**实 验 报 告 （ 3 ）**

|  |  |
| --- | --- |
| **实验名称**：图的应用 | **实验地点**：信息楼318 |
| **所使用的工具软件及环境：Win10, Java** | |
| **一、实验目的：**  1、理解图的含义；  2、掌握用邻接矩阵和邻接表的方法描述图的存储结构；  3、理解并掌握深度优先遍历和广度优先遍历的存储结构； | |
| **二、实验内容描述：**（填写题目内容及输入输出要求）  1、列出连通集。  给定一个有N个顶点和E条边的无向图，请用深度优先遍历（DFS）和广度优先遍历（BFS）分别列出其所有的连通集。假设顶点从0到N-1编号。进行搜索时，假设总是从编号最小的顶点出发，按编号递增的顺序访问邻接点。  输入：输入第1行给出2个整数N(0<N≤10)和E，分别是图的顶点数和边数。随后E行，每行给出一条边的两个端点。每行中的数字之间用一个空格分隔。  输出：按照“{v1v2…vk}”的格式，每行输出一个连通集。先输出DFS的结果，再输出BFS的结果。  Input：8 6 output：{0 1 4 2 7}   1. 7 {3 5} 2. 1 {6}   2 0 {0 1 2 7 4}  4 1 {3 5}  2 4 {6}  3 5 | |
| **三、程序运行结果（说明设计思路，解释使用的数据结构，计算时间复杂度）**  **第1题**   1. 实验运行结果截图      1. 数据结构   采用数组存储图中顶点并建立邻接数组，采用链表充当队列，在图类中使用一个布尔数组用来标记是否访问过顶点，构造类如下：  //声明一个图的边类  class EdgeElement  {  int formvex;  int endvex;  int weight;  }  //声明一个图类  class Graph  {  ArrayList<Object> vex;//用来装图的顶点  int[][] edges;//邻接数组  boolean[] visited;//用来标记是否访问过顶点  LinkedList<Object> queue;//在BFS中用来充当队列  }   1. 设计思路   根据输入图的各边建立关系并放入图中，用边作为参数建立邻接数组，然后自定义BFS、DFS方法以某一顶点为起点寻找其所在连通集。  ·用边作为参数建立邻接数组时，将两边都设置为联通；  public Graph(int n) {  visited=new boolean[n];  vex=new ArrayList<Object>();  queue=new LinkedList<Object>();  edges=new int[n][n];  for (int i=0;i<n ;i++ )  {//用边作为参数建立邻接数组  for (int j=0;j<n ;j++ ) edges[i][j]=0;  visited[i]=false;  vex.add(i);  }  }  public void inPut(EdgeElement edg)  {  //两边都设置为1--即两边都联通  edges[edg.formvex][edg.endvex]=1;  edges[edg.endvex][edg.formvex]=1;  }  ·DFS函数中当遇到满足edges[i][j]==1&&visited[j]==false的顶点时，标记为visited[i]=true并执行递归，即一个点走到头；BFS函数中使用for循环一个点走一遍，遇到满足edges[n][j]==1&&visited[j]==false条件的点，进行标记visited[i]=true，并加入队列中。  public void serchByDFS(int i) {  visited[i]=true;  System.out.print(vex.get(i)+" ");  for (int j=0; j<vex.size();j++ ) {  if (edges[i][j]==1&&visited[j]==false)  serchByDFS(j);//一个点走到头  }  }  public void DFS() {  clear(visited);  for (int i=0;i<vex.size() ;i++ ) {  if (visited[i]==false) {  System.out.print("{ ");  serchByDFS(i);  System.out.print("}");  System.out.println();  }  }  }  public void serchByBFS(int i) //以某个顶点作为起点，寻找其所在连通集（BFS） {  visited[i]=true;  queue.add(vex.get(i));  while (queue.size()>0) {  int n=(int)queue.remove();  System.out.print(n+" ");  for (int j=0; j<vex.size();j++ )  {  if (edges[n][j]==1&&visited[j]==false) {  visited[j]=true;  queue.add(vex.get(j));//一个点走一遍，连通的放进队列，进入下一个循环  }  }  }  }  public void BFS() {  clear(visited);  for (int i=0;i<vex.size() ;i++ ) {  if (visited[i]==false) {  System.out.print("{ ");  serchByBFS(i);  System.out.print("}");  System.out.println();  }  }  }  ·在执行DFS、BFS函数时，首先需要将boolean[] visited清空。   1. 时间复杂度   创建图的过程有一个for循环，时间复杂度为O(n)；DFS函数为递归函数，时间复杂度为O(n)，BFS函数中有一个for循环，时间复杂度为O(n)，所以最终程序的时间复杂度为O(n)。 | |

2023年5月28日