《数据结构》实验报告

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 姓名 | 马昆 | 班级 | 22级软件工程专升本1班 | 学号 | 2206831544 |
| 实验名称 | **实验六 图** | | | | |
| 实验时间 | 2023年5月7日 | | 成绩 | |  |
| 1. **实验目的：**   1.理解图的逻辑结构及存储结构；  2.掌握图的基本操作；  3.掌握图的深度优先和广度优先遍历算法。  **二、实验工具：**  软件平台：Windows7或以上版本，Visual Studio 2019  **三、实验原理：**  1、算法描述：  （1）深度优先遍历函数：函数无返回值，并接收一个整型，表示需要遍历的顶点的下标。首先根据实参打印当前遍历顶点的值，然后将全局的遍历标记数组visited对应的顶点位置赋值为1，表示已经遍历过了。定义一个for循环，循环次数为顶点的个数，循环体内根据边的二维数组找到所有与当前遍历顶点有关系的顶点，并且如果这个顶点没有被遍历过就迭代调用自身。  （2）广度优先遍历函数：函数无返回值，并接收一个整型，表示需要遍历的顶点的下标。定义一个整型数组，数组长度为图中最大顶点数，作为存放遍历顶点下标的顺序队列，同时定义两个整型，用于记录顺序队列的头和尾，并初始化为-1，表示队列为空。定义两个临时变量，用于存放遍历时需要暂时存放的顶点下标。随后打印当前遍历的顶点的值，并且将全局的遍历标记数组visited对应的顶点位置赋值为1，表示已经遍历过了。随后将遍历的顶点的下标赋值给顺序队列的队尾，并且队尾指针往后移动一位。定义一个while循环，条件为头指针不等于尾指针，即队列非空时执行，循环体内先将队头元素出队，然后定义for循环，循环次数为顶点的个数，根据边数组使用for循环体内获取本次while循环出队的下标元素对应的顶点所关联的所有顶点，并判断这些顶点是否被遍历过，如果没有被遍历过，就在for循环体内输出此关联顶点对应的数据，并将全局的遍历标记数组visited对应的顶点位置赋值为1。然后再将此关联顶点入队。  **四、实验步骤和内容：**  1. 1.建立任意无向图，采用邻接矩阵存储，完成以下操作：  （1）对该图进行深度优先遍历，并输出遍历结果；  （2）对该图进行广度优先遍历，并输出遍历结果。   |  | | --- | | **MGraph.h**  #pragma once  #include <iostream>  using namespace std;  const int MaxSize = 10; // 图中最多顶点数  int visited[MaxSize] = { 0 }; // 全局数组变量visited初始化  template <typename DataType>  class MGraph  {  public:  MGraph(DataType a[], int n, int e); // 构造函数  void DFTraverse(int v); // 深度优先遍历图  void BFTraverse(int v); // 广度优先遍历图  private:  DataType vertex[MaxSize]; // 存放图中顶点的数组  int edge[MaxSize][MaxSize]; // 存放图中边的数组  int vertexNum, edgeNum; // 图的顶点数和边数  };  // 无向图邻接矩阵的构造函数  template <typename DataType>  MGraph<DataType>::MGraph(DataType a[], int n, int e)  {  int i, j, k;  vertexNum = n; edgeNum = e;  for (i = 0; i < vertexNum; i++)  {  vertex[i] = a[i]; // 存储顶点  }  for (i = 0; i < vertexNum; i++) // 初始化邻接矩阵  {  for (j = 0; j < vertexNum; j++)  {  edge[i][j] = 0;  }  }  for (k = 0; k < edgeNum; k++) // 依次输入每一条边  {  cin >> i >> j; // 输入边依附的两个顶点的编号  edge[i][j] = 1;  edge[j][i] = 1; // 置有边标志  }  }  // 深度优先遍历  template<typename DataType>  void MGraph<DataType>::DFTraverse(int v)  {  cout << vertex[v] << '\t';  visited[v] = 1;  for (int i = 0; i < vertexNum; i++)  {  if (edge[v][i] == 1 && visited[i] == 0) DFTraverse(i);  }  }  // 广度优先遍历  template <typename DataType>  void MGraph<DataType>::BFTraverse(int v)  {  int Q[MaxSize], front = -1, rear = -1; // 采用顺序队列并初始化队列  int w, j; // 临时变量  cout << vertex[v] << '\t';  visited[v] = 1;  Q[++rear] = v; // 被访问顶点入队  while (front != rear) // 当队列非空时  {  w = Q[++front]; // 将队头元素出队并送到v中  for (j = 0; j < vertexNum; j++)  {  if (edge[w][j] == 1 && visited[j] == 0)  {  cout << vertex[j] << '\t';  visited[j] = 1;  Q[++rear] = j;  }  }  }  } | | **Main.cpp**  #include "MGraph.h"  int main()  {  int i;  char ch[] = { 'A', 'B', 'C', 'D', 'E' };  MGraph<char> MG{ ch, 5, 6 }; // 建立具有5个顶点，6条边的无向图  for (i = 0; i < MaxSize; i++)  {  visited[i] = 0;  }  cout << "深度优先遍历序列是：" << endl;  MG.DFTraverse(0);  for (i = 0; i < MaxSize; i++)  {  visited[i] = 0;  }  cout << "\n广度优先遍历序列是：" << endl;  MG.BFTraverse(0);  return 0;  } | | **运行结果**  **案例1**      **案例2** |   **五、实验总结：**  通过本次实验了解了无向图的一种存储实现，以及两种遍历方法的实现，广度优先遍历与深度优先遍历。对无向图的存储实现还是比较容易理解的，但是这两种遍历方法还是有些模糊，学习得不够透彻。代码的实现是照着书上写的，感觉书上的代码耦合度很高，main函数中的代码与有向图数据结构的类高度耦合，打印函数也写在了图的数据结构类中，感觉不是很好，不过无向图的实现思路很值得学习。  **六、教师评语：** | | | | | |