

图实验报告

课程名称：数据结构与算法

学 院： 计算机与信息安全学院

专业班级：19003603

学 号：1900301517

姓 名：陆洪业

报告日期：2021年 01 月 10 日

目录

**[1. 利用队列实现密码加密解密实验目的](#_Toc1831915961_WPSOffice_Level1)** **[1](#_Toc1831915961_WPSOffice_Level1)**

[1.1. 选用储存结构](#_Toc647397264_WPSOffice_Level2) [1](#_Toc647397264_WPSOffice_Level2)

[1.2. 实验代码](#_Toc1130979933_WPSOffice_Level2) [1](#_Toc1130979933_WPSOffice_Level2)

[1.3. 系统测试与结果](#_Toc1068613576_WPSOffice_Level2) [7](#_Toc1068613576_WPSOffice_Level2)

[1.4. 实验小结](#_Toc1598252537_WPSOffice_Level2) [8](#_Toc1598252537_WPSOffice_Level2)

# 图课上课后练

根据下面的描述和要求，完成图的邻接矩阵数据结构定义，及图初始化函数。

编写一个能输入图的基本信息（含图的类型，图的顶点，边等），并用邻接表存储图的程序。

完成程序能实现图的广度遍历。

完成程序能实现图的深度遍历。

编写一个能输出有向无环图的拓扑排序的函数。

实现prim算法生成最小生成树。

## 选用储存结构

图

## 实验代码

#include <stdio.h>

#include <stdlib.h>

#define N 10

//邻接矩阵数据结构

typedef struct{

int vcount;//顶点数

int type ;//0 无向图，1 有向图

char vexs[N] ; // 顶点信息

int arcs[N][N]; //关系矩阵

} GraphMatrix;

//邻接表数据结构

struct EdgeNode {//边表中的结点

int start,end;

int endvex; //相邻顶点在顶点表中下标

int weight; //边的权

struct EdgeNode \* nextedge; //链字段

};

typedef struct EdgeNode \* EdgeList;

typedef struct

{

char vertex; //记录顶点信息

int degree;//用于记录顶点的入度，在拓扑排序时需使用

EdgeList edgelist; //指向边表的指针

} VexNode;

typedef struct{

VexNode vexs[N]; //N个顶点

int type ;//0 无向图，1 有向图

int vcount;//顶点数

} GraphList;

/\* 本关任务：根据下面的描述和要求，完成图的邻接矩阵数据结构定义，及图初始化函数。

\*in\_put 0 5 6 0 1 0 3 1 2 1 4 2 3 2 4

out\_put 0 1 0 1 0

1 0 1 0 1

0 1 0 1 1

1 0 1 0 0

0 1 1 0 0 \*/

/\*第一关 完成图初始化

\*/

void printGraph(GraphMatrix \*G)

{

//本函数输出图的邻接矩阵

int i,j;

for(i=0;i<G->vcount;i++)

{

// printf("%c ",G->vexs[i]);

for( j=0;j<G->vcount;j++)

printf("%d ",G->arcs[i][j]);

printf("\n");

}

}

/\*第一关 完成图初始化-邻接矩阵

\*/

GraphMatrix \*initGraphMatrix( )

{

/\*第一行输入图的类型、图的顶点数和边数，第二行输入各条边的两顶点的编号，按顶点编号从小到大的顺序输入。

输出邻接矩阵，不需要考虑输入的数字非法情况，不输入顶点的信息\*/

GraphMatrix\* Ga = (GraphMatrix\*)malloc(sizeof(GraphMatrix));

int sum ,x ,y;

scanf("%d%d%d",&Ga->type,&Ga->vcount,&sum);

for (int i = 0;i<Ga->vcount ; i++)

for (int j =0; j< Ga->vcount;j++)

Ga->arcs[i][j]=0;

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

{

scanf("%d%d",&x,&y);

Ga->arcs[x][y]=1;

if (Ga->type == 0)

Ga->arcs[y][x]=1;

}

return Ga;

}

/\*第二关 完成图初始化-邻接表,并完成输出图的邻接表函数

\*/

GraphList \*initGraphList( )

{

/\*第一行输入图的类型、图的顶点数和边数，第二行输入各条边的两顶点的编号，按顶点编号从小到大的顺序输入。

输出邻接表。不需考虑输入的数字非法情况，输入顶点的信息\*/

GraphList \*p=(GraphList \*)malloc(sizeof(GraphList ));

int edgecount;

scanf("%d %d %d",&p->type,&p->vcount,&edgecount);

scanf("%s",p->vexs);

for(int t=0;t<p->vcount;t++){

p->vexs[t].edgelist=NULL;

}

for(int t=0;t<p->vcount;t++){

p->vexs[t].degree=0;

}

for(int t=0;t<edgecount;t++){

int i,j,w;

scanf("%d %d",&i,&j);

// scanf("%d",&w);//只有第六关需要打开

if(p->type==1){

p->vexs[j].degree++;

EdgeList q1=(EdgeList)malloc(sizeof(struct EdgeNode ));

q1->nextedge=NULL;

q1->endvex=j;

q1->nextedge=p->vexs[i].edgelist;

p->vexs[i].edgelist=q1;

}

else {EdgeList q1=(EdgeList)malloc(sizeof(struct EdgeNode ));

EdgeList q2=(EdgeList)malloc(sizeof(struct EdgeNode ));

q1->weight=w;

q2->weight=w;

q1->nextedge=NULL;

q2->nextedge=NULL;

q1->endvex=j;

q2->endvex=i;

q1->nextedge=p->vexs[i].edgelist;

p->vexs[i].edgelist=q1;

q2->nextedge=p->vexs[j].edgelist;

p->vexs[j].edgelist=q2;

}

}

return p;

}

void printGraphLit( GraphList \*G)

{

/\*输出邻接表，输出格式：顶点->邻接顶点编号->...\*/

for(int i=0;i<G->vcount;i++){

printf("%d",i);

EdgeList p=G->vexs[i].edgelist;

while(p!=NULL){

printf("->%d",p->endvex);

p=p->nextedge;

}

printf("\n");

}

}

/\*第三关 完成图的广度遍历（周游），可根据需要添加自定义函数

\*/

void BFS\_list(GraphList \*G)

{

int a[100]={0};

EdgeList p=G->vexs[0].edgelist;

printf("0 ");

a[0]=1;

while(p!=NULL){

printf("%d ",p->endvex);

a[p->endvex]=1;

p=p->nextedge;

}

p=G->vexs[0].edgelist;

while(p!=NULL){

for(EdgeList q=G->vexs[p->endvex].edgelist;q!=NULL;q=q->nextedge){

if(a[q->endvex]!=1){

printf("%d ",q->endvex);

a[q->endvex]=1;

}

}

p=p->nextedge;

}

}

/\*第四关 完成图的深度遍历（周游），可根据需要添加自定义函数

\*/

void dfs(GraphList \*G,EdgeList p,int a[]){

a[p->endvex]=1;

for(EdgeList q=G->vexs[p->endvex].edgelist;q!=NULL;q=q->nextedge){

if(a[q->endvex]!=1){printf("%d ",q->endvex);dfs(G,q,a);}

}

}

void DFS\_list(GraphList \*G)

{

int a[100]={0};

printf("0 ");

a[0]=1;

for(EdgeList q=G->vexs[0].edgelist;q!=NULL;q=q->nextedge){

if(a[q->endvex]!=1){

printf("%d ",q->endvex);

dfs(G,q,a);

}

}

}

/\*第五关 生成图的拓扑排序，可根据需要添加自定义函数

\*/

void top1(GraphList \*G,int a[],int i){

a[i]=1;

EdgeList p=G->vexs[i].edgelist;

while(p!=NULL){

G->vexs[p->endvex].degree--;

p=p->nextedge;

}

for(int j=0;i<G->vcount;i++){

if(G->vexs[j].degree==0&&a[j]==0){

printf("%d ",j);

top1(G,a,j);

}

}

}

void Top\_list(GraphList \*G)

{ int a[100]={0};

int i;

for(i=0;i<G->vcount;i++){

if(G->vexs[i].degree==0&&a[i]==0){

printf("%d ",i);

top1(G,a,i);

}

}

}

/\*第六关 prim算法生成最小生成树

\*/

void Prim\_list(GraphList \*G)

{

struct EdgeNode a[G->vcount-1];

for(int i=0;i<G->vcount-1;i++)

a[i].weight=10000;

for(int i=0;i<G->vcount;i++){

EdgeList p=G->vexs[i].edgelist;

while(p!=NULL){

int t=p->endvex-1;

if(p->endvex>i){

if(p->weight<a[t].weight){

a[t].weight=p->weight;

a[t].start=i;

a[t].end=p->endvex;

}

}

p=p->nextedge;

}

}

if(a[G->vcount-2].weight<a[G->vcount-3].weight)

{

struct EdgeNode min;

min.weight=a[G->vcount-2].weight;

min.start=a[G->vcount-2].start;

min.end=a[G->vcount-2].end;

a[G->vcount-2].weight=a[G->vcount-3].weight;

a[G->vcount-2].start=a[G->vcount-3].start;

a[G->vcount-2].end=a[G->vcount-3].end;

a[G->vcount-3].weight=min.weight;

a[G->vcount-3].start=min.start;

a[G->vcount-3].end=min.end;

}

for(int i=0;i<G->vcount-1;i++)

printf("%d %d\n",a[i].start,a[i].end);

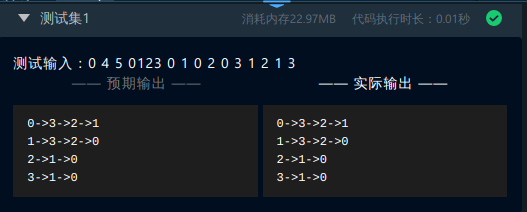
}

## 系统测试与结果

图的邻接表存储及图初始化



图的邻接矩阵存储及图初始化



广度遍历



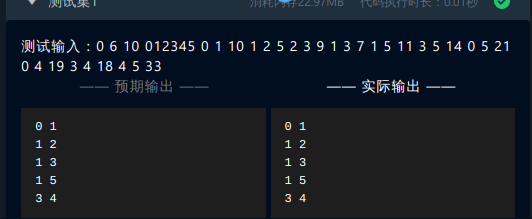
深度遍历



拓扑排序



Prim算法生成最小生成树



上述测试及结果证明了系统测试通过,各项操作函数能正常执行,并达到了实验要求的目标,实现了图的储存遍历以及拓扑排序和最小生成树.

## 实验小结

通过本次实验, 我学会了图的基本操作以及拓扑排序和最小生成树.