|  |  |  |
| --- | --- | --- |
| **软件学院实验报告** | | |
| 姓名： 胡逢彬 学号：2225060803 专业：网络工程 年级：2022级 | | |
| 课程名称 | | 数据结构 |
| 实验名称 | | 实验9、图的遍历 |
| 实验的准备阶段 | 实验内容 | （1）实验目的  通过该实验，使学生掌握图的几种存储结构，理解图的深度优先和广度优先遍历算法的思想和实现办法，  （2）实验内容  实现教材算法7.2利用邻接矩阵构造无向图的算法，在此基础上进行深度优先遍历和广度优先遍历。  （3）参考界面    1  2  3  4  5  6  （4）验收/测试用例   * 创建所示无向图   屏幕输出邻接矩阵  0 1 0 0 0 1  1 0 1 1 0 0  0 1 0 1 1 0  0 1 1 0 1 0  0 0 1 1 0 1  1 0 0 0 1 0   * 深度优先遍历   屏幕输出： 1 2 3 4 5 6   * 广度优先遍历   屏幕输出：1 2 6 3 4 5   * 另外：自己画出一个无向图，再测试一遍。 |
| 实验类型 | 验证性 |
| 实验的重点、难点 | 重点： 图遍历  难点： 图的深度优先遍历 |
| 实验环境 | VC++6.0 |
| 实验的实施阶段 | 实验步骤及完成任务情况 | 一、设计思想  我们的目标是实现图的深度优先遍历和广度优先遍历算法，首先需要构建图的存储结构。这里我们选择使用邻接矩阵来表示无向图。邻接矩阵是一个二维数组，数组元素表示图中顶点之间的连接关系。  接下来，我们需要实现深度优先遍历和广度优先遍历算法。深度优先遍历可以使用递归或者栈来实现，它会尽可能深地访问图的各个分支。而广度优先遍历则使用队列来实现，按照层次逐步访问图中的各个顶点。  最后，我们可以为用户提供选择菜单，让用户可以选择构建无向图、输出邻接矩阵、进行深度优先遍历、广度优先遍历以及退出程序等功能。  通过以上实现，用户可以通过操作菜单来执行相应功能，从而加深对图的存储结构和遍历算法的理解。  二、主要源代码  #include <iostream>  #include <vector>  #include <queue>  using namespace std;  void addEdge(vector<vector<int>>& adjacencyMatrix, int src, int dest) {  adjacencyMatrix[src][dest] = 1;  adjacencyMatrix[dest][src] = 1;  }  void displayMatrix(const vector<vector<int>>& adjacencyMatrix) {  cout << "邻接矩阵：" << endl;  for (const vector<int>& row : adjacencyMatrix) {  for (int val : row) {  cout << val << " ";  }  cout << endl;  }    }  void DFSUtil(int vertex, vector<bool>& visited, const vector<vector<int>>& adjacencyMatrix) {  visited[vertex] = true;  cout << vertex << " ";  for (int i = 0; i < adjacencyMatrix.size(); ++i) {  if (adjacencyMatrix[vertex][i] && !visited[i]) {  DFSUtil(i, visited, adjacencyMatrix);  }  }  }  void DFS(const vector<vector<int>>& adjacencyMatrix) {  int startVertex;  cout << "输入起始顶点进行深度优先遍历：";  cin >> startVertex;  vector<bool> visited(adjacencyMatrix.size(), false);  cout << "深度优先遍历结果：";  DFSUtil(startVertex, visited, adjacencyMatrix);  cout << endl;  }  void BFS(const vector<vector<int>>& adjacencyMatrix) {  int startVertex;  cout << "输入起始顶点进行广度优先遍历：";  cin >> startVertex;  vector<bool> visited(adjacencyMatrix.size(), false);  queue<int> queue;  visited[startVertex] = true;  queue.push(startVertex);  cout << "广度优先遍历结果：";  while (!queue.empty()) {  int vertex = queue.front();  cout << vertex << " ";  queue.pop();  for (int i = 0; i < adjacencyMatrix.size(); ++i) {  if (adjacencyMatrix[vertex][i] && !visited[i]) {  visited[i] = true;  queue.push(i);  }  }  }  cout << endl;  }  int main() {  int numVertices, choice;  cout << "输入图中顶点的数量：";  cin >> numVertices;  vector<vector<int>> adjacencyMatrix(numVertices, vector<int>(numVertices, 0));  while (true) {  cout << "---------- 菜单 ----------" << endl;  cout << "1. 添加边" << endl;  cout << "2. 显示邻接矩阵" << endl;  cout << "3. 深度优先遍历" << endl;  cout << "4. 广度优先遍历" << endl;  cout << "5. 退出程序" << endl;  cout << "请输入您的选择：";  cin >> choice;  switch (choice) {  case 1: {  int src, dest;  cout << "请输入边的起始顶点和目标顶点：";  cin >> src >> dest;  addEdge(adjacencyMatrix, src, dest);  break;  }  case 2: {  displayMatrix(adjacencyMatrix);  break;  }  case 3: {  DFS(adjacencyMatrix);  break;  }  case 4: {  BFS(adjacencyMatrix);  break;  }  case 5: {  cout << "正在退出程序..." << endl;  return 0;  }  default: {  cout << "无效选择，请重新输入。" << endl;  }  }  cout << endl;  }  } |
| 实验结果的处理阶段 | 实验结果 | IMG_256  IMG_256 |
| 实验结果总结 |  |