Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования «Московский государственный технический университет имени Н. Э. Баумана (национальный исследовательский университет)

Факультет «Робототехника и комплексная автоматизация»

Кафедра «Системы автоматизированного проектирования»

**Отчет по практической работе**

По курсу «Проектно-технологическая практика»

Выполнил: Студент Петраков С.А.

Группа РК6-36Б

Проверил: \_\_Козов А.В.\_\_\_\_\_\_\_\_\_

Дата \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Подпись\_\_\_\_\_\_\_\_\_\_\_\_\_

Москва, 2020 г.

**Вариант P11**

**Задание:**

Транспортную сеть образует конечное множество пунктов (узлов) и связывающих их магистралей (коммуникаций). Конфигурацию сети фиксирует взаимное расположение её узлов на плоскости и топология магистралей. Считается, что пункты сети не имеет размера, а магистрали состоят из ортогональных сегментов и могут пересекаться только в узлах, не имея толщины. Необходимо разработать ООП AUTONET, которая может в интерактивном режиме представлять оперативную информация для выбора маршрута между любой парой пунктов сети. В частности, все возможные пути между заданной парой пунктов с учетом необязательных ограничений, например, чтобы число промежуточных узлов каждого пути не превышало заданную величину. Исходная конфигурация сети должна представляться в алфавитно-цифровом формате, пригодном для хранения в текстовом файле. Например, для обозначения узлов можно использовать заглавные латинские буквы, а с помощью горизонтальных и вертикальных цепочек строчных латинских букв можно изображать магистрали сети. Размеры файла конфигурации сети должны быть согласованы с параметрами стандартного экрана в алфавитно-цифровом режиме. Для решения проблемы поиска путей по исходной конфигурации сети целесообразно построить топологическую модель в форме конечного графа, который задаётся матрицей смежности. Её значащие элементы должны соответствовать обозначениям магистралей сети. По матрице смежности должна быть построена матрица путей, в которой содержатся все возможные варианты маршрутов между всеми узлами сети. Её элементы можно рассматривать, как объединения конкатенаций литер, обозначающих магистрали сети. При этом каждая конкатенация соответствует одному возможному варианту маршрута, а их объединение представляет, очевидно, все варианты путей для данной пары пунктов сети. Матрица путей должна быть путем логического суммирования логических степеней матрицы смежности графа сети. Её элементы определяются по обычным формулам вычисления произведения и суммы матриц, где сложение элементов следует заменить на конкатенацию, а умножение на объединение цепочек символов. Для объектно-ориентированной программы реализации указанного подхода проектируемая программа AUTONET должна предусматривать перегрузку операций сложения и умножения логических матриц.

**Алгоритм:**

Для задачи файла конфигурации передаем его программе через аргумент командной строки.

Для решения задачи создан класс «AutoNet»

В нем присутствуют приватные поля матриц смежности и путей, массива узлов.

Приватные методы вычисления матрицы смежности и матрицы путей.

Публичные методы конструктора, вывода на экран матрицы смежности,2 метода поиска путей по матрице путей в зависимости от того есть ли ограничение на максимальное количество магистралей или нет. Метод вывода всех узлов графа.

После передаем название и путь файла в конструктор данного класса.

Конструктор осуществляет преобразование файла в конечную матрицу смежности. Для этого мы ищем все узлы нашего графа и магистрали выходящих из этих узлов. После этого перебором сопоставляем узлы матрицы в методе(calcAdjacencyMatrix).

После этого составляем матрицу путей, для этого необходимо выбрать максимальное число возможных магистралей, и передаем его в метод вычисления матрицы путей(calcPathwayMatrix).

После этого в цикле с постусловием начинаем спрашивать пользоваателя из какого и в какой пункт назначения он хочет приехать. Необходимы ли ему ограничения на длину строки. Если ограничения есть, то вызываем функцию, с дополнительным параметром ограничений. Если нет, то без этого параметра.

**Входные данные:**

Файл с конфигурацией сети. Пункты назначения являются заглавные буквы латинского алфавита(A-Z), в случае нехватки в конец названия добавляется новая буква(A,B,C…Z, AA, BA…ZA,AC…). Каждая магистраль обозначается своей строчной буквой латинского алфавита(a-z), в случае нехватки в конец добавляется цифра, либо набор цифр (a,b,c…z,a1,b1,c1…z1,a3…).

Откуда и куда едем. Возможные ограничения

**Выходные данные:**

Граф из файла

Пути - маршруты между соответствующими узлами сети.

**Текст программы:**

#include <iostream>

#include <vector>

#include <string>

#include <fstream>

#include <cstdlib>

class AutoNet

{

private:

std::vector <std::vector<std::string>> \_adjacencyMatrix;

std::vector <std::vector<std::string>> \_pathwayMatrix;

std::vector<std::string> \_nodes;

void calcPathwayMatrix(int);

void calcAdjacencyMatrix(std::vector<std::vector <std::string>>);

public:

AutoNet(std::string);

void printAdjacencyMatrix();

void findPath(std::string, std::string);

void findPath(std::string, std::string, int);

void printNodes();

};

std::string removeRepeating(std::string s)

{

int ind1 = 0, i = 0, flag = 0, n = s.size();

while (i < (n - 1))

{

if (s[i] == ' ')

{

if (flag == 1)

{

s.erase(ind1, i - ind1 + 1);

n = n - i + ind1 - 1;

i = ind1;

--i;

}

flag = 0;

ind1 = i + 1;

}

else

{

if ((s[i] >= 'a' && s[i] <= 'z') && (s[i + 1] >= 'a' && s[i + 1] <= 'z') && (s[i] == s[i + 1]))

{

flag = 1;

}

else if ((s[i] >= 'a' && s[i] <= 'z') && (s[i + 1] >= '0' && s[i + 1] <= '9'))

{

if ((i + 3 < s.size()) && ((s[i + 3] >= '0' && s[i + 3] <= '9')) && (s[i + 3] == s[i + 1]) && (s[i] == s[i + 2]))

flag = 1;

}

}

++i;

}

if (flag == 1)

{

s.erase(ind1, i - ind1 + 1);

}

return s;

}

void AutoNet::printAdjacencyMatrix()

{

std::cout << "AdjacencyMatrix" << std::endl;

std::cout << " \t";

for (int i = 0; i < \_nodes.size(); i++)

std::cout << \_nodes[i] << "\t";

std::cout << "\n";

for (int i = 0; i < \_adjacencyMatrix.size(); i++)

{

std::cout << \_nodes[i] << "\t";

for (int j = 0; j < \_adjacencyMatrix[i].size(); j++)

{

std::cout << \_adjacencyMatrix[i][j];

std::cout << "\t";

}

std::cout << "\n";

}

}

bool isEnd(std::vector<std::vector <std::string>>\* q)

{

for (int i = 0; i < (\*q).size(); ++i)

{

for (int j = 0; j < (\*q).size(); ++j)

{

if ((\*q)[i][j] != "")

return true;

}

}

return false;

}

std::vector <std::vector<std::string>> operator\*(const std::vector <std::vector<std::string>>& m1, const std::vector <std::vector<std::string>>& m2)

{

std::vector <std::vector<std::string>> result;

result.resize(m1.size());

for (int i = 0; i < m1.size(); i++)

result[i].resize(m1.size(), "");

for (int i = 0; i < result.size(); ++i)

{

for (int j = 0; j < result.size(); ++j)

{

if (i != j)

{

std::string qwerty = "";

for (int k = 0; k < result.size(); k++)

{

if (m1[i][k] != "" && m2[k][j] != "")

{

for (int u = 0; u < m1[i][k].size(); u++)

{

if (m1[i][k][u] == ' ')

{

qwerty = qwerty + m2[k][j];

qwerty.pop\_back();

}

qwerty = qwerty + m1[i][k][u];

}

}

}

if (qwerty != "")

result[i][j] = removeRepeating(qwerty);

//result[i][j] = qwerty;

}

}

}

return result;

}

std::vector <std::vector<std::string>> operator+(const std::vector <std::vector<std::string>>& m1, const std::vector <std::vector<std::string>>& m2)

{

std::vector <std::vector<std::string>> result;

result.resize(m1.size());

for (int i = 0; i < m1.size(); i++)

result[i].resize(m1.size(), "");

for (int i = 0; i < result.size(); ++i)

{

for (int j = 0; j < result.size(); ++j)

{

result[i][j] += m1[i][j] + m2[i][j];

}

}

return result;

}

AutoNet::AutoNet(std::string fileName)

{

//Open file

std::ifstream in(fileName);

if (!in.is\_open())

{

std::cout << "Can't open file." << std::endl;

exit(1);

}

//Read file

std::string temp;

std::vector <std::string> textFile;

while (std::getline(in, temp))

{

textFile.push\_back(temp);

std::cout << temp << std::endl;

}

std::cout << std::endl;

in.close();

std::vector<std::vector <std::string>> neighbors;

//Find all nodes and neighbors

for (int i = 0; i < textFile.size(); i++)

{

for (int j = 0; j < textFile[i].size(); j++)

{

if ('A' <= textFile[i][j] && textFile[i][j] <= 'Z')

{

int t = 0;

while ('A' <= textFile[i][j + t] && textFile[i][j + t] <= 'Z')

t++;

temp = "";

for (int k = 0; k < t; k++)

temp += textFile[i][j + k];

\_nodes.push\_back(temp);

//Find neighbors

neighbors.resize(\_nodes.size());

//up

if (i != 0)

{

if ('a' <= textFile[i - 1][j] && textFile[i - 1][j] <= 'z')

{

int t2 = 0;

while ('1' <= textFile[i - 1][j + t2 + 1] && textFile[i - 1][j + t2 + 1] <= '9')

t2++;

temp = textFile[i - 1][j];

for (int u = 0; u < t2; u++)

{

temp += textFile[i - 1][j + u + 1];

}

neighbors[\_nodes.size() - 1].push\_back(temp);

//std::cout << temp << std::endl;

}

}

//down

if (i != (textFile.size() - 1))

{

if ('a' <= textFile[i + 1][j] && textFile[i + 1][j] <= 'z')

{

int t2 = 0;

while ('1' <= textFile[i + 1][j + t2 + 1] && textFile[i + 1][j + t2 + 1] <= '9')

t2++;

temp = textFile[i + 1][j];

for (int u = 0; u < t2; u++)

{

temp += textFile[i + 1][j + u + 1];

}

neighbors[\_nodes.size() - 1].push\_back(temp);

//std::cout << temp << std::endl;

}

}

//left

if (j != 0)

{

if (textFile[i][j - 1] != ' ')

{

int t2 = 0;

while ('a' > textFile[i][j - 1 - t2] || textFile[i][j - 1 - t2] > 'z')

t2++;

if (t2 == 0)

temp = textFile[i][j - 1];

else

{

temp = "";

for (int u = 0; u < t2 + 1; u++)

{

temp += textFile[i][j - 1 - t2 + u];

}

}

neighbors[\_nodes.size() - 1].push\_back(temp);

//std::cout << temp<<std::endl;

}

}

//right

if (\_nodes[\_nodes.size() - 1].size() > 1)

{

j += t - 1;

t = 1;

}

if (j != textFile[i].size() - 1)

{

if (textFile[i][j + 1] != ' ')

{

int t2 = 0;

char q = textFile[i][j + 2 + t2];

while ('a' > textFile[i][j + 2 + t2] || textFile[i][j + 2 + t2] > 'z')

{

t2++;

q = textFile[i][j + 2 + t2];

}

if (t2 == 0)

temp = textFile[i][j + 1];

else

{

temp = "";

for (int u = 0; u < t2 + 1; u++)

{

temp += textFile[i][j + 1 + u];

}

}

neighbors[\_nodes.size() - 1].push\_back(temp);

//std::cout << temp << std::endl;

}

}

j += t;

}

else

continue;

}

}

//Complete Adjacency Matrix

\_adjacencyMatrix.resize(\_nodes.size());

for (int i = 0; i < \_nodes.size(); i++)

\_adjacencyMatrix[i].resize(\_nodes.size(), "");

calcAdjacencyMatrix(neighbors);

}

void AutoNet::calcAdjacencyMatrix(std::vector<std::vector <std::string>> neighbors)

{

for (int i = 0; i < \_nodes.size() - 1; i++)

{

for (int m = 0; m < neighbors[i].size(); m++)

{

for (int j = i + 1; j < \_nodes.size(); j++)

{

for (int n = 0; n < neighbors[j].size(); n++)

{

if (neighbors[i][m] == neighbors[j][n] && \_nodes[i] != \_nodes[j])

{

\_adjacencyMatrix[i][j] = neighbors[i][m] + " ";

\_adjacencyMatrix[j][i] = neighbors[i][m] + " ";

}

}

}

}

}

}

void AutoNet::printNodes()

{

for (int i = 0; i < \_nodes.size(); i++)

std::cout << \_nodes[i] << " ";

}

void AutoNet::findPath(std::string a, std::string b)

{

//find Departure and destination

int Departure = -1, Destination = -1;

for (int i = 0; i < \_nodes.size(); i++)

{

if (\_nodes[i] == a)

Departure = i;

if (\_nodes[i] == b)

Destination = i;

}

if (Departure == -1 || Destination == -1)

{

std::cout << "Incorrect input. Terminate.\n";

exit(2);

}

calcPathwayMatrix(-1);

if (\_pathwayMatrix[Departure][Destination] == "")

{

std::cout << "No ways from \"" << a << "\" to \"" << b << "\"." << std::endl;

}

else

{

std::cout << "Ways from \"" << a << "\" to \"" << b << "\":" << std::endl;

std::cout << \_pathwayMatrix[Departure][Destination];

}

}

void AutoNet::findPath(std::string a, std::string b, int limit)

{

//find Departure and destination

int Departure = -1, Destination = -1;

for (int i = 0; i < \_nodes.size(); i++)

{

if (\_nodes[i] == a)

Departure = i;

if (\_nodes[i] == b)

Destination = i;

}

if (Departure == -1 || Destination == -1)

{

std::cout << "Incorrect input. Terminate.\n";

exit(2);

}

calcPathwayMatrix(limit);

if (\_pathwayMatrix[Departure][Destination] == "")

{

std::cout << "No ways from \"" << a << "\" to \"" << b << "\":" << std::endl;

}

else

{

std::cout << "Ways from \"" << a << "\" to \"" << b << "\", with limit " << limit << ":" << std::endl;

std::string temp = \_pathwayMatrix[Departure][Destination];

bool flag = false;

int count;

for (int i = 0; i < temp.size(); i++)

{

if (temp[i] != ' ' && !flag)

{

flag = true;

count = 1;

}

else if (temp[i] == ' ')

{

flag = false;

if (count > limit)

{

temp.erase(i - count, temp.size() - i + count);

}

}

else

count++;

}

std::cout << temp;

}

}

void AutoNet::calcPathwayMatrix(int limits)

{

if (limits == -1)

limits = \_nodes.size() / 2;

\_pathwayMatrix.resize(\_nodes.size());

for (int i = 0; i < \_pathwayMatrix.size(); i++)

\_pathwayMatrix[i].resize(\_nodes.size(), "");

std::vector <std::vector<std::string>> temp;

temp.resize(\_nodes.size());

for (int i = 0; i < temp.size(); i++)

temp[i].resize(\_nodes.size(), "");

//Init

temp = \_adjacencyMatrix;

//Start cycle

for (int i = 0; i < limits; i++)

{

\_pathwayMatrix = \_pathwayMatrix + temp;

temp = temp \* \_adjacencyMatrix;

}

}

int main(int argc, char\* argv[])

{

AutoNet name(argv[1]);

std::cout << "Localities: ";

name.printNodes();

std::string Destination, Departure, input;

int limit;

bool next=0;

do {

std::cout << std::endl;

std::cout << "Choose Departure: ";

std::cin >> Departure;

std::cout << "Choose Destination: ";

std::cin >> Destination;

std::cout << "Do you want to put a limit on the length of the route? [Y/N]: ";

std::cin >> input;

if (input == "Y")

{

std::cout << "Enter limit: ";

std::cin >> limit;

name.findPath(Departure, Destination, limit);

}

else if (input == "N")

{

name.findPath(Departure, Destination);

}

else

{

std::cout << "Incorrect input. Terminate.\n";

exit(2);

}

std::cout << std::endl << std::endl;

std::cout << "Do you want to continue using the program? [Y/N]: ";

std::cin >> input;

if (input == "Y")

{

next = true;

}

else if (input == "N")

{

next = false;

}

else

{

std::cout << "Incorrect input. Terminate.\n";

exit(2);

}

} while (next);

return 0;

}

**Тесты:**

|  |  |
| --- | --- |
| **Номер** | **Выход** |
| 1 |  |

**Список использованной литературы:**

* Волосатова Т.М., Родионов С.В. Лекции по курсу «Объектно-ориентированное программирование»
* bigor.bmstu.ru