|  |  |  |  |
| --- | --- | --- | --- |
| Algorithm and Data Structure Experimental Assignment | | | |
| Class | 17 CS 01-02 | Teacher Name | Mano and Hong Zhang |
| Student Name | 张钧豪 | Submitted Time | 2019.11.26 |
| Assignment Topic | Graph Traversal | | |
| Assignment Requirement:   1. Develop a recursive program that accomplishes depth-first traversal. Your program should take as input an adjacency list, and output the node labels in the order of traversal. Use your program to verify your answer to the depthfirst traversal question above. 4. Develop a program that accomplishes breadth-first traversal. Your program should take as input an adjacency list, and output the node labels in the order. 2. Develop a program that accomplishes breadth-first traversal. Your program should take as input an adjacency list, and output the node labels in the order of traversal. Use your program to verify your answer to the breadth-first traversal question above. | | | |
| Code of Main Functions:  #include <iostream>  #include <vector>  #include <stack>  #include <queue>  using namespace std;  //typedef int DATA\_TYPE; // 权值为int型  //邻接表  struct AdjTableGraph  {  vector<vector<int> > adjTable;  vector<vector<int> > adjWeights;  };  void AdjTableDFS(AdjTableGraph graph, vector<int>& visited, vector<int>& visitOrder, int startNode)  {  visited[startNode] = 1;  visitOrder.push\_back(startNode);  if (graph.adjTable[startNode].size() > 0)  {  for (size\_t i = 0; i < graph.adjTable[startNode].size(); ++i)  {  if (visited[graph.adjTable[startNode][i]] == 0)  {  AdjTableDFS(graph, visited, visitOrder, graph.adjTable[startNode][i]);  }  }  }  }  vector<int> AdjTableBFS(AdjTableGraph graph, int startNode)  {  int vertexNum = graph.adjTable.size();  vector<int> visited(vertexNum, 0);  vector<int> visitOrder;  queue<int> trace;  trace.push(startNode);  visited[startNode] = 1;  while (!trace.empty())  {  int currentNode = trace.front();  trace.pop();  visitOrder.push\_back(currentNode);  if (graph.adjTable[currentNode].size() > 0)  {  for (size\_t i = 0; i < graph.adjTable[currentNode].size(); ++i)  {  if (visited[graph.adjTable[currentNode][i]] == 0)  {  trace.push(graph.adjTable[currentNode][i]);  visited[graph.adjTable[currentNode][i]] = 1;  }  }  }  }  return visitOrder;  }  vector<int> DFSBFS\_next(AdjTableGraph graph, int startNode)  {  int nodeNum = graph.adjTable.size();  int currentNode = startNode;  vector<int> visited (nodeNum, 0);  vector<int> visitOrder;  queue<int> q;  q.push(startNode);  visited[startNode] = 1;  while (!q.empty()) {  if (graph.adjTable[currentNode].size() > 0) {  for (int i = 0; i < graph.adjTable[currentNode].size(); i++) {  if (visited[graph.adjTable[currentNode][i]] != 1) {  currentNode = graph.adjTable[currentNode][i];  }  }  }  }  return visitOrder;  } | | | |
| Experimental Results:    C:\Users\Acer\Documents\Tencent Files\862231364\Image\C2C\{362C7A40-425B-AACE-CA7B-6176AF18E67C}.png  C:\Users\Acer\Documents\Tencent Files\862231364\Image\C2C\{4AC622D0-06CB-6CE3-A020-9EEF079B34B3}.png | | | |