

**计算机与信息 学院实验报告**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 实验课程： | 数据结构 | | | | |
| 实验编号： | 实验九 | | | | |
| 实验名称： | 图的存储与遍历 | | | | |
| 实验人员： | 学号 | 18111303044 | | | |
| 姓名 | 邵一波 | | | |
| 班级 | 2018级计算机类1班 | | | |
| 实验日期： | 2019-12-12 | | | | |
| 实验室： | 2号实验楼202室 | | | | |
|  |  | | | | |
| 实验评价： |  | | | | |
| 实验成绩： | |  | 评价日期： |  |
|  | 指导教师： | |  | | |

|  |
| --- |
| **注：具体内容可根据专业特点和实验性质略作调整，页面不够可附页。**   1. **需求分析** 2. 描述：熟练掌握无向图的两种储存方式，已经广度优先和深度优先两种搜素方法。 3. 输入的形式和输入值的范围 4. 顶点数(小于20的正整数) 5. 边数(小于20的正整数) 6. 顶点(大写或小写的英文字母) 7. 边(格式为两个顶点的字符对）      1. 输出的形式 2. 邻接矩阵      1. 邻接链表      1. DFS,BFS      1. 程序所能达到的功能   从键盘接收顶点数，边数，顶点，边。输出邻接矩阵和邻接链表，以及其对应的广度优先和深度的搜索序列。   1. 测试数据   7  7  ABCDEFG  AC  AD  AF  BC  CD  EG  FG   1. **概要设计** 2. 抽象数据类型的定义   **CommonDef.h**  #include<stdio.h>  #include<stdlib.h>  #include<malloc.h>  #include<string.h>  #define MAX 20  #define MVNum 20  int visited[MVNum];  **AdjList.h**  #include "CommonDef.h"  typedef struct ArcNode  {  int adjvex; //该边所指的顶点的位置  struct ArcNode \*nextarc; //指向下一条边的指针  }ArcNode; //表的结点  typedef struct VNode  {  char data; //顶点信息（如数据等）  ArcNode \*firstarc; //指向第一条依附该顶点的边的弧指针  }VNode,AdjList[MAX]; //头结点  typedef struct ALGraph  {  AdjList vertices;  int vexnum, arcnum; //图的当前顶点数和弧数  }ALGraph;  **AdjMatrix.h**  #include "CommonDef.h"  typedef struct  {  char vexs[MAX];  int vexnum;  int edgnum;  int matrix[MAX][MAX];  }AMGraph;  **Queue.h**  #include <stdio.h>  #define QueueSize 20  //队列  typedef struct  {  int front,rear,count;  int data[QueueSize];  }CirQueue;   1. 函数的定义   **AdjList.h**  #include "CommonDef.h"  /\* 返回顶点的位置 \*/  int LocateVex(ALGraph G , char v);  /\* 构造无向图(邻接链表) \*/  ALGraph CreateAdjList();  /\* 输出无向图(邻接链表) \*/  void PrintAdjList(ALGraph G);  **AdjMatrix.h**  #include "CommonDef.h"  /\* 返回顶点的位置 \*/  int PositionVex(AMGraph g,char ch);  /\* 构造无向图(邻接矩阵) \*/  AMGraph\* CreateAdjMatrix();  /\* 输出无向图(邻接矩阵) \*/  void PrintAdjMatrix(AMGraph G);  **Queue.h**  void InitQueue(CirQueue \*Q);  int QueueEmpty(CirQueue \*Q);  int QueueFull(CirQueue \*Q);  void EnQueue(CirQueue \*Q,int x);  int DeQueue(CirQueue \*Q);  **ListSearch.h**  #include "AdjList.h"  // 深度优先搜索  void DFS(ALGraph G,int i);  void DFSL(ALGraph G);  // 广度优先搜索  void BFS(ALGraph G,int k);  void BFSL(ALGraph G);  **MatrixSearch.h**  #include "AdjMatrix.h"  void BFSTraverse(AMGraph G);  void DFS2(AMGraph G,int i);  void DFStraverse(AMGraph G);   1. 主程序的流程图      1. **详细设计**   **UDNtest.c**  #include "AdjMatrix.h"  #include "AdjList.h"  /\* 主函数 \*/  int main()  {  AMGraph \*G1;  ALGraph G2;  printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n无向图的邻接矩阵存储\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");  G1=CreateAdjMatrix();  PrintAdjMatrix(\*G1);  DFStraverse(\*G1);  BFSTraverse(\*G1);  printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n无向图的邻接链表存储\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");  G2 = CreateAdjList();  PrintAdjList(G2);  DFSL(G2);  BFSL(G2);  return 0;  }  **Queue.c**  #include "Queue.h"  //队列  void InitQueue(CirQueue \*Q)  {  Q->front=Q->rear=0;  Q->count=0;  }  int QueueEmpty(CirQueue \*Q)  {  return Q->count==0;  }  int QueueFull(CirQueue \*Q)  {  return Q->count==QueueSize;  }  void EnQueue(CirQueue \*Q,int x)  {  if(QueueFull(Q))  printf("Queue overflow");  else  {  Q->count++;  Q->data[Q->rear]=x;  Q->rear=(Q->rear+1)%QueueSize;  }  }  int DeQueue(CirQueue \*Q)  {  int temp;  if(QueueEmpty(Q))  {  printf("Queue underflow");  return -1;  }  else  {  temp=Q->data[Q->front];  Q->count--;  Q->front=(Q->front+1)%QueueSize;  return temp;  }  }  **AdjList.c**  #include "AdjList.h"  /\* 返回顶点的位置 \*/  int LocateVex(ALGraph G , char v)  {  int i;  for(i = 0;v != G.vertices[i].data && i < G.vexnum; i++);  if(i >= G.vexnum)  return -1;  return i;  }  /\* 构造无向图(邻接链表) \*/  ALGraph CreateAdjList()  {  ALGraph G;  int i,j,k;  char v1, v2;  ArcNode \*s, \*t;  printf("输入顶点数:");  scanf("%d", &G.vexnum);  printf("输入边数:");  scanf("%d", &G.arcnum);  getchar();    printf("输入顶点:");  for(i = 0; i < G.vexnum; i++)  {  scanf("%c", &G.vertices[i].data); //构造顶点向量  G.vertices[i].firstarc = 0;    }  getchar();    printf("输入边(例:AB):\n");  for(k = 0; k < G.arcnum; k++)  {  scanf("%c", &v1);  scanf("%c", &v2);  getchar();  i = LocateVex(G , v1);  j = LocateVex(G , v2); //确定 v1 , v2在 G 中的位置  s = (ArcNode\*) malloc (sizeof(ArcNode));  t = (ArcNode\*) malloc (sizeof(ArcNode));  s->adjvex = j; //该边所指向的顶点的位置为 j  s->nextarc = G.vertices[i].firstarc;  G.vertices[i].firstarc =s;  t->adjvex = i; //该边所指向的顶点的位置为 j  t->nextarc = G.vertices[j].firstarc;  G.vertices[j].firstarc =t;  }  return G;  }  /\* 输出无向图(邻接链表) \*/  void PrintAdjList(ALGraph G)  {  int i;  ArcNode \*p;  printf("Adjacency List:\n");  for(i = 0; i < G.vexnum; i++)  {  printf("%d%4c", i, G .vertices[i].data);  for(p = G.vertices[i].firstarc; p; p = p->nextarc)  printf(" ->%2d", p->adjvex);  printf("\n");  }  }  **AdjMatrix.c**  #include "AdjMatrix.h"  /\* 返回顶点的位置 \*/  int PositionVex(AMGraph g,char ch)  {  int i;  for(i=0;i<g.vexnum;i++)  if(g.vexs[i]==ch)  return i;  return -1;  }  /\* 构造无向图(邻接矩阵) \*/  AMGraph\* CreateAdjMatrix()  {  int vlen;  int elen;  int i,p1,p2;  char \*vexs;  char (\*edges)[2];  AMGraph \*pG;  //获取顶点数和边数  printf("输入顶点数:");  scanf("%d",&vlen);  printf("输入边数:");  scanf("%d",&elen);  getchar();  //开辟空间用于储存点和边  vexs = (char \*)malloc(sizeof(char) \* vlen);  edges = (char(\*)[2])malloc(sizeof(char)\*elen\*2);  //获得顶点和边的信息  printf("输入顶点:");  for(i = 0;i < vlen;i++)  scanf("%c",&vexs[i]);  getchar();  printf("输入边(例:AB):\n");  for(i = 0;i < elen;i++)  {  scanf("%c%c",&edges[i][0],&edges[i][1]);  getchar();  }  //根据点和边的信息，以邻接矩阵构造无向图  if((pG=(AMGraph\*)malloc(sizeof(AMGraph)))==NULL)  return NULL;  memset(pG,0,sizeof(AMGraph));  pG->vexnum=vlen;  pG->edgnum=elen;  for(i=0;i<pG->vexnum;i++)  {  pG->vexs[i]=vexs[i];  }  for(i=0;i<pG->edgnum;i++)  {  p1=PositionVex(\*pG,edges[i][0]);  p2=PositionVex(\*pG,edges[i][1]);  pG->matrix[p1][p2]=1;  pG->matrix[p2][p1]=1;  }  return pG;  }  /\* 输出无向图(邻接矩阵) \*/  void PrintAdjMatrix(AMGraph G)  {  int i,j;  printf("Adjacency Matrix:\n");  for(i=0;i<G.vexnum;i++)  {  for(j=0;j<G.vexnum;j++)  printf("%d ",G.matrix[i][j]);  printf("\n");  }  }  **ListSearch.c**  #include "ListSearch.h"  #include "Queue.h"  //深度优先搜索  void DFS(ALGraph G,int i)  {  ArcNode \*p;  printf("%c ",G.vertices[i].data); //访问顶点 vi  visited[i]=1; // 标记 vi 已访问  p=G.vertices[i].firstarc; // 取 vi 边表的头指针  while(p)  {  // 依次搜索 vi 的邻接点 vj，这里 j=p->adjvex  if(!visited[p->adjvex]) //若vi尚未被访问  DFS(G,p->adjvex); // 则以vj为出发点向纵深搜索  p=p->nextarc; // 找 vi 的下一邻接点  }  }  void DFSL(ALGraph G)  {  int i;  printf("DFS: ");  for(i=0;i<G.vexnum;i++)  visited[i]=0;  for(i=0;i<G.vexnum;i++)  if(!visited[i])  DFS(G,i);  printf("\n");  }  //广度优先搜索  void BFS(ALGraph G,int k)  {  int i;  CirQueue Q; // 须将队列定义中 DataType 改为 int  ArcNode \*p;  InitQueue(&Q); // 队列初始化  printf("%c ",G.vertices[k].data); // 访问源点 vk  visited[k]=1;  EnQueue(&Q,k); //vk已访问，将其人队。（实际上是将其序号人队）  while(!QueueEmpty(&Q))  {  // 队非空则执行  i=DeQueue(&Q); // 相当于 vi 出队  p=G.vertices[i].firstarc; // 取 vi 的边表头指针  while(p)  {  // 依次搜索 vi 的邻接点 vj( 令 p->adjvex=j)  if(!visited[p->adjvex])  {  // 若 vj 未访问过  printf("%c ",G.vertices[p->adjvex].data); // 访问 vj  visited[p->adjvex]=1;  EnQueue(&Q,p->adjvex); // 访问过的 vj 人队  }  p=p->nextarc; // 找 vi 的下一邻接点  }  }  }  void BFSL(ALGraph G)  {  int i;  printf("BFS: ");  for(i=0;i<G.vexnum;i++)  visited[i]=0;  for(i=0;i<G.vexnum;i++)  if(!visited[i])  BFS(G,i);  printf("\n");  }  **MatrixSearch.c**  #include "MatrixSearch.h"  #include "Queue.h"  //深度优先递归算法  void DFS2(AMGraph G,int i)  {  int j;  visited[i]=1; //被访问的标记  printf("%c ",G.vexs[i]);  for(j=0;j<G.vexnum;j++)  if(G.matrix[i][j]==1 && !visited[j]) //边(i,j)存在且j顶点未被访问，递归  DFS2(G,j);  }    //深度优先遍历  void DFStraverse(AMGraph G)  {    int i;  for(i=0;i<G.vexnum;i++)  visited[i]=0;  printf("DFS: ");  for(i=0;i<G.vexnum;i++)  if(!visited[i])  DFS2(G,i);  printf("\n");  }  //广度优先  void BFSTraverse(AMGraph G)  {  int i;  int v;  CirQueue Q;  InitQueue(&Q);  for(i =0;i<G.vexnum;i++)  visited[i]=0;  printf("BFS: ");  for(v =0;v<G.vexnum;v++)  {  if(!visited[v])  {  visited[v]=1;  printf("%c ",G.vexs[v]);  EnQueue(&Q,v);  while(!QueueEmpty(&Q))  {  int u = DeQueue(&Q);  for(i=0;i<G.vexnum;i++)  {  if(G.matrix[u][i]&&visited[i]==0)  {  visited[i]=1;  printf("%c ",G.vexs[i]);  EnQueue(&Q,i);  }/\*if\*/  }  }/\*WHILE\*/    }/\*IF\*/    }/\*FOR\*/  printf("\n");  }   1. **调试分析** 2. 调试过程中遇到的问题是如何解决的以及对设计与实现的回顾讨论和分析   运算结果出错，经检查是函数参数问题   1. 算法的时空分析，改进设想   本次实验同时采用了链表和矩阵两种储存方式   1. **测试数据与结果** 2. 无向图的邻接矩阵      1. 无向图的邻接链表 |