图

一 目的

1、掌握图的基本存储结构和操作实现方法；

2、掌握图的搜索和遍历方法；

3、练习并解决实际问题。

二 内容

1、实现含有 n 个顶点和 m 条边的无向图的邻接表或邻接矩阵存储，并对存储结果进行测试。

2、在第 1 题存储结构的基础上，实现图的深度优先和广度优先遍历，并对结果进行测试。

3、设有 n 个城镇需要架设通信线路使各个城镇均能相互通信，在这些城镇中，除城镇 D、E 之间和城镇 F、G 之间由于地形问题无法直接架设线路之外，任意两个城镇之间均可直接架设线路且架设线路的费用为 X_{ij}，请采用合适的数据结构设计线路方案使得任意两个城镇间均能通信且架设线路的费用最少。

4*、苏州科技大学石湖校区有多个教学楼和图书馆，以及多个位于不同区域位置上的学习场所和多个不同的食堂，请为同学们设计从不同的学习场所到达最近食堂的方案，并通过测试验证方案的正确性。

三 设计说明

图的邻接矩阵是用来表示顶点之间相邻关系的矩阵，广度优先搜索类似树的层次遍历，是树的层次遍历的推广（需使用队列实现），深度优先搜索类似于树的先根遍历，是树的先根便利的推广（使用递归算法）。

四 功能说明

```
public enum GraphKind {
    UDG,    //无向图
    DG,     //有向图
    UDN,    //无向网络
    DN;     //有向网络
}
public boolean   CreateGraph(EdgeElement [] d); //建立图的存储结构
public int graphType();        //返回图的类型
public int vertices();         //返回图中的顶点数
public int edges();            //返回图中的边数
public boolean find(int i, int j);  //从图中查找一条边(i,j)是否存在，若存在则返回
```

真否则返回假
//public boolean putEdge(EdgeElement theEdge);　　//向图中插入一条边
//public boolean removeEdge(int i,int j);　　//向图中删除一条边
public int degree(int i);　　//返回顶点 i 的度
public int inDegree(int i); //返回顶点 i 的入度
public int outDegree(int i);//返回顶点 i 的出度
public void output();//以顶点集和边集的形式输出一个图
public void depthFirstSearch(int v);　　//从顶点 v 开始深度优先搜索遍历图
public void breadthFirstSearch(int v);　　//从顶点 v 开始广度优先搜索遍历图

五 调试分析

总的来说就是在树的遍历基础上进行推广，但是首先我们要对图的相关定义熟悉，学会在算法前手动作图。

六 测试结果

```
请输入图的类型：
UDN
请输入顶点个数和边的个数：
3
2
请依次输入顶点：
Q
W
E
请依次输入边依附的两个顶点及权值：
Q
W
5
Q
E
7
Q E W
```

七 带注释的源程序

```
package sj;
import java.util.*;
//图的抽象数据类型定义
interface Graph {
public boolean　CreateGraph(EdgeElement [] d); //建立图的存储结构
public int graphType();　　//返回图的类型
public int vertices();　　//返回图中的顶点数
public int edges();　　//返回图中的边数
public boolean find(int i, int j);　//从图中查找一条边(i,j)是否存在，若存在则返回真否则返回假
//public boolean putEdge(EdgeElement theEdge);　　//向图中插入一条边
```

```
//public boolean removeEdge(int i,int j);     //向图中删除一条边
public int degree(int i);     //返回顶点 i 的度
public int inDegree(int i); //返回顶点 i 的入度
public int outDegree(int i);//返回顶点 i 的出度
public void output();//以顶点集和边集的形式输出一个图
public void depthFirstSearch(int v);          //从顶点 v 开始深度优先搜索遍历图
public void breadthFirstSearch(int v);       //从顶点 v 开始广度优先搜索遍历图
void clearGraph();                            //清除图中的所有内容
}
interface Queue{
        public boolean isEmpty();          //若为空表返回 true,否则返回 false
        public void enter(int obj);     //结点(以编号表示)入队
        public int leave();                 //结点(以编号表示)出队
        }
class EdgeElement {                 //定义边集数组中的元素类型
   int fromvex;                     //边的起点域
   int endvex;                      //边的终点域
   int weight;                      //边的权值域，假定为整型，对于无权图，权值可
为 1
   public EdgeElement(int v1, int v2)
   {        //对无权图中的边进行初始化
        fromvex=v1; endvex=v2; weight=1;
   }
   public EdgeElement(int v1, int v2, int wgt)
   {        //对有权图中的边进行初始化
        fromvex=v1; endvex=v2; weight=wgt;
   }
}
class AdjacencyGraph implements Graph,Queue
{
   final static int MaxValue=1000;//一个不存在的边所使用的权值
   final static int MAXSIZE=18;
   private int n;                       //图的顶点数
   private int e;                       //图的边数
   private int type;                    //图的类型，分别用 0-3 的值表示图的四种类型
   private int [][]a;                   //图的邻接矩阵，假定元素类型为 int
   public int[][] getArray() {return a;}        //返回邻接矩阵的引用
   private int[]    elem;       //存循环队列中的元素的数组
   private int front,rear;            //队头和队尾指针
   public void initqueue()     //初始化空队列
      { elem= new int[MAXSIZE];
      front=0;
      rear=0; }
```

```
  public boolean isEmpty(){
      if(rear==front)    return true;     //表空,返回 true
      return false;
  }
  public int leave() {
     int obj;
       if(rear==front ){
       System.out.println("--队列空不能删除元素--");
       return -1;       }
       obj=elem[(front+1)%MAXSIZE];
       front=(front+1)%MAXSIZE;
       return obj;
       }

  public void enter(int    obj) {
        if(((rear+1)% MAXSIZE)!=front){   //入队操作，判队列是否满，是否能入
队
           rear=(rear+1)% MAXSIZE;
           elem[rear]=obj;
        }
      else
          System.out.println("--队列满不能入队!--");
      }

  public AdjacencyGraph(int n,    int t)          //构造函数定义
  {                                          //参数 n 和 t 分别表示图的顶点数和类型
      if(n<1 || t<0 || t>3) {
            System.out.println("初始化图的参数值错，退出运行!");
            System.exit(1);
      }
      this.n=n;    e=0;                          //初始化图的顶点数和边数的变量 n 和
e
      type=t;                                    //初始化表示图类型的数据成员 type
      a=new int[n][n];                          //给表示邻接矩阵的数组 a 分配存储空间
      for(int i=0; i<n; i++)                  //初始化数组 a 中的每个元素值
            for(int j=0; j<n; j++) {
                  if(i==j) a[i][j]=0;       //对角线元素被初始化为 0
                  else if (type==0 || type==2)
                        a[i][j]=0;              //对无权图的元素初始化为 0
                  else a[i][j]=MaxValue; //对带权图的元素初始化为 MaxValue
            }
  }
  public boolean CreateGraph(EdgeElement[] d)
  { for(int i=0;i<d.length;i++)    {
```

```
        if(d[i]==null) {System.out.println("边集空，停止建图，返回假！");
        return false;}
         int v1,v2;
         v1=d[i].fromvex;              v2=d[i].endvex;
         if(v1<0 || v1>n-1 ||v2<0 || v2>n-1 || v1==v2)
         {System.out.println("边集空，停止建图，返回假！");
        return false;}
         if(a[v1][v2]!=0 && a[v1][v2]!=MaxValue){System.out.println("边集空，停止
建图，返回假！");
        return false;}
          if(type==0 )     a[v1][v2] = a[v2][v1] =1;        //无向图且没有权值
          if(type==1 )     a[v1][v2] = a[v2][v1] =d[i].weight;       //无向图有权值
          if(type==2 )     a[v1][v2]   =1;        //有向图且没有权值
          if(type==3 )     a[v1][v2] = d[i].weight;        //有向图且有权值
        }
       e=d.length;
       return true;
    }   //根据边集数组参数 d 建立一个采用邻接矩阵表示的图
    public int graphType() {return type;}       //返回图的类型
    public int vertices() {return n;}             //返回图中的顶点数
    public int edges() {return e;}                 //返回图中的边数
  public boolean find(int i, int j)   //从图中查找一条边(i,j)是否存在，若存在则返回
真否则返回假
    { if(i<0 || i>n-1 || j<0 ||j>n-1 || i==j)
          { System.out.println("边的顶点序号无效，退出运行！"); System.exit(1);}
       if(a[i][j]!=0 && a[i][j]!=MaxValue) return true ;else return false;
    }
/*    public boolean putEdge(EdgeElement theEdge)
    { }   //向图中插入一条由参数给定的边 theEdge，若该边存在则不插入并返回
假
    public boolean removeEdge(int i, int j)
    { }   //从图中删除一条边(i,j)，若该边不存在则返回假     */
  public int inDegree(int i)   //顶点 i 的入度
  { int k=0;
        if(i<0 || i>n-1)
         { System.out.println("参数的顶点序号无效，退出运行！"); System.exit(1);}
    if(type==0 ||type==1) return degree(i);
    for(int j=0;j<n;j++)
             if(a[j][i]!=0 && a[j][i]!=MaxValue) k++;
         return k;
  }
  public int outDegree(int i) //返回顶点 i 的出度
  { int k=0;
        if(i<0 || i>n-1)
```

```
  { System.out.println("参数的顶点序号无效，退出运行！"); System.exit(1);}
     if(type==0 ||type==1) return degree(i);
     for(int j=0;j<n;j++)
      if(a[i][j]!=0 && a[i][j]!=MaxValue) k++;
       return k;

}
public int degree(int i)//返回顶点 i 的度
 {   if(i<0 || i>n-1)
       { System.out.println("参数的顶点序号无效，退出运行！"); System.exit(1);}
       int k=0;
       if(type==0 || type==1)
       {for(int j=0;j<n;j++)
          if(a[i][j]!= 0 && a[i][j]!=MaxValue) k++;
       return k;}
       else    return (inDegree(i)+outDegree(i));

 }
 public void clearGraph() {n=e=type=0; a=null; }   //清除图中的所有内容

 public void breadthFirstSearch(int v)
{       //从顶点 v 开始对图进行广度优先搜索遍历
   boolean []visited= new boolean[n];            //定义布尔型辅助数组
   for(int i=0; i<n; i++) visited[i]=false; //每个元素被初始化为假
   bfs(v, visited);          //调用进行广度优先搜索遍历的内部非递归方法
   System.out.println(); //输出一个空行
}

 public   void bfs(int i, boolean[] visited) //进行广度优先搜索遍历的内部非递归方
法的定义
  { int k;
  initqueue();
  System.out.print(i+" ");
  visited[i]=true;
  enter(i);
  while( !isEmpty() ){
       k=leave();
       for(int j=0; j<n; j++){
       if(a[k][j]!=0 && a[k][j]!=MaxValue && !visited[j])
       { System.out.print(j+" ");
       visited[j]=true;
       enter(j);}    //if
       }//for
      } //while
```

```
    }
   public void depthFirstSearch(int v)
   {       //从顶点 v 开始对图进行深度优先搜索遍历
         boolean []visited= new boolean[n];          //定义布尔型辅助数组
         for(int i=0; i<n; i++) visited[i]=false; //每个元素被初始化为假
         dfs(v, visited);          //调用进行深度优先搜索遍历的内部递归方法
         System.out.println(); //输出一个空行
   }

public void dfs(int i, boolean[] visited)   //进行深度优先搜索遍历的内部递归方法的定义
   {System.out.print(i+" ");
   visited[i]=true;        //标记 vi 已被访问过
    for(int j=0; j<n; j++)
        if(a[i][j]!=0 && a[i][j]!=MaxValue && !visited[j])
        { dfs(j, visited); }

    }
public void output()                  //输出用邻接矩阵表示的图所对应的顶点集和边集
{ int i,j;
 System.out.print("V={")   ;      //输出顶点集
 for(i=0;i<n-1;i++) System.out.print(i+",");
 System.out.println(n-1+"}");
 System.out.print("E={")   ;
 if(type==0 || type==2) {
    for(i=0;i<n;i++)
         for(j=0;j<n;j++)
              if(a[i][j]!=0 && a[i][j]!=MaxValue )
                   if(type==0) {if(i<j) System.out.print("("+i+","+j+"),");}
                   else System.out.print("<"+i+","+j+">,");
 }
 else
 {for(i=0;i<n;i++)
         for(j=0;j<n;j++)
              if(a[i][j]!=0 && a[i][j]!=MaxValue )
                   if(type==1) {if(i<j) System.out.print("("+i+","+j+")"+a[i][j]+",");}
                   else System.out.print("<"+i+","+j+">"+a[i][j]+",");

 }
 System.out.println("}");
}    //输出用集合表示的一个图的顶点集和边集

}
```

```
public class TestGh {
public static void main(String[] args) {
System.out.println("---下面测试图的遍历算法实现情况----");
int n,t;
AdjacencyGraph ga=new AdjacencyGraph(5,1);
int [][]a={{0,1,5},{0,2,7},{1,2,12},{1,3,3},{2,3,6},{1,4,8},{3,4,15},{2,4,20}};
EdgeElement []dd=new EdgeElement[a.length];
for(int i=0;i<a.length;i++){
        dd[i]=new EdgeElement(a[i][0],a[i][1],a[i][2]);
          }
if(ga.CreateGraph(dd)) System.out.println("图的邻接矩阵存储成功！");
else {System.out.println("图的邻接矩阵存储不成功！推出程序运行！");
System.exit(1);}
System.out.println("输出的图的顶点集和边集为：");
ga.output();
System.out.println("输出的图的深度优先遍历结果为：");
ga.depthFirstSearch(4);
 System.out.println("输出的图的广度优先遍历结果为：");
ga.breadthFirstSearch(4);
n=2;
System.out.println(n+"的度："+ ga.degree(n));

}
}
```