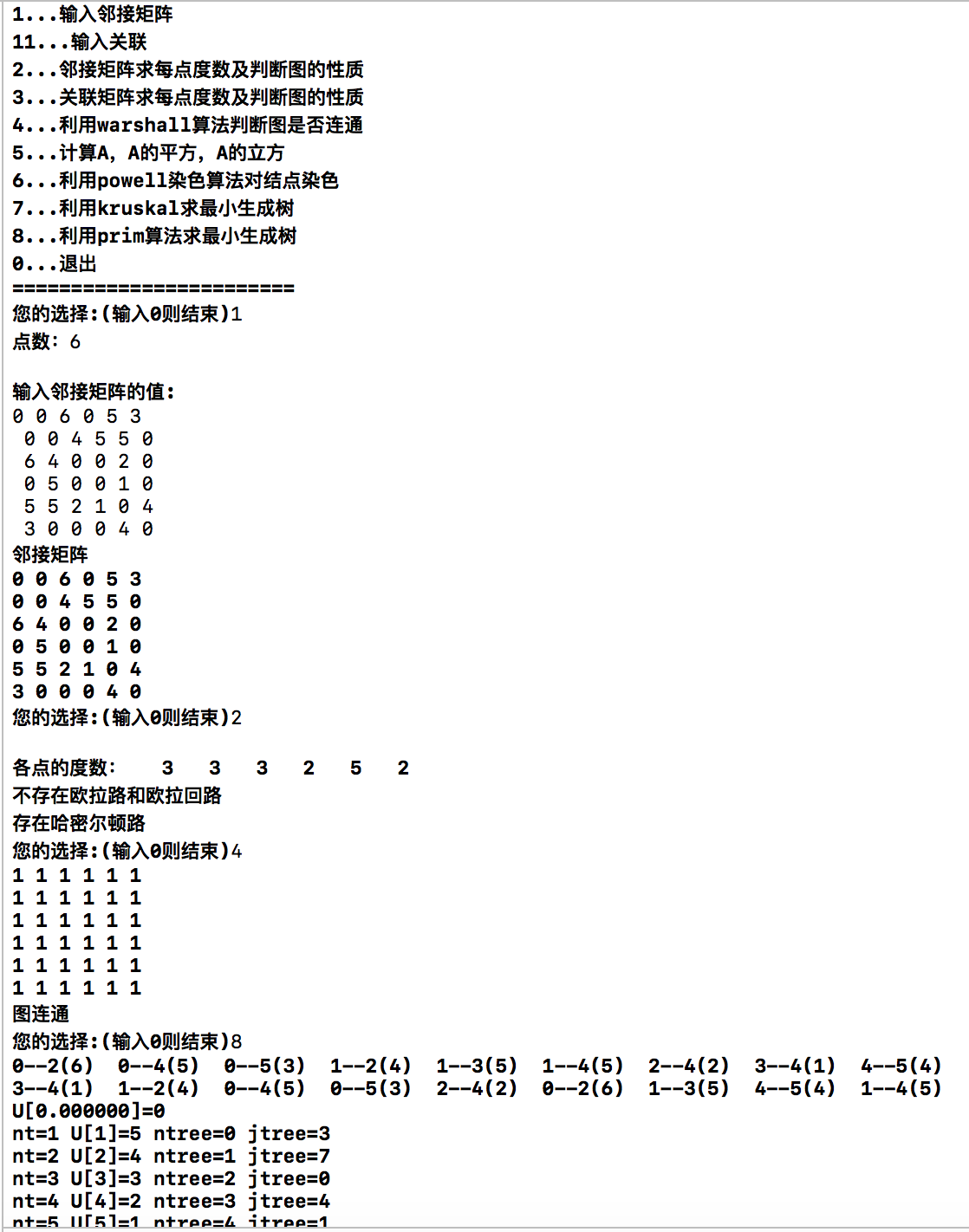
w2判断公式的合法性

1. 运行结果：



1. 代码改进与注释

// main.cpp

// 图论

//

// Created by zsy on 2018/6/10.

// Copyright © 2018年 zsy. All rights reserved.

//

#include<iostream>

#include<set>

#include<cstring>

using namespace std;

void printM(int m[100][100],int n,int p){

for(int i=0;i<n;i++){

for(int j=0;j<p;j++)

cout<<m[i][j]<<' ';

cout<<endl;

}

}

**把我的矩阵逐行打印**

void getMar(int M[100][100],int n,int m){

for(int i=0;i<n;i++){

for(int j=0;j<m;j++){

scanf("%d",&M[i][j]);

}

}

}

**接收我要输入的矩阵，逐行接收**

void matrixmulti(int M[100][100],int m[100][100],int T[100][100],int n){

for(int i=0;i<n;i++)

for(int j=0;j<n;j++)

for(int k=0;k<n;k++)

T[i][j]+=M[i][k]\*m[k][j];

}

**两个矩阵相乘用三个for循环即可达成**

void matrixplus(int M[100][100],int m[100][100],int n){

for(int i=0;i<n;i++)

for(int j=0;j<n;j++)

M[i][j]+=m[i][j];

}

**矩阵相加就直接加就可以了，两个for循环就ok**

void calDegree(int A[100][100],int n,int D[]){

for(int i=0;i<n;i++){

for(int j=0;j<n;j++){

D[i]+=(A[i][j]==0?0:1);

//D[j]+=(A[i][j]==0?0:1)\*\*\*

}

}

}

**求度数，遍历我这个数组，不为0的就代表有出度或者入度，就度数+1**

void isEular(int D[],int n){

int odd=0,iodd=0,jodd=0;

cout<<endl<<"各点的度数：";

for(int i=0;i<n;i++){

printf("%4d",D[i]);

if(D[i]%2==1){

odd++;

if(iodd==0) iodd=i;

else jodd=i;

}

}

**上面那个就是在算有多少个奇数度数的节点**

if(odd==0)

cout<<endl<<"存在欧拉回路，即为欧拉图"<<endl;

else if(odd==2)//

cout<<endl<<"存在欧拉路，起点为"<<iodd<<" 终点为"<<jodd<<endl;

else

cout<<endl<<"不存在欧拉路和欧拉回路"<<endl;

}

**欧拉图的条件：没有奇数度数的节点或者只有两个奇数度数的节点**

void isHamilton(int D[],int n){

int ishp=1,ishg=1;

for(int i=0;i<n;i++){

for(int j=0;j<n;j++){

if(D[i]+D[j]<n-1){

ishp=0;

ishg=0;

break;

}

**上面就是如果节点度数合小于n-1了，那就直接break输出不存在哈密尔顿路，也不存在哈密尔顿图**

else if(D[i]+D[j]<n){

ishg=0;

break;

}

**上面就是如果节点度数合小于n了，那就直接break输出不存在哈密尔顿图**

}

if(!ishp)

break;

}

if(ishg)

cout<<"为哈密尔顿图"<<endl;

else if(ishp)

cout<<"存在哈密尔顿路"<<endl;//

}

**哈密顿图的条件：G是n节点简单图（没有自旋），G中每一对节点的度数和不小于n**

bool warshall(int M[100][100],int n){

int m[100][100];

memcpy(m,M,100\*100);

for(int j=0;j<n;j++)//warshall

for(int i=0;i<n;i++)

if(m[i][j]>=1)

for(int k=0;k<n;k++)

if(m[j][k]>=1)

m[i][k]=1;

printM(m,n,n);

for(int i=0;i<n;i++)

for(int j=0;j<n;j++)

if(!m[i][j])

return false;

return true;

}

**用传递闭包来判断图的连通性，首先把权值或者说路径长度都用1代替，然后判断是不是全都是1，只有矩阵全都是1的时候，才是联通的。**

void power(int M[100][100],int n){

int A[100][100],B[100][100],C[100][100],m[100][100];

memset(m,0,sizeof(m));

for(int i=0;i<n;i++)

for(int j=0;j<n;j++)

if(M[i][j]>=1)

m[i][j]=1;

memcpy(B,m,100\*100);

memcpy(C,m,100\*100);

cout<<"长度为1的路的情况即直接连接情况"<<endl;

printM(m,n,n);

**把有一条路径的输出出来**

for(int i=0;i<n-1;i++){

memcpy(A,B,100\*100);

memset(B,0,sizeof(B));

matrixmulti(A,m,B,n);

cout<<"长度为"<<i+2<<"路的情况"<<endl;

printM(B,n,n);

matrixplus(C,B,n);

cout<<endl;

}

cout<<"汇总表:"<<endl;;

printM(C,n,n);

}

**先做一个矩阵乘法，把n条路的输出——有几条路乘几次**

struct stree{

int pointa,pointb,weight;

};

int powellColor(int A[][100],int n,int Color[]/\*,int d[]\*/){

//先求出结点的度数

int subIndex[100],i=0,j=0,k=0,k0=0,itmp=0,thisColor[100],m=0,nthisColor=0,D[100];

for(i=0;i<n;i++){

subIndex[i]=i;

Color[i]=0;

}

calDegree(A,n,D);

**将结点度数从高到低排队**

for(i=0;i<n;i++){

for(j=n-1;j>i;j--){

if(D[j]>D[j-1]){

**小于后面的互换，小的往后走**

itmp=D[j];D[j]=D[j+1];D[j+1]=itmp;

itmp=subIndex[j];subIndex[j]=subIndex[j+1];subIndex[j+1]=itmp;

}

}

}

printf("排序后的结点度数：\n");

for(i=0;i<n;i++)

printf("%4d[%d]",D[i],subIndex[i]);

**排序后最先的颜色数**

**5[4] 3[0] 3[1] 3[2] 2[3] 2[5]各点的颜色**

itmp=0;//颜色号

for(i=0;i<n;i++){

//thisColor清空

for(j=0;j<n;j++){

thisColor[j]=0;

}

nthisColor=0;

**寻找为着色的第一个结点**

j=0;

while((D[j]==-1)&&(j<n)){

j++;

}

if(j>=n){

break;

}

k0=subIndex[j];

**上面的是未着色的结点号**

itmp++;**新的颜色**

Color[k0]=itmp;**着色**

D[j]=-1;**已着色点度数为-1，表示已经处理过了**

thisColor[nthisColor]=k0;**首种颜色的结点序号**

printf("\nk0=%d j=%d nthisColor=%d itmp=%d\n",k0,j,nthisColor,itmp);

nthisColor++;**同色下一个结点保存位置号**

j++;**下一个染色结点的寻找起始位置**

**从j以后的点中寻找与本轮已经着色的点不相邻的点染相同的色**

while(1){

while((D[j]==-1)&&(j<n)){

j++;

}

if(j==n){

break;

}

k=subIndex[j];

**未找到了未染色的结点**

**判断k是否与thisColor中点相邻**

for(m=0;m<nthisColor;m++){

**只要与一个相邻就**

if(A[k][thisColor[m]]>0){

break;

}

}

printf("j=%d n=%d k=%d m=%d nthisColor=%d\n",j,n,k,m,nthisColor);

if(m>=nthisColor){

Color[k]=itmp;

thisColor[m]=k;

nthisColor++;

D[j]=-1;

}

j++;

**再去看下一个节点**

}

}

return 0;

}

int getEdge(int A[][100],int n,struct stree T[]){

int i=0,j=0,nstree=0;

for(i=0;i<n;i++){

**因为对称，所以只要右上角就可以了**

for(j=i;j<n;j++){

if(A[i][j]>0){

T[nstree].pointa=i;

T[nstree].pointb=j;

T[nstree].weight=A[i][j];

nstree++;

}

}

}

return nstree;

}

void printTree(struct stree T[],int nT){//打印树

int i=0;

for(i=0;i<nT;i++)

printf("%d--%d(%d) ",T[i].pointa,T[i].pointb,T[i].weight);

cout<<endl;

}

先定义一个输出生成树的函数

void sortEdge(struct stree T[],int nT){

struct stree t0;

int i=0,j=0;

for(i=0;i<nT-1;i++)

for(j=0;j<nT-1-i;j++)

if(T[i].weight>T[j+1].weight)

**违反了低到高的原则，则要交换**

{

t0=T[j];

T[j]=T[j+1];

T[j+1]=t0;

}

}

int kruskal(struct stree T0[],int nt0,int n,struct stree T[]){

**根据边权从低到高排好序，选择最小的n-1条边**

int i=0,j=0,k=0,nt=0,B[100][100],m=0,mtmp=0,nCount=0;

T[nt]=T0[0];

nt++;

**针对每条边进行处理**

for(i=1;i<nt0;i++){

**判断当前边是否跟已经选定的边构成回路** **清空当前关联矩阵，最多到所有边**

for(j=0;j<n;j++){

for(k=0;k<nt0;k++){

B[j][k]=0;

}

}

for(j=0;j<nt;j++){

**第j条边的两个端点**

B[T[j].pointa][j]=1;

B[T[j].pointb][j]=1;

}

**加上第i条边的第二个端点**

B[T0[i].pointa][j]=1;

B[T0[i].pointb][j]=1;

printf("加入新边后的关联矩阵\n");

for(j=0;j<n;j++){

for(k=0;k<=nt;k++){

printf("%4d",B[j][k]);

}

printf("\n");

}

**将首列第一个1所在行互换到首行，然后将剩下其他行清0**

**从第0列到最后一列**

for(k=0;k<=nt;k++){

**从第k行起向下寻找首个1**

for(j=k;j<n;j++){

if(B[j][k]!=0){

if(j>k){

for(m=0;m<=nt;m++){

mtmp=B[j][m];B[j][m]=B[k][m];B[k][m]=mtmp;

}

}

break;

}

}

**将k+1行--最后一行，减去第k行**

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

if(B[j][k]==1){

for(m=k;m<=nt;m++){

B[j][m]=B[j][m]-B[k][m]\*B[j][k]/B[k][k];

}

}

}

}

printf("关联矩阵处理以后：\n");

for(j=0;j<n;j++){

for(k=0;k<=nt;k++){

printf("%4d",B[j][k]);

}

printf("\n");

}

**统计B[j][j]非0的个数**

nCount=0;

for(k=0;k<=nt;k++){

if(B[k][k]!=0){

nCount++;

}

}

if(nCount==(nt+1)){

**线性无关**

T[nt]=T0[i];

nt++;

if(nt==(n-1)){

break;

}

}

}

return nt;

}

int prim(struct stree T0[],int nt0,int n,struct stree T[]){

**根据边权从低到高排好序，便于选择**

int i=0,j=0,k=0,nt=0;

int U[100],mDis=0,iDis=0,jtree=0,ntree=0;

**预置第0个点**

U[nt]=0;

printf("U[%f]=%d\n",nt,U[nt]);

nt++;

**当点数没有到n时**

while(nt<n){

mDis=999999;

iDis=-1;

**依次寻找离U中各点最近的点**

for(i=0;i<nt;i++){

k=U[i];**U中点i的编号**

**寻找离k最近的点**

for(j=0;j<nt0;j++){

**k出去的点**

if(T0[j].pointa==k){

if(mDis>T0[j].weight){

mDis=T0[j].weight;

iDis=T0[j].pointb;

jtree=j;

}

}

**到k的点**

else if(T0[j].pointb==k){

if(mDis>T0[j].weight){

mDis=T0[j].weight;

iDis=T0[j].pointa;

jtree=j;

}

}

}

}

U[nt]=iDis;**吸收最短的点进来**

T[ntree]=T0[jtree];**吸收最短的边，让这个最短的边进入集合。**

T0[jtree].weight=7484006;**用很大的数字来使他不能被选**

printf("nt=%d U[%d]=%d ntree=%d jtree=%d\n",nt,nt,U[nt],ntree,jtree);

ntree++;

nt++;

}

return ntree;

}

int main(){

struct stree st0[100],st1[100];

int A[100][100],B[100][100],D[100],n,m,choice,nt0=0,nt=0;

int Color[100];

string a;

printf("\n========================\n");

printf("1...输入邻接矩阵\n");

printf("11...输入关联\n");

printf("2...邻接矩阵求每点度数及判断图的性质\n");

printf("3...关联矩阵求每点度数及判断图的性质\n");

printf("4...利用warshall算法判断图是否连通\n");

printf("5...计算A，A的平方，A的立方\n");

printf("6...利用powell染色算法对结点染色\n");

printf("7...利用kruskal求最小生成树\n");

printf("8...利用prim算法求最小生成树\n");

printf("0...退出\n========================\n");

while(printf("您的选择:(输入0则结束)")){

scanf("%d",&choice);

fflush(stdin);

if(choice==0) break;

switch(choice){

case 1:{

cout<<"点数：";

cin>>n;

cout<<endl<<"输入邻接矩阵的值:"<<endl;

getMar(A,n,n);

cout<<"邻接矩阵"<<endl;

printM(A,n,n);

break;

}

case 11:{

cout<<"边数：";

cin>>m;

cout<<endl<<"输入关联矩阵的值:"<<endl;

getMar(B,n,m);

cout<<"关联矩阵"<<endl;

printM(B,n,m);

break;

}

case 2:{

memset(D,0,sizeof(D));

calDegree(A,n,D);

isEular(D,n);

isHamilton(D,n);

break;

}

case 3:{

memset(D,0,sizeof(D));

for(int i=0;i<n;i++){

for(int j=0;j<m;j++){

if(B[i][j])

D[i]++;

}

}

isEular(D,n);

isHamilton(D,n);

break;

}

case 4:{

if(warshall(A,n))

cout<<"图连通"<<endl;

else

cout<<"图不连通"<<endl;

break;

}

case 5:{

power(A,n);

break;

}

case 6:{

powellColor(A,n,Color);

printf("各点的颜色：\n");

for(int i=0;i<n;i++)

printf("%2d",Color[i]);

break;

}

case 7:{

nt0=getEdge(A,n,st0);

printTree(st0,nt0);

sortEdge(st0,nt0);

printTree(st0,nt0);

nt=kruskal(st0,nt0,n,st1);

printTree(st1,nt);

cout<<endl;

break;

}

case 8:{

nt0=getEdge(A,n,st0);

printTree(st0,nt0);

sortEdge(st0,nt0);

printTree(st0,nt0);

nt=prim(st0,nt0,n,st1);

printTree(st1,nt);

break;

}

}

}

return 0;

}

/\*

1

6

0 0 6 0 5 3

0 0 4 5 5 0

6 4 0 0 2 0

0 5 0 0 1 0

5 5 2 1 0 4

3 0 0 0 4 0

1 0 0 0 0 1 1 0 0

0 1 1 0 0 0 0 0 1

1 1 0 0 0 0 0 1 0

0 0 1 1 0 0 0 0 0

0 0 0 1 1 0 1 1 1

0 0 0 0 1 1 0 0 0

\*/