哈尔滨工业大学计算机科学与技术学院

实验报告

|  |  |
| --- | --- |
| 课程名称： | 数据结构与算法 |
| 课程类型： | 必修 |
| 实验名称： | 图型结构的建立与搜索 |
|  |  |
| 班级： | 1703005 |
| 学号： | 1170300520 |
| 姓名： | 郭子阳 |

1. 实验目的

熟悉图型结构的建立和搜索。

1. 实验要求及实验环境

实验要求：编写程序演示图两种典型存储结构的建立和搜索（遍历）过程。

实验环境：macOS，VSCode，JDK 11

1. 设计思想（本程序中用到的所有数据类型的定义，核心算法的流程图等）

顶点表结构定义如下：

class VertexNode {

private int vertex;

private EdgeNode firstEdge;

public VertexNode(int *vertex*) {

this.vertex = vertex;

}

public int getVertex() {

return vertex;

}

public void setFirstEdge(EdgeNode *firstEdge*) {

this.firstEdge = firstEdge;

}

public EdgeNode getFirstEdge() {

return firstEdge;

}

}

边表结构定义入下：

class EdgeNode {

private int adjvex;

private EdgeNode next;

public EdgeNode(int *adjvex*) {

this.adjvex = adjvex;

next = null;

}

public int getAdjvex() {

return adjvex;

}

public void setNext(EdgeNode *next*) {

this.next = next;

}

public EdgeNode getNext() {

return next;

}

}

邻接表结构定义如下：

class AdjGraph {

private ArrayList<VertexNode> vexList;

public AdjGraph() {

vexList = new ArrayList<>();

}

public void addVertex(VertexNode *vertex*) {

vexList.add(vertex);

}

public VertexNode getVertex(int *index*) {

return vexList.get(index);

}

}

栈结构定义如下：

class Stack<T> {

private ArrayList<T> list;

public Stack() {

list = new ArrayList<>();

}

public void push(T *element*) {

list.add(element);

}

public T pop() {

return list.remove(list.size() - 1);

}

public T getTop() {

return list.get(list.size() - 1);

}

public boolean isEmpty() {

return list.isEmpty();

}

}

队列结构定义如下：

class Queue<T> {

private ArrayList<T> list;

public Queue() {

list = new ArrayList<>();

}

public void enQueue(T *element*) {

list.add(element);

}

public T deQueue() {

return list.remove(0);

}

public T getFirst() {

return list.get(0);

}

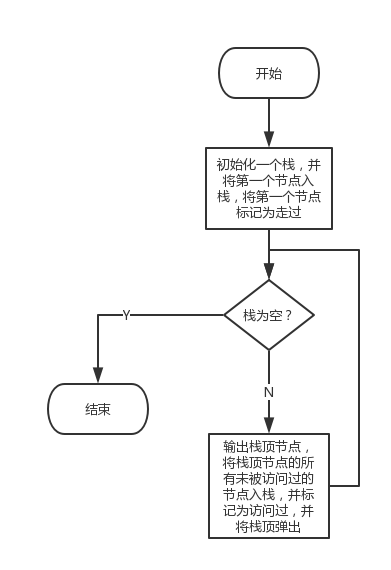
public boolean isEmpty() {

return list.isEmpty();

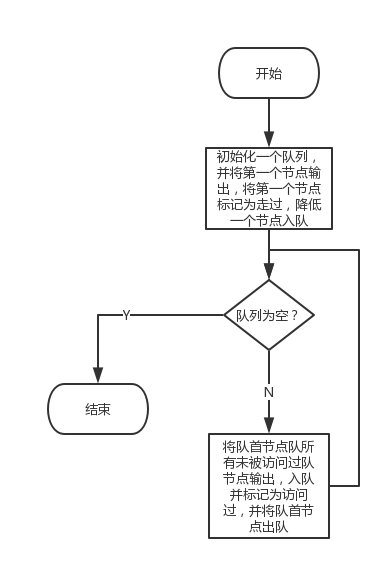
}

}

非递归深度优先搜索：



广度优先搜索：



1. 测试结果



1. 系统不足与经验体会

本次实验深刻理解了图结构的设计与搜索算法。

1. 源代码

import java.io.BufferedReader;

import java.util.ArrayList;

import java.util.Scanner;

import java.io.FileReader;

import java.io.IOException;

public class Main {

private static Scanner scanner = new Scanner(System.in);

private static int numberOfNodes = 0;

private static int[][] matrix;

private static AdjGraph table;

private static boolean[] visit;

private static int[] dfn;

private static int count;

public static void main(String[] args) {

while(true) {

menu();

}

}

private static void menu() {

System.out.println();

System.out.println();

System.out.println("1. 建立图结构");

System.out.println("2. 邻接矩阵、邻接表转换");

System.out.println("3. 深度优先搜索");

System.out.println("4. 广度优先搜索");

System.out.println("5. 退出");

System.out.print("请输入选择：");

int choice = scanner.nextInt();

System.out.println();

System.out.println();

switch(choice) {

case 1:

buildStructure();

break;

case 2:

convertStructure();

break;

case 3:

depthFirst();

break;

case 4:

breadthFirst();

break;

case 5:

System.exit(0);

break;

default:

System.out.println("选择有误，请重新选择！");

break;

}

}

private static void buildStructure() {

System.out.print("请输入顶点个数：");

numberOfNodes = scanner.nextInt();

System.out.println("1. 邻接矩阵存储");

System.out.println("2. 邻接表存储");

System.out.print("请输入选择：");

int choice = scanner.nextInt();

switch(choice) {

case 1:

buildMatrix();

break;

case 2:

buildTable();

break;

default:

System.out.println("选择有误！");

break;

}

}

private static void buildMatrix() {

matrix = new int[numberOfNodes][numberOfNodes];

BufferedReader reader = null;

try {

reader = new BufferedReader(new FileReader("data.txt"));

String line = null;

String[] elementStr = null;

int[] elementInt = new int[2];

while((line = reader.readLine()) != null) {

elementStr = line.split(",");

elementInt[0] = Integer.parseInt(elementStr[0]);

elementInt[1] = Integer.parseInt(elementStr[1]);

matrix[elementInt[0]][elementInt[1]] = 1;

matrix[elementInt[1]][elementInt[0]] = 1;

}

} catch(Exception e) {

System.out.println("读取文件错误！请检查文件！");

e.printStackTrace();

} finally {

if(reader != null) {

try {

reader.close();

} catch (IOException e) {

e.printStackTrace();

}

}

}

System.out.println("邻接矩阵建立完成！邻接矩阵如下：");

showMatrix();

}

private static void buildTable() {

table = new AdjGraph();

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

table.addVertex(new VertexNode(i));

}

BufferedReader reader = null;

try {

reader = new BufferedReader(new FileReader("data.txt"));

String line = null;

String[] elementStr = null;

int[] elementInt = new int[2];

while((line = reader.readLine()) != null) {

elementStr = line.split(",");

elementInt[0] = Integer.parseInt(elementStr[0]);

elementInt[1] = Integer.parseInt(elementStr[1]);

if(table.getVertex(elementInt[0]).getFirstEdge() == null) {

table.getVertex(elementInt[0]).setFirstEdge(new EdgeNode(elementInt[1]));

} else {

EdgeNode tempNode = table.getVertex(elementInt[0]).getFirstEdge();

while(tempNode.getNext() != null) {

tempNode = tempNode.getNext();

}

tempNode.setNext(new EdgeNode(elementInt[1]));

}

if(table.getVertex(elementInt[1]).getFirstEdge() == null) {

table.getVertex(elementInt[1]).setFirstEdge(new EdgeNode(elementInt[0]));

} else {

EdgeNode tempNode = table.getVertex(elementInt[1]).getFirstEdge();

while(tempNode.getNext() != null) {

tempNode = tempNode.getNext();

}

tempNode.setNext(new EdgeNode(elementInt[0]));

}

}

} catch (Exception e) {

System.out.println("读取文件错误！请检查文件！");

e.printStackTrace();

} finally {

if(reader != null) {

try {

reader.close();

} catch (IOException e) {

e.printStackTrace();

}

}

}

System.out.println("邻接表建立完成！邻接表如下：");

showTable();

}

private static void convertStructure() {

System.out.println("1. 邻接矩阵转邻接表");

System.out.println("2. 邻接表转邻接矩阵");

System.out.print("请输入选择：");

int choice = scanner.nextInt();

switch(choice) {

case 1:

convertToTable();

break;

case 2:

convertToMatrix();

break;

default:

System.out.println("选择有误！");

break;

}

}

private static void convertToTable() {

if(matrix == null) {

System.out.println("暂无邻接矩阵数据！");

return;

}

table = new AdjGraph();

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

table.addVertex(new VertexNode(i));

}

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

for(int j = i; j < numberOfNodes; j ++) {

if(matrix[i][j] == 1) {

if(table.getVertex(i).getFirstEdge() == null) {

table.getVertex(i).setFirstEdge(new EdgeNode(j));

} else {

EdgeNode tempNode = table.getVertex(i).getFirstEdge();

while(tempNode.getNext() != null) {

tempNode = tempNode.getNext();

}

tempNode.setNext(new EdgeNode(j));

}

if(table.getVertex(j).getFirstEdge() == null) {

table.getVertex(j).setFirstEdge(new EdgeNode(i));

} else {

EdgeNode tempNode = table.getVertex(j).getFirstEdge();

while(tempNode.getNext() != null) {

tempNode = tempNode.getNext();

}

tempNode.setNext(new EdgeNode(i));

}

}

}

}

System.out.println("转换完成！");

showTable();

}

private static void convertToMatrix() {

if(table == null) {

System.out.println("暂无邻接表数据！");

return;

}

matrix = new int[numberOfNodes][numberOfNodes];

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

VertexNode tempNode = table.getVertex(i);

EdgeNode tempEdgeNode = tempNode.getFirstEdge();

while(tempEdgeNode != null) {

matrix[i][tempEdgeNode.getAdjvex()] = 1;

matrix[tempEdgeNode.getAdjvex()][i] = 1;

tempEdgeNode = tempEdgeNode.getNext();

}

}

System.out.println("转换完成！");

showMatrix();

}

private static void depthFirst() {

System.out.println("1. 对邻接矩阵进行深搜（递归）");

System.out.println("2. 对邻接矩阵进行深搜（非递归）");

System.out.println("3. 对邻接表进行深搜（递归）");

System.out.println("4. 对邻接表进行深搜（非递归）");

System.out.print("请输入选择：");

int choice = scanner.nextInt();

switch(choice) {

case 1:

recursiveMatrixDFS();

break;

case 2:

iterativeMatrixDFS();

break;

case 3:

recursiveTableDFS();

break;

case 4:

iterativeTableDFS();

break;

default:

System.out.println("选择有误！");

break;

}

}

private static void recursiveMatrixDFS() {

if(matrix == null) {

System.out.println("暂无邻接矩阵数据！");

return;

}

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

visit = new boolean[numberOfNodes];

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

visit[j] = false;

}

dfn = new int[numberOfNodes];

count = 0;

DFS2(i);

System.out.println();

}

System.out.println("深搜完成，DFS序列如上");

}

private static void DFS2(int i) {

System.out.print(i + " ");

visit[i] = true;

dfn[i] = count;

count ++;

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

if(matrix[i][j] == 1 && !visit[j]) {

DFS2(j);

}

}

}

private static void iterativeMatrixDFS() {

if(matrix == null) {

System.out.println("暂无邻接矩阵数据！");

return;

}

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

Stack<Integer> stack = new Stack<>();

visit = new boolean[numberOfNodes];

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

visit[j] = false;

}

stack.push(i);

visit[i] = true;

while(!stack.isEmpty()) {

int temp = stack.pop();

System.out.print(temp + " ");

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

if(matrix[temp][j] == 1 && !visit[j]) {

stack.push(j);

visit[j] = true;

}

}

}

System.out.println();

}

System.out.println("深搜完成，DFS序列如上");

}

private static void recursiveTableDFS() {

if(table == null) {

System.out.println("暂无邻接表数据！");

return;

}

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

visit = new boolean[numberOfNodes];

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

visit[j] = false;

}

dfn = new int[numberOfNodes];

count = 0;

DFS1(i);

System.out.println();

}

System.out.println("深搜完成，DFS序列如上");

}

private static void DFS1(int i) {

System.out.print(i + " ");

visit[i] = true;

dfn[i] = count ++;

EdgeNode p = table.getVertex(i).getFirstEdge();

while(p != null) {

if(!visit[p.getAdjvex()]) {

DFS1(p.getAdjvex());

}

p = p.getNext();

}

}

private static void iterativeTableDFS() {

if(table == null) {

System.out.println("暂无邻接表数据！");

return;

}

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

Stack<Integer> stack = new Stack<>();

visit = new boolean[numberOfNodes];

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

visit[j] = false;

}

stack.push(i);

visit[i] = true;

while(!stack.isEmpty()) {

VertexNode tempNode = table.getVertex(stack.pop());

System.out.print(tempNode.getVertex() + " ");

EdgeNode tempEdge = tempNode.getFirstEdge();

while(tempEdge != null) {

if(!visit[tempEdge.getAdjvex()]) {

stack.push(tempEdge.getAdjvex());

visit[tempEdge.getAdjvex()] = true;

}

tempEdge = tempEdge.getNext();

}

}

System.out.println();

}

System.out.println("深搜完成，DFS序列如上");

}

private static void breadthFirst() {

System.out.println("1. 邻接矩阵的广度优先搜索");

System.out.println("2. 邻接表的广度优先搜索");

System.out.print("请输入选择：");

int choice = scanner.nextInt();

switch(choice) {

case 1:

matrixBFS();

break;

case 2:

tableBFS();

break;

default:

System.out.println("选择有误！");

break;

}

}

private static void matrixBFS() {

if(matrix == null) {

System.out.println("暂无邻接矩阵数据！");

return;

}

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

Queue<Integer> queue = new Queue<>();

visit = new boolean[numberOfNodes];

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

visit[j] = false;

}

System.out.print(i + " ");

visit[i] = true;

queue.enQueue(i);

while(!queue.isEmpty()) {

int temp = queue.deQueue();

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

if(matrix[temp][j] == 1 && !visit[j]) {

System.out.print(j + " ");

visit[j] = true;

queue.enQueue(j);

}

}

}

System.out.println();

}

System.out.println("广搜完成，BFS序列如上");

}

private static void tableBFS() {

if(table == null) {

System.out.println("暂无邻接表数据！");

return;

}

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

Queue<Integer> queue = new Queue<>();

visit = new boolean[numberOfNodes];

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

visit[j] = false;

}

System.out.print(i + " ");

visit[i] = true;

queue.enQueue(i);

while(!queue.isEmpty()) {

EdgeNode tempNode = table.getVertex(queue.deQueue()).getFirstEdge();

while(tempNode != null) {

if(!visit[tempNode.getAdjvex()]) {

System.out.print(tempNode.getAdjvex() + " ");

visit[tempNode.getAdjvex()] = true;

queue.enQueue(tempNode.getAdjvex());

}

tempNode = tempNode.getNext();

}

}

System.out.println();

}

System.out.println("广搜完成，BFS序列如上");

}

private static void showMatrix() {

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

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

System.out.print(matrix[i][j] + " ");

}

System.out.println();

}

}

private static void showTable() {

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

System.out.print(i + " => ");

EdgeNode tempEdge = table.getVertex(i).getFirstEdge();

while(tempEdge.getNext() != null) {

System.out.print(tempEdge.getAdjvex() + " -> ");

tempEdge = tempEdge.getNext();

}

System.out.println(tempEdge.getAdjvex());

}

}

}

class EdgeNode {

private int adjvex;

private EdgeNode next;

public EdgeNode(int adjvex) {

this.adjvex = adjvex;

next = null;

}

public int getAdjvex() {

return adjvex;

}

public void setNext(EdgeNode next) {

this.next = next;

}

public EdgeNode getNext() {

return next;

}

}

class VertexNode {

private int vertex;

private EdgeNode firstEdge;

public VertexNode(int vertex) {

this.vertex = vertex;

}

public int getVertex() {

return vertex;

}

public void setFirstEdge(EdgeNode firstEdge) {

this.firstEdge = firstEdge;

}

public EdgeNode getFirstEdge() {

return firstEdge;

}

}

class AdjGraph {

private ArrayList<VertexNode> vexList;

public AdjGraph() {

vexList = new ArrayList<>();

}

public void addVertex(VertexNode vertex) {

vexList.add(vertex);

}

public VertexNode getVertex(int index) {

return vexList.get(index);

}

}

class Stack<T> {

private ArrayList<T> list;

public Stack() {

list = new ArrayList<>();

}

public void push(T element) {

list.add(element);

}

public T pop() {

return list.remove(list.size() - 1);

}

public T getTop() {

return list.get(list.size() - 1);

}

public boolean isEmpty() {

return list.isEmpty();

}

}

class Queue<T> {

private ArrayList<T> list;

public Queue() {

list = new ArrayList<>();

}

public void enQueue(T element) {

list.add(element);

}

public T deQueue() {

return list.remove(0);

}

public T getFirst() {

return list.get(0);

}

public boolean isEmpty() {

return list.isEmpty();

}

}