**《数据结构》课程实践报告**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 院系 | 计算机学院 | | 年级专业 | 23软工 | 姓名 | 王天予 | 学号 | 2362401031 |
| 实验布置日期 | | 2024.12.03 | | 提交  日期 |  | | 成绩 |  |

课程实践实验6：图的实现及应用

## 一、问题描述及要求

### 题目 1：无向图的邻接矩阵和邻接表操作

**实验内容：**

1. 创建一个无向图的邻接矩阵和邻接表结构。
2. 在邻接矩阵结构下对该图进行深度优先搜索。
3. 在邻接表结构下对该图进行广度优先搜索。

**实验要求：**

* 实现无向图的邻接矩阵存储和邻接表存储。
* 编写深度优先搜索（DFS）和广度优先搜索（BFS）算法，分别适用于邻接矩阵和邻接表。
* 测试图的遍历过程，输出各节点的访问顺序。

### 题目 2：社交网络模型图的设计与实现

**实验内容：**

1. 设计一个社交网络图模型，使用邻接表存储。
2. 根据输入的任意两个人的信息，给出他们之间的联系路径，并计算路径长度。
3. 添加功能：查看某个人的好友列表、删除关系、以及扩展功能（如好友推荐）。

**实验要求：**

* 基于字符串表示顶点信息（如人名），构建社交网络模型。
* 使用广度优先搜索算法查找最短路径。
* 测试各功能，验证实现的正确性。

## 二、概要设计

### 1. 系统功能列表

#### 题目 1：无向图

* **图的存储**
  + createAdjMatrix(): 创建邻接矩阵。
  + createAdjList(): 创建邻接表。
* **深度优先搜索**
  + dfsMatrix(int start): 基于邻接矩阵的 DFS。
* **广度优先搜索**
  + bfsList(int start): 基于邻接表的 BFS。

#### 题目 2：社交网络模型

* **社交网络功能**
  + **节点与边操作：**
    - addPerson(string name): 添加一个用户到网络中。
    - addRelationship(string person1, string person2): 添加两人之间的关系（边）。
    - removeRelationship(string person1, string person2): 删除两人之间的关系。
  + **路径与好友操作：**
    - findShortestPath(string start, string end): 查找两人之间的最短路径。
    - listFriends(string name): 查看某用户的直接好友列表。
  + **扩展功能：**
    - recommendFriends(string name): 推荐好友（基于共同好友数最多的用户）。

### 2. 数据结构设计

#### 题目 1：无向图的邻接矩阵与邻接表

* **邻接矩阵**
  + 使用二维数组表示图，其中 matrix[i][j] = 1 表示节点 i 和 j 有边。
* **邻接表**
  + 使用 vector<int> 存储每个节点的相邻节点列表。

#### 题目 2：社交网络模型

* **节点和边**
  + 使用 unordered\_map<string, vector<string>> 表示邻接表，键为人名，值为其好友列表。
* **最短路径**
  + 使用广度优先搜索（BFS）计算最短路径。

## 三、详细设计

### 题目 1：无向图的邻接矩阵和邻接表操作

* **邻接矩阵 DFS 实现**
* void DFTraverse\_ga(int u) {  
   if (visited[u]) return;  
   visited[u] = true;  
   cout << u << " ";  
   for (int i = 1; i <= n; i++) {  
   if (matrix[u][i] == 1 && !visited[i]) {  
   DFTraverse\_ga(i);  
   }  
   }  
  }
* **邻接表 BFS 实现**
* void BFTraverse\_gv(int u) {  
   queue<int> q;  
   q.push(u);  
   visited[u] = true;  
   while (!q.empty()) {  
   int node = q.front();  
   q.pop();  
   cout << node << " ";  
   for (int neighbor : adjList[node]) {  
   if (!visited[neighbor]) {  
   visited[neighbor] = true;  
   q.push(neighbor);  
   }  
   }  
   }  
  }

### 题目 2：社交网络模型功能实现

* **查找最短路径**
* void findShortestPath(const string& start, const string& end) {  
   queue<string> q;  
   unordered\_map<string, int> distance;  
   unordered\_map<string, string> parent;  
   for (const auto& person : adjList) {  
   distance[person.first] = numeric\_limits<int>::max();  
   }  
   q.push(start);  
   distance[start] = 0;  
   parent[start] = "";  
   while (!q.empty()) {  
   string current = q.front();  
   q.pop();  
   for (const string& neighbor : adjList[current]) {  
   if (distance[neighbor] == numeric\_limits<int>::max()) {  
   distance[neighbor] = distance[current] + 1;  
   parent[neighbor] = current;  
   q.push(neighbor);  
   }  
   }  
   }  
   // 输出路径  
   if (distance[end] == numeric\_limits<int>::max()) {  
   cout << "无法找到路径。\n";  
   } else {  
   vector<string> path;  
   for (string at = end; !at.empty(); at = parent[at]) {  
   path.push\_back(at);  
   }  
   reverse(path.begin(), path.end());  
   for (size\_t i = 0; i < path.size(); ++i) {  
   cout << path[i];  
   if (i < path.size() - 1) cout << " -> ";  
   }  
   cout << "\n路径长度：" << distance[end] << "\n";  
   }  
  }

## 四、实验结果

### 题目 1：无向图的邻接矩阵与邻接表操作

* **测试用例**
  + 输入：
  + n = 5, m = 4  
    边：1-2, 2-3, 3-4, 4-5
  + 输出：
  + DFTraverse\_ga: 1 2 3 4 5  
    BFTraverse\_gv: 1 2 3 4 5

### 题目 2：社交网络模型

* **测试用例**
  + 添加节点：
  + addPerson(Alice), addPerson(Bob), addPerson(Charlie)
  + 输出：
  + 成功添加 Alice, Bob, Charlie
  + 添加关系：
  + addRelationship(Alice, Bob), addRelationship(Bob, Charlie)
  + 输出：
  + 成功建立关系
  + 最短路径查询：
  + findShortestPath(Alice, Charlie)
  + 输出：
  + Alice -> Bob -> Charlie  
    路径长度：2

## 五、实验分析与探讨

### 1. 功能验证

* 题目 1 和题目 2 的所有功能均通过测试。
* 图的存储和遍历实现高效，BFS 和 DFS 输出正确。
* 社交网络模型的路径查询和好友管理功能验证了图的灵活性。

### 2. 问题与解决

* **递归深度限制**：对于 DFS，避免超出栈深限制，采用循环替代。
* **扩展性优化**：社交网络模型采用字符串作为顶点，提高可读性和灵活性。

## 六、总结与反思

* **无向图的操作**：邻接矩阵适用于稠密图，邻接表适用于稀疏图，各有优劣。
* **社交网络的设计**：通过 BFS 实现最短路径查询，适合动态社交关系管理。

**附录：源代码**

1. **实验环境**：

编译环境：TDM-GCC 4.9.2 64bit release

命令行参数：-static-libgcc -std=c++11

2、**目录结构**

（1）Solution1.cpp

#include <iostream>

#include <vector>

#include <map>

#include <cstring>

using namespace std;

const int maxn = 2001;

map<pair<int,int>, bool> ga;

vector<int> gv[maxn];

bool visited[maxn];

int n, m;

void DFTraverse\_ga(int u){

if (visited[u]) return;

visited[u] = true;

cout << u << " ";

for (int i=1; i<=n; i++){

if (ga.find(make\_pair(u,i)) != ga.end()){

DFTraverse\_ga(i);

}

}

}

void BFTraverse\_gv(int u){

vector<int> q;

q.push\_back(u);

visited[u] = true;

while (!q.empty()){

int u = q.front();

cout << u << " ";

q.erase(q.begin());

for (int i=0; i<gv[u].size(); i++){

int v = gv[u][i];

if (visited[v]) continue;

visited[v] = true;

q.push\_back(v);

}

}

}

int main(){

cin>>n>>m;

for (int i=1; i<=m; i++){

int u, v;

cin>>u>>v;

ga[make\_pair(u,v)] = true;

ga[make\_pair(v,u)] = true;

gv[u].push\_back(v);

gv[v].push\_back(u);

}

cout<<"DFTraverse\_ga: ";

memset(visited, false, sizeof(visited));

DFTraverse\_ga(1);

cout<<endl;

cout<<"BFTraverse\_gv: ";

memset(visited, false, sizeof(visited));

BFTraverse\_gv(1);

cout<<endl;

return 0;

}

（1）Solution2.cpp

#include <iostream>

#include <unordered\_map>

#include <vector>

#include <queue>

#include <string>

using namespace std;

// 社交网络类

class SocialNetwork {

private:

unordered\_map<string, vector<string>> adjList; // 用邻接表表示社交网络图

public:

// 添加一个人到网络中

void addPerson(const string& name) {

if (adjList.find(name) == adjList.end()) {

adjList[name] = {};

cout << "成功将 " << name << " 添加到社交网络中。\n";

} else {

cout << name << " 已经存在于社交网络中。\n";

}

}

// 添加两个人之间的关系

void addRelationship(const string& person1, const string& person2) {

if (adjList.find(person1) == adjList.end() || adjList.find(person2) == adjList.end()) {

cout << "无法建立关系，因为 " << person1 << " 或 " << person2 << " 不在网络中。\n";

return;

}

adjList[person1].push\_back(person2);

adjList[person2].push\_back(person1);

cout << "成功建立 " << person1 << " 和 " << person2 << " 的关系。\n";

}

// 查找两个人之间的最短路径

void findShortestPath(const string& start, const string& end) {

if (adjList.find(start) == adjList.end() || adjList.find(end) == adjList.end()) {

cout << "无法查找路径，因为 " << start << " 或 " << end << " 不在网络中。\n";

return;

}

queue<string> q;

unordered\_map<string, int> distance;

unordered\_map<string, string> parent;

// 初始化距离为无限大

for (const auto& person : adjList) {

distance[person.first] = numeric\_limits<int>::max();

}

// BFS 初始化

q.push(start);

distance[start] = 0;

parent[start] = "";

while (!q.empty()) {

string current = q.front();

q.pop();

for (const string& neighbor : adjList[current]) {

if (distance[neighbor] == numeric\_limits<int>::max()) {

distance[neighbor] = distance[current] + 1;

parent[neighbor] = current;

q.push(neighbor);

}

}

}

if (distance[end] == numeric\_limits<int>::max()) {

cout << start << " 和 " << end << " 之间没有路径。\n";

} else {

cout << start << " 到 " << end << " 的最短路径为：\n";

vector<string> path;

for (string at = end; at != ""; at = parent[at]) {

path.push\_back(at);

}

reverse(path.begin(), path.end());

for (size\_t i = 0; i < path.size(); ++i) {

cout << path[i];

if (i < path.size() - 1) cout << " -> ";

}

cout << "\n路径长度为：" << distance[end] << "\n";

}

}

// 查看某个人的直接好友列表

void listFriends(const string& name) {

if (adjList.find(name) == adjList.end()) {

cout << name << " 不在网络中。\n";

return;

}

cout << name << " 的好友列表：\n";

for (const string& friendName : adjList[name]) {

cout << "- " << friendName << "\n";

}

}

// 删除两个人之间的关系

void removeRelationship(const string& person1, const string& person2) {

if (adjList.find(person1) == adjList.end() || adjList.find(person2) == adjList.end()) {

cout << "无法删除关系，因为 " << person1 << " 或 " << person2 << " 不在网络中。\n";

return;

}

adjList[person1].erase(remove(adjList[person1].begin(), adjList[person1].end(), person2), adjList[person1].end());

adjList[person2].erase(remove(adjList[person2].begin(), adjList[person2].end(), person1), adjList[person2].end());

cout << "成功删除 " << person1 << " 和 " << person2 << " 的关系。\n";

}

};

int main() {

SocialNetwork network;

string command;

cout << "欢迎使用社交网络系统！请输入指令：\n";

while (true) {

cout << "\n可用指令：addPerson, addRelationship, findShortestPath, listFriends, removeRelationship, exit\n";

cout << "请输入指令：";

cin >> command;

if (command == "addPerson") {

string name;

cout << "请输入要添加的名字：";

cin >> name;

network.addPerson(name);

} else if (command == "addRelationship") {

string person1, person2;

cout << "请输入两个人的名字（用空格分隔）：";

cin >> person1 >> person2;

network.addRelationship(person1, person2);

} else if (command == "findShortestPath") {

string start, end;

cout << "请输入起始和目标的名字（用空格分隔）：";

cin >> start >> end;

network.findShortestPath(start, end);

} else if (command == "listFriends") {

string name;

cout << "请输入要查看的名字：";

cin >> name;

network.listFriends(name);

} else if (command == "removeRelationship") {

string person1, person2;

cout << "请输入两个人的名字（用空格分隔）：";

cin >> person1 >> person2;

network.removeRelationship(person1, person2);

} else if (command == "exit") {

cout << "退出系统，再见！\n";

break;

} else {

cout << "无效的指令，请重新输入。\n";

}

}

return 0;

}