2(mgraph p,int **&ab);
//建立一个临时的参考数组
void findcircle(mgraph &p);
//寻找回路的初级函数
float detmatrix(matrix p);
//求矩阵的行列式
float det(matrix p,int ia);
//p保存了方阵，ia即表示要求的回路电流，又是要删去的列
void makecurrent(mgraph &p,char current[][max_n]);
//由回路判断每个回路的电流
void domatrix(mgraph &p,char current[][max_n],matrix &deter,float u);
//将回路转化为方阵 u是电压
void makematrix(mgraph &p);
//构造矩阵
void UI();
//视图界面


void printatoi(char *aa,int length){
//打印字符串
    int a;
    for(a=0;a<length;a++){
        printf("%d",*(aa+a));
    }
    printf("\n");
```

break

```
}//printatoi

void printgraph(mgraph p){
//打印邻接矩阵
    int a,b;
    printf("顶点数：%d\t边数：%d \t回路数：%d\n",p.vexnum,p.arcnum,p.cirnum);
    printf("顶点标记：%s\n邻接矩阵：\n",p.vexs);
    for(a=0;a<p.vexnum;a++){
        for(b=0;b<p.vexnum;b++){
            if(p.arcs[a][b]==max_float)
                printf("NULL\t");
            else    printf("%.3f\t",p.arcs[a][b]);
        }
        printf("\n");
    }
    printf("结点，结点，阻值，状态\n");
    for(a=0;a<p.arcnum;a++){
        printf("<%d,%d,%.3f,%d>\t",p.adjarc[0][a],p.adjarc[1][a],p.resist[a],p.state[a]);
    }
    printf("\n回路:\n");
    for(a=0;a<p.cirnum;a++){
        printatoi(p.circle[a]+1,p.circle[a][0]-1);
    }
    system("pause");
    system("cls");
}//printgraph

int graphfind(mgraph p,int vi,int vj){
//寻找vi vj对应的由阴下标
```

break

```
    int a;
    for(a=0;a<p.arcnum;a++){
        if(p.adjarc[0][a]==vi&&p.adjarc[1][a]==vj)
            return a;
        if(p.adjarc[1][a]==vi&&p.adjarc[0][a]==vj)
            return a;
    }
    return -1;            //暂时没做对-1报错的处理，算是系统漏洞了
}//grephfind

int firstadjvex(mgraph p,int vi){
//找到vi的第一个邻接点
    int a;
    for(a=0;a<p.vexnum;a++){
        if(p.arcs[vi][a]!=max_float)
            return a;
    }
    return -1;     //暂时没做对-1报错的处理，算是系统漏洞了
}//firstadjvex

int nextadjvex(mgraph p,int vi,int vj){
//找到vi的所有邻接点中在vj的下一个；
    int a;
    for(a=vj+1;a<p.vexnum;a++){
        if(p.arcs[vi][a]!=max_float)
            return a;
    }
    return -1;     //暂时没做对-1报错的处理，算是系统漏洞了
}//nextadjvex
```

break

```
int degree(mgraph p,int vi){
//对vi度的判断
    int a,b;
    for(a=0,b=0;a<p.vexnum;a++){
        if(p.arcs[vi][a]!=max_float)
            b++;
    }
    return b;
}//degree

void vexjudge(mgraph &p){
//对度为1的顶点进行判断并将相关的边的state存为-1
    int a,b,c,d;
    for(a=0;a<p.vexnum;a++){
        if(degree(p,a)==1){
            b=firstadjvex(p,a);
            c=graphfind(p,a,b);
            p.state[c]=-1;
            for(;degree(p,b)<=2;){
                if(degree(p,b)==1)
                    break;
                if(degree(p,b)==2){
                    c=firstadjvex(p,b);
                    d=graphfind(p,b,c);
                    if(p.state[d]==-1){
                        c=nextadjvex(p,b,c);
                        d=graphfind(p,b,c);
                        if(p.state[d]==-1) break;
                    }
```

break

```
                    p.state[d]=-1;
                    b=c;
                }
            }
        }
    }
}//vexjudge

void circle4(mgraph &p,int vi,int vj,char *a,int &length,char *ac){
//a先申请空间，初始好，a的第一个空间放1表示还未构成回路，2表示已经构成会路
//转存时注意第一个2不存入circle   寻找已有回路中的半回路 length用来记录a的长度a[length]一定是没用的空间
//第一次调用这个函数时a的第一个空间应该初始好，length初始为1,ac用来做参照  判断是否访问过
//这个专门用来找干路半回路的
    int w,i;
    if(*a==2) return;
    a[length]=vi;    //这里将int直接当char存了
    ac[vi]=1;
    length++;
    if(vi==vj&&length>2){
        *a=2;
        return;
    }
    for(w=firstadjvex(p,vi);*a==1&&w!=-1;w=nextadjvex(p,vi,w)){
        i=graphfind(p,vi,w);
        if(ac[w]==0){
            circle4(p,w,vj,a,length,ac);
            if(*a!=2)
                a[length--]=0;
        }
```

break

```
    }
}//circle4

void circle3(mgraph &p,int vi,int vj,char *a,int &length,char *ac){
//a先申请空间，初始好，a的第一个空间放1表示还未构成回路，2表示已经构成会路
//转存时注意第一个2不存入circle   寻找未构成回路中的半回路 length用来记录a的长度a[length]一定是没用的空间
//第一次调用这个函数时a的第一个空间应该初始好，length初始为1,ac用来做参照
    int w,i;
    if(*a==2) return;
    a[length]=vi;    //这里将int直接当char存了
    ac[vi]=1;
    length++;
    if(vi==vj&&length>2){
        *a=2;
        return;
    }
    for(w=firstadjvex(p,vi);*a==1&&w!=-1;w=nextadjvex(p,vi,w)){
        i=graphfind(p,vi,w);
        if(p.state[i]==0&&ac[w]==0){
            circle3(p,w,vj,a,length,ac);
            p.state[i]=1;
            if(*a!=2){
                a[length--]=0;
                p.state[i]=0;
            }
        }
    }
}//circle3
```

break

```
void circle2(mgraph &p,int vi,int vj,char *a,int &length,char *ac){
//a先申请空间，初始好，a的第一个空间放1表示还未构成回路，2表示已经构成会路
//转存时注意第一个2不存入circle   寻找已有回路中的半回路 length用来记录a的长度a[length]一定是没用的空间
//第一次调用这个函数时a的第一个空间应该初始好，length初始为1,ac用来做参照   判断是否访问过
    int w,i;
    if(*a==2) return;
    a[length]=vi;    //这里将int直接当char存了
    ac[vi]=1;
    length++;
    if(vi==vj&&length>2){
        *a=2;
        return;
    }
    for(w=firstadjvex(p,vi);*a==1&&w!=-1;w=nextadjvex(p,vi,w)){
        i=graphfind(p,vi,w);
        if(p.state[i]==1&&ac[w]==0){
            circle2(p,w,vj,a,length,ac);
            if(*a!=2)
                a[length--]=0;
        }
    }
}//circle2

void circle1(mgraph &p,int vi,int vj,char *a,int &length){
//a先申请空间，初始好，a的第一个空间放1表示还未构成回路，2表示已经构成会路
//转存时注意第一个2不存入circle   寻找第一个回路 length用来记录a的长度a[length]一定是没用的空间
//第一次调用这个函数时a的第一个空间应该初始好，length初始为1
    int w,i;
    if(*a==2) return;
```

break

```
    a[length]=vi;    //这里将int直接当char存了
    length++;
    if(vi==vj&&length>2){
        *a=2;
        return;
    }
    for(w=firstadjvex(p,vi);*a==1&&w!=-1;w=nextadjvex(p,vi,w)){
        i=graphfind(p,vi,w);

        if(p.state[i]==0){
            p.state[i]=1;
            circle1(p,w,vj,a,length);
            if(*a!=2)
                a[length--]=0;
        }
    }
}//circle1



void initation(mgraph &p){
//对图初始化
    int a,b;
    p.vexnum=p.arcnum=p.cirnum=0;      //对三个数初始为0
    for(a=0;a<max_n;a++){
        p.vexs[a]='\0';                //字母标记初始为0
        p.state[a]=0;                  //状态值初始为零
        p.adjarc[0][a]=p.adjarc[1][a]=max_n;   //边顶点初始为max_n,一个不可能使用得到的数
```

break

```
        p.resist[a]=max_float;                    //阻值初始为max_float
    }
    for(a=0;a<max_n;a++){
        for(b=0;b<max_n;b++){
            p.arcs[a][b]=max_float;     //邻接矩阵全初始为max_float
            p.circle[a][b]=max_n;           //回路各点初始为max_n,一个不可能使用得到的数
        }
    }
}//initation

void create1(mgraph &p){
//创建
    int a,b,c,d=0,e;
    float i;
    char aa,bb;
    printf("please input the vexum,the arcnum\n");  //默认无向图处理
    scanf("%d%d",&p.vexnum,&p.arcnum);      //前者定点数 后者边数
    printf("%d\t",p.vexnum );
    printf("please input the vexs,like abcdefgh...\n");
    for(a=0;a<1;a++){
        scanf("%s",p.vexs);
        if((int)strlen(p.vexs)!=p.vexnum ){
            printf("error,please input again\n");
            for(a=0;a<max_n;a++){
                p.vexs[a]='\0';                //字母标记重新初始为0
            }
            a--;
        }
    }
```

break

```
    printf("please input the arcs,like a,b,4\n");
    puts(p.vexs);
    for(a=0;a<p.arcnum;a++){
        scanf("\n%c,%c,%f",&aa,&bb,&i);
        for(b=0;b<p.vexnum;b++){
            if(p.vexs[b]==aa) break;
        }
        for(c=0;c<p.vexnum;c++){
            if(p.vexs[c]==bb) break;
        }
        if(b==p.vexnum||c==p.vexnum){
            printf("input error,please input again\n");
            a--;continue;
        }
        p.arcs[b][c]=i;
        p.arcs[c][b]=i;
        e=graphfind(p,b,c);
        if(e==-1){
            p.adjarc[0][a]=b;
            p.adjarc[1][a]=c;           //b,c是标志字母对应的下标,这里将int当char存了
            p.resist[a]=i;
        }
        else{
            p.adjarc[0][e]=b;
            p.adjarc[1][e]=c;           //b,c是标志字母对应的下标,这里将int当char存了
            p.resist[e]=i;
        }
    }
    system("cls");
```

break

```
}//create1

void ininit(mgraph &p){
//内部初始化
    p.arcnum=7;p.vexnum=6;
    p.vexs[0]='a';p.vexs[1]='b';p.vexs[2]='c';p.vexs[3]='d';p.vexs[4]='e';p.vexs[5]='f';
    p.arcs[0][1]=p.arcs[1][0]=p.resist[0]=1;p.adjarc[0][0]=0;p.adjarc[1][0]=1;
    p.arcs[1][2]=p.arcs[2][1]=p.resist[1]=2;p.adjarc[0][1]=1;p.adjarc[1][1]=2;
    p.arcs[1][3]=p.arcs[3][1]=p.resist[2]=3;p.adjarc[0][2]=1;p.adjarc[1][2]=3;
    p.arcs[2][3]=p.arcs[3][2]=p.resist[3]=4;p.adjarc[0][3]=2;p.adjarc[1][3]=3;
    p.arcs[2][4]=p.arcs[4][2]=p.resist[4]=5;p.adjarc[0][4]=2;p.adjarc[1][4]=4;
    p.arcs[3][4]=p.arcs[4][3]=p.resist[5]=6;p.adjarc[0][5]=3;p.adjarc[1][5]=4;
    p.arcs[4][5]=p.arcs[5][4]=p.resist[6]=7;p.adjarc[0][6]=4;p.adjarc[1][6]=5;
}//ininit

void creategraph(mgraph &p){
//创建图,同时将回路找好
    int a;
    initation(p);
    printf("1:新电路图; 2:内部电路图\n");
    scanf("%d",&a);
    switch(a){
    case 1:
        create1(p);
        break;
    case 2:
        ininit(p);
        break;
    }
```

break

```
    printgraph(p);
    vexjudge(p);        //对度为1的顶点进行判断并将相关的边的state存为-1
    findcircle(p);
    printgraph(p);
}  //creategraph

void state2(mgraph p,int **&ab){
//建立一个临时的参考数组
    int a,b,c;
    ab=(int**)malloc(2*sizeof(int*));
    *ab=(int*)malloc(sizeof(int)*(p.vexnum));
    for(a=0;a<p.vexnum;a++){
        *(*ab+a)=0;
    }
    *(ab+1)=(int*)malloc(sizeof(int)*(p.vexnum));
    for(a=0;a<p.vexnum;a++){
        *(*(ab+1)+a)=0;
    }
    for(a=0;a<p.arcnum;a++){
        if(p.state[a]==0){
            b=p.adjarc[0][a];
            c=p.adjarc[1][a];
            *(*(ab+1)+b)=1;
            *(*(ab+1)+c)=1;
        }                   //未被访问过的路线的两个顶点标记在第二行，下标为1；均标记为1
        if(p.state[a]==1){
            b=p.adjarc[0][a];
            c=p.adjarc[1][a];
            *(*ab+b)=1;
```

break

```
        }                   //被访问过的路线的两个顶点标记在第一行，下标为0；均标记为1
    }
}//state2

void findcircle(mgraph &p){
//寻找回路的初级函数
    char *aa,*ac;
    int **ab;
    int a,b,c,d,length=1,e,f;
    aa=(char*)malloc(sizeof(char)*(p.vexnum));    //aa用来存放临时回路的
    *aa=1;                                        //接下来三行都是用来初始化aa的
    for(a=1;a<p.vexnum;a++){
        aa[a]='\0';
    }
    for(a=0;a<p.arcnum;a++){
        if(p.state[a]!=-1) break;
    }
    circle1(p,a,a,aa,length);
    p.circle[0][0]=length;              //回路中第一个存储的是长度，打印时考虑加减1
    for(a=1;a<length;a++){
        p.circle[0][a]=aa[a];
    }
    p.cirnum++;
    while(1){
        *aa=1;length=1;                                   //初始化
        for(a=1;a<p.vexnum;a++){
            aa[a]='\0';
        }
        state2(p,ab);
```

break

```
for(a=0;a<p.vexnum;a++){
    if(*(*ab+a)==1&&*(*(ab+1)+a)==1)
        break;
}
for(b=a+1;b<p.vexnum;b++){
    if(*(*ab+b)==1&&*(*(ab+1)+b)==1)
        break;
}
if(b>=p.vexnum){
    break;
}
ac=(char*)malloc(p.vexnum*sizeof(char));
for(c=0;c<p.vexnum;c++){
    ac[c]='\0';
}
circle2(p,a,b,aa,length,ac);
c=p.cirnum;
p.circle[c][0]=length;
for(e=1;e<length;e++){
    p.circle[c][e]=aa[e];
}
d=length-1;
*aa=1;length=1;                                    //初始化
for(e=1;e<p.vexnum;e++){
    aa[e]='\0';
    ac[e]=0;
}
circle3(p,b,a,aa,length,ac);
for(f=1;f<length;f++){
```

break

```
            p.circle[c][d]=aa[f];
            d++;
        }
        p.circle[c][0]=d;
        p.cirnum++;
    }
}

float detmatrix(matrix p){
//求矩阵的行列式
    matrix q;
    int i=0,j=0,x=0,c=0,d=0;
    float sum=0;
    for(i=0;i<max_n;i++){
        for(j=0;j<max_n;j++){
            q.mat[i][j]=0;
        }
    }
    q.dim=p.dim-1;
    if(p.dim==1)
        return p.mat[0][0];
    for(i=0;i<p.dim;i++){
        for(c=0;c<p.dim-1;c++){
            for(j=0;j<p.dim-1;j++){
                if(c<i){
                    d=0;
                }
                else{
                    d=1;
```

break

```
                }
                q.mat[c][j]=p.mat[c+d][j+1];
            }
        }
        if(i%2==0) x=1;
        else x=-1;
        sum+=p.mat[i][0]*detmatrix(q)*(x);
    }
    return sum;
}//detmatrix

float det(matrix p,int ia){
//p保存了方阵，ia即表示要求的回路电流，又是要删去的列
    int ib,ic;
    ic=p.dim;
    for(ib=0;ib<ic;ib++){
        p.mat[ib][ia]=p.mat[ib][ic];
        p.mat[ib][ic]=0;
    }
    return (detmatrix(p));
}//det

void makecurrent(mgraph &p,char current[][max_n]){
//由回路判断每个回路的电流
    int ia,ib,ic,id;
    for(ia=0;ia<p.cirnum;ia++){
        for(ib=1,ic=2;ic<p.circle[ia][0];ib++,ic++){
            id=graphfind(p,p.circle[ia][ib],p.circle[ia][ic]);
            if(p.adjarc[0][id]==p.circle[ia][ib])
```

break

```
                current[id][ia]=1;              //id是电阻的下标，ia是电流的下标
            else current[id][ia]=-1;
        }
    }
}//makecurrent

void domatrix(mgraph &p,char current[][max_n],matrix &deter,float u){
//将回路转化为方阵 u是电压
    int ia,ib,ic,id,ie,ig;
    deter.dim=p.cirnum;
    for(ia=0;ia<p.cirnum;ia++){
        for(ib=1,ic=2;ic<p.circle[ia][0];ib++,ic++){
            id=graphfind(p,p.circle[ia][ib],p.circle[ia][ic]);
            if(p.adjarc[0][id]==p.circle[ia][ib])
                ie=1;
            else ie=-1;
            for(ig=0;ig<p.cirnum;ig++){
                deter.mat[ia][ig]+=ie*(current[id][ig])*p.resist[id];
            }
        }
    }
    deter.mat[deter.dim-1][deter.dim]=u;
}//domatrix

void makematrix(mgraph &p){
//构造矩阵
    int ia,ib,ic,id,length;
    float fa,fb,u;
    char ca,cb;
```

break

```
matrix deter;
for(ia=0;ia<max_n;ia++){
    for(ib=0;ib<max_n;ib++){
        deter.mat[ia][ib]=0;
    }
}
deter.dim=0;                          //初始化方阵
printf("plese input the two nodes to be worked out the resist,like b,c\n");
for(ic=0;ic<1;ic++){
    scanf("\n%c,%c",&ca,&cb);       //最好能清空数据流
    for(ia=0;ia<p.vexnum;ia++){
        if(p.vexs[ia]==ca) break;
    }
    for(ib=0;ib<p.vexnum;ib++){
        if(p.vexs[ib]==cb) break;
    }
    if(ia==p.vexnum||ib==p.vexnum){
        printf("input error,please input again\n");
        ic--;continue;
    }
}
a=(char*)malloc(max_n*sizeof(char));
b=(char*)malloc(max_n*sizeof(char));
for(ic=0;ic<max_n;ic++){
    *(a+ic)=0;
    *(b+ic)=0;
}
length=1;*a=1;
circle4(p,ia,ib,a,length,b);   //a用来暂时保存回路，b只是一个参照作用
```

break

```
    id=p.cirnum;
    p.circle[id][0]=length;
    for(ic=1;ic<length;ic++){
        p.circle[id][ic]=a[ic];
    }
    p.cirnum++;                       //最后记得把这个回路删掉
    makecurrent(p,current);           //对过每个电阻的电流进行判断
    u=100;                            //电压预设为100好了
    domatrix(p,current,deter,u);
    fa=detmatrix(deter);              //存储系数矩阵的行列式；
    id=deter.dim;
    for(ic=0;ic<deter.dim;ic++){
        deter.mat[id][ic]=det(deter,ic)/fa;
    }                                  //每个环电流存在第p.dim行（非存储行）
    fb=u/deter.mat[id][id-1];
    printf("the risistance between %c and %c is:%f\n",ca,cb,fb);
    id=--p.cirnum;
    for(ic=0;ic<max_n;ic++){
        p.circle[id][ic]=0;
    }
    system("pause");
    system("cls");
}//makematrix



void UI(){
```

break

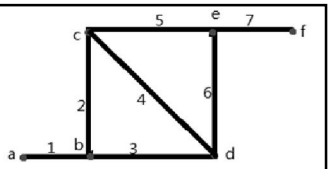
```
//视图界面
    mgraph p;
    int a=1;
    printf("\twelcome to use this application\n");
    system("pause");
    system("cls");
    for(a=1;a;){
        printf("=======================================\n");
        printf("1.建立电路图\t\t\t2.阻值求解\n");
        printf("3.修改电路图\t\t\t4.含多电源后的电路求解\t\n");
        printf("0.退出\n");
        printf("=======================================\n");
        scanf("%d",&a);
        switch(a){
        case 1:
            system("cls");
            creategraph(p);
            break;
        case 2:
            makematrix(p);
            break;
        case 3:
            printf("程序猿正在努力写代码中...\n");
            break;
        case 4:
            printf("程序猿正在努力写代码中...\n");
            break;
        case 0:
            break;
```

```
            }
        }
    }

    int main(){
        UI();
        printf("Thanks,goodbye.\n");
        system("pause");
        system("cls");
        return 0;
    }
```

break

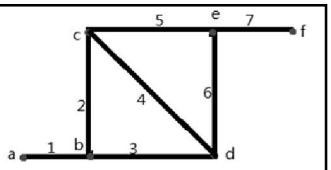
四、实例演示




```
=======================================
1.建立电路图                    2.阻值求解
3.修改电路图                    4.含多电源后的电路求解
0.退出
=======================================
1
```

break

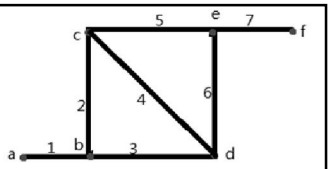
1、建立新电路图




```
1:新电路图; 2:内部电路图
1
```

break

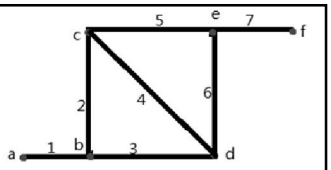
2、输入电路图中的顶点数与边数




```
1:新电路图; 2:内部电路图
1
please input the vexum,the arcnum
6 7
```

break

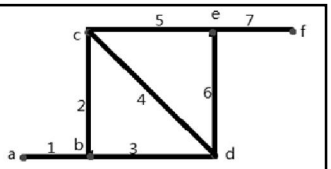
3、对电路图中各顶点编号




```
1:新电路图; 2:内部电路图
1
please input the vexum,the arcnum
6 7
6       please input the vexs,like abcdefgh...
abcdef
```

break

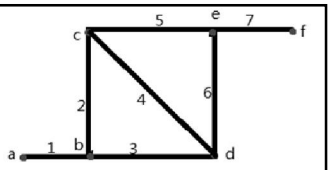
4、输入两点之间的阻值




```
1:新电路图; 2:内部电路图
1
please input the vexum,the arcnum
6 7
6       please input the vexs,like abcdefgh...
abcdef
please input the arcs,like a,b,4
abcdef
a,b,1
b,c,2
b,d,3
c,d,4
c,e,5
e,d,6
e,f,7
```

break

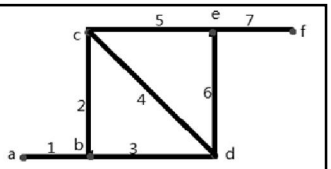
5、寻找回路前




```
顶点数：6       边数：7         回路数：0
顶点标记：abcdef
邻接矩阵：
NULL    1.000   NULL    NULL    NULL    NULL
1.000   NULL    2.000   3.000   NULL    NULL
NULL    2.000   NULL    4.000   5.000   NULL
NULL    3.000   4.000   NULL    6.000   NULL
NULL    NULL    5.000   6.000   NULL    7.000
NULL    NULL    NULL    NULL    7.000   NULL
结点，结点，阻值，状态
<0,1,1.000,0>   <1,2,2.000,0>   <1,3,3.000,0>   <2,3,4.000,0>   <2,4,5.000,0>
<4,3,6.000,0>   <4,5,7.000,0>
回路：
请按任意键继续. . .
```

break

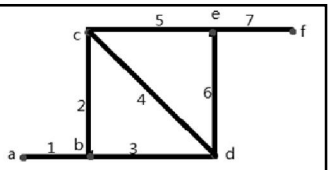
6、寻找回路后




```
顶点数：6       边数：7         回路数：2
顶点标记：abcdef
邻接矩阵：
NULL    1.000   NULL    NULL    NULL    NULL
1.000   NULL    2.000   3.000   NULL    NULL
NULL    2.000   NULL    4.000   5.000   NULL
NULL    3.000   4.000   NULL    6.000   NULL
NULL    NULL    5.000   6.000   NULL    7.000
NULL    NULL    NULL    NULL    7.000   NULL
结点，结点，阻值，状态
<0,1,1.000,-1>  <1,2,2.000,1>   <1,3,3.000,1>   <2,3,4.000,1>   <2,4,5.000,1>
<4,3,6.000,1>   <4,5,7.000,-1>
回路：
1231
21342
请按任意键继续. . .
```

break

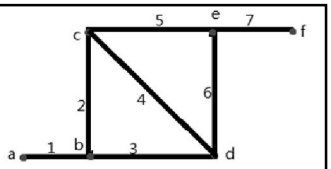
7、阻值求解




```
====================================
1.建立电路图                    2.阻值求解
3.修改电路图                    4.含多电源后的电路求解
0.退出
====================================
2
```

break

8、求a,f之间的阻值




```
====================================
1.建立电路图                    2.阻值求解
3.修改电路图                    4.含多电源后的电路求解
0.退出
====================================
2
plese input the two nodes to be worked out the resist,like b,c
a,f
the risistance between a and f is:11.932773
请按任意键继续. . .
```

break

五、算法评估

- 此算法适用范围：电路图对称性差，但电路图并不算太大的电路图

六、知识拓展

- 现在我们做到了给定有限大电路图（纯电阻电路）求任意两点的电阻，接下来我们来考虑如何对程序进行修改从而使其实现更多的功能。

1、若将电阻全部转换成电容器，求任意两点的电容。由于电容的并联与电阻的串联在数学运算上是等价的，所以我们可以将所有电容器的电容**取倒数**从而将其当做电阻进行运算。

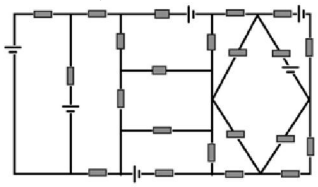
- 串联电路：$R=\sum R_i$　　$1/C=\sum 1/C_i$
- 并联电路：$1/R=\sum 1/R_i$　　$C=\sum C_i$

2、现在来考虑求解含电源的电路图：

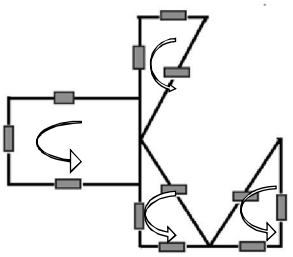
- 通过上述的算法，可知，对于最后电源的加入，只需要**增加一个含电源的回路**即可；
- 所以对于含有多电源的电路，可以先将**去除**电源的电路寻找回路，之后**再添加**与电源数目相等的含电源的回路方程。

- 优点：方程中含有的电源数尽可能的少。

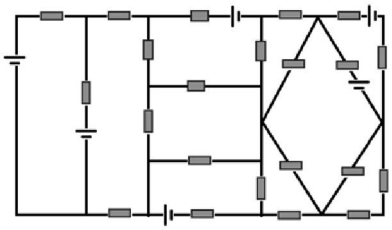
- 电路图2



- 去除电源，寻找回路



- 添加电源，再依次增加回路



3、**对无限大电路图阻值的逼近**；如果是无限大电路图，由于C语言**存储空间有限**以及无法将大量**数据难以手工输入**，因此我们考虑用**有限足够大**的电路对其进行**近似**处理。比如该电路图存在无限重复的某个电路基元（此时电路图应有良好的对称性），可以进行无限循环使原电路图加入该基元部分，令程序计算前后两次阻值之间的差别，规定当两次阻值之间的相对误差达到某个规定值时即跳出循环，以现在的阻值近似作为最后的阻值。（意义不太大）

4、使电源为交流电（将U存为一个t的函数），通过该程序即可计算出某个时刻某一支路中的电流情况，从而可以实时监控电流变化情况。

5、如果电路中含二极管，也可以实现所需功能，只需要对二极管进行条件[illegible]使用基尔霍夫定律时用的是正向穿过[illegible]没有达到击穿电压）。

这些功能具体取决于用户需要。

6、由于源程[illegible]并且给出电源参数（内阻，电动势）后[illegible]两边的电流从而也知道任意两边的电压，因此我们可以用此来快速判断大型电路图的故障位置。

7、可以判断电路等势点，方便简化电路。

谢谢！