**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра САПР**

**ОТЧЁТ**

**по лабораторной работе №3**

**по дисциплине «Алгоритмы и структуры данных»**

**Тема: «Алгоритмы на графах»**

**Вариант 1.**

|  |  |  |
| --- | --- | --- |
| Студент гр. 9302 |  | Горбанев А.В |
| Преподаватель |  | Тутуева А.В. |

Санкт-Петербург

2021

**Постановка задачи и описание реализуемого класса и методов.**

Задание: Реализовать алгоритм Дейкстры и списки смежности для поиска наиболе эффективного по стоимости перелета из одного города в тругой.

Для этого мне понадобился класс “List”.

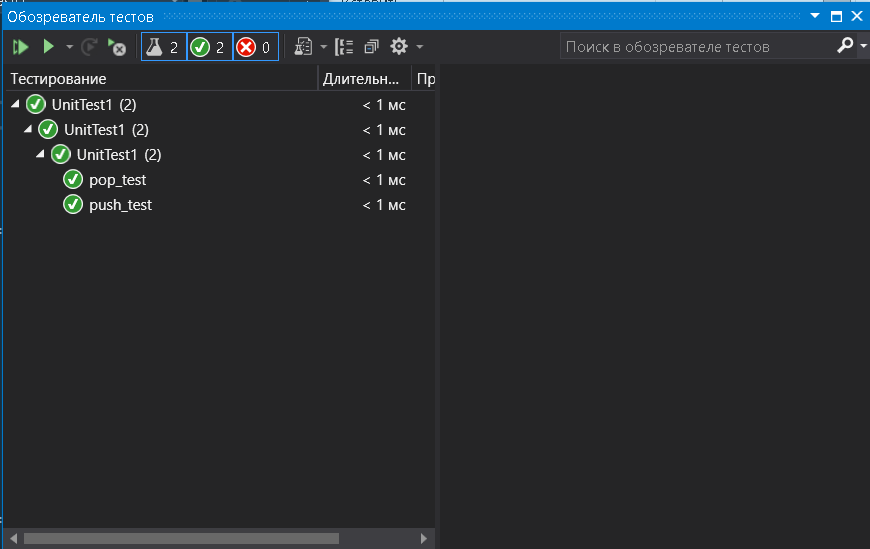
Оценка временной сложности каждого метода:

|  |  |
| --- | --- |
| Функция | Сложность |
| Insert(key,data) | O(log(n)) |
| Remove(key) | O(log(n)) |
| Find(key) | O(log(n)) |
| Clear | O(n) |
| Get\_keys | O(n) |
| Get\_values | O(n) |
| print | O(n) |

**Описание реализованных unit-тестов.**

|  |  |
| --- | --- |
| Проверка методов класса «List» | |
| Название теста | Что проверяет |
| Push\_test | Проверка метода push |
| Pop\_test | Проверка метода pop |

**Результат выполнения всех unit-тестов**



**Листинг**

|  |
| --- |
| List.h |
| #ifndef LIST\_H  #define LIST\_H  #include <string>  template<class T>  class list {  private:  class nodeList;  nodeList\* \_head;  int \_size;  public:  list();  ~list();  T pop();  T pop(int);  void push(T value);  int getSize();  void sort();  std::string getString();  const T operator[](const int index) const;  //friend bool operator==(const T &val1, const T& val2);  };  template <class T>  class list<T>::nodeList {  private:  T \_value;  nodeList\* \_next;  public:  nodeList() : \_next(nullptr) {};  nodeList(T value, nodeList\* next) :  \_value(value), \_next(next) {}  T getValue() {  return \_value;  }  void setValue(T value) {  \_value = value;  }  nodeList\* getNext() {  return \_next;  }  void setNext(nodeList\* next) {  \_next = next;  }  };  // methods list  template <class T>  list<T>::list() : \_head(nullptr), \_size(0) {}  template<class T>  list<T>::~list()  {  while (\_size > 0)  this->pop();  }  template <class T>  T list<T>::pop() {  if (\_size == 0)  throw;  if (\_size == 1) {  T tempNextVal = \_head->getValue();  delete \_head;  \_head = nullptr;  this->\_size = 0;  return tempNextVal;  }  nodeList\* temp = \_head;  while (temp->getNext()->getNext() != nullptr) temp = temp->getNext();  T tempNextVal = temp->getNext()->getValue();  delete temp->getNext();  \_size--;  temp->setNext(nullptr);  return tempNextVal;  }  template <class T>  T list<T>::pop(int index) {  if (\_size == 1 || \_size == 0 || index == \_size - 1) {  return this->pop();  }  if (index == 0) {  nodeList\* temp = \_head->getNext();  T tempvalue = \_head->getValue();  \_head = temp;  \_size--;  return tempvalue;  }  nodeList\* temp = \_head;  for (int i = 0; i < index - 1; i++)  temp = temp->getNext();  T tempNextVal = temp->getNext()->getValue();  nodeList\* tempNext = temp->getNext()->getNext();  \_size--;  temp->setNext(tempNext);  return tempNextVal;  }  template <class T>  void list<T>::push(T value) {  if (\_size == 0) {  \_head = new nodeList(value, nullptr);  \_size++;  return;  }  nodeList\* temp = \_head;  while (temp->getNext() != nullptr) temp = temp->getNext();  temp->setNext(new nodeList(value, nullptr));  \_size++;  return;  }  template <class T>  int list<T>::getSize() {  return \_size;  }  template <class T>  void list<T>::sort() {  if (\_size == 1 || \_size == 0)  return;  nodeList\* temp = \_head;  bool f = false;  for (int i = 0; i < \_size - 1; i++) {  temp = \_head;  f = false;  for (int j = 0; j < \_size - 1 - i; j++) {  if ((temp->getValue()).cost > (temp->getNext()->getValue()).cost) {  T tempVal = temp->getValue();  temp->setValue(temp->getNext()->getValue());  temp->getNext()->setValue(tempVal);  f = true;  }  temp = temp->getNext();  }  if (!f)  return;  }  }  template <class T>  std::string list<T>::getString() {  nodeList\* temp = \_head;  std::string \_out = "";  while (temp != NULL) {  \_out += temp->getValue() + '0';  temp = temp->getNext();  }  return \_out;  }  template <class T>  const T list<T>::operator[](const int index) const {  nodeList\* it = \_head;  if (index < 0 || index > \_size)  throw;  for (int i = 0; i < index; i++) {  it = it->getNext();  }  return it->getValue();  }  #endif |
| Source.cpp |
| #include <iostream>  #include <fstream>  #include <string>  #include <windows.h>  #include "list.h"  using namespace std;  class City {  public:  string name;  int cost;  City(string nameP, int costP) : name(nameP), cost(costP) {};  };  class ProcessCity {  public:  string name;  int disatnce;  bool isVisited;  ProcessCity() {};  ProcessCity(string nameP) : name(nameP) {};  };  bool isDouble(list<string>& arr, string str) {  for (int i = 0; i < arr.getSize(); i++) {  string a = arr[i];  if (arr[i] == str)  return true;  }  return false;  }  bool allVisited(ProcessCity\* pCity, int count) {  for (int i = 0; i < count; i++) {  if (pCity[i].isVisited == false)  return false;  }  return true;  }  int minDistance(ProcessCity\* pCity, int count) {  int index = -1;  int min = INT\_MAX;  for (int i = 0; i < count; i++) {  if (pCity[i].disatnce < min && pCity[i].isVisited == false) {  min = pCity[i].disatnce;  index = i;  }  }  return index;  }  int getIndex(string\* arr, string str) {  for (int i = 0; i < arr->size(); i++) {  if (arr[i] == str)  return i;  }  return -1;  }  int isConnected(string\* indices, list<City>\*\* adjancencyList, int begin, int end) {  for (int i = 0; i < adjancencyList[begin]->getSize(); i++)  if (getIndex(indices, (\*adjancencyList[begin])[i].name) == end)  return (\*adjancencyList[begin])[i].cost;  return -1;  }  int main() {  SetConsoleCP(1251);  SetConsoleOutputCP(1251);  setlocale(LC\_ALL, "rus");  fstream f;  f.open("in.txt");  int countString = 0;  string bufString;  while (!f.eof()) {  getline(f, bufString);  countString++;  }  f.close();  f.open("in.txt");  string\*\* inputList = new string \* [countString];  for (int i = 0; i < countString; i++) {  inputList[i] = new string[4];  for (int j = 0; j < 3; j++) {  getline(f, bufString, ';');  if (bufString != "N/A")  inputList[i][j] = bufString;  else  inputList[i][j] = "-1";  }  getline(f, bufString);  if (bufString != "N/A")  inputList[i][3] = bufString;  else  inputList[i][3] = "-1";  }  f.close();  list<string> cities;  for (int i = 0; i < countString; i++) {  if (!isDouble(cities, inputList[i][0]))  cities.push(inputList[i][0]);  if (!isDouble(cities, inputList[i][1]))  cities.push(inputList[i][1]);  }  list<City>\*\* adjancencyList = new list<City> \* [cities.getSize()];  for (int i = 0; i < cities.getSize(); i++) {  City\* buf = new City(cities[i], 0);  list<City>\* buf2 = new list<City>;  buf2->push(\*buf);  adjancencyList[i] = buf2;  }  for (int i = 0; i < countString; i++) {  int j = 0;  if (inputList[i][2] != "-1") {  while (inputList[i][0] != (\*adjancencyList[j])[0].name)  j++;  City\* buf = new City(inputList[i][1], stoi(inputList[i][2]));  (\*adjancencyList[j]).push(\*buf);  }  j = 0;  if (inputList[i][3] != "-1") {  while (inputList[i][1] != (\*adjancencyList[j])[0].name)  j++;  City\* buf = new City(inputList[i][0], stoi(inputList[i][3]));  (\*adjancencyList[j]).push(\*buf);  }  }    for (int i = 0; i < cities.getSize(); i++) {  (\*adjancencyList[i]).sort();  }    string source, destination;  cout << "Source:";  cin >> source;  cout << "Destination:";  cin >> destination;  int infinity = INT\_MAX;  ProcessCity\* processCity = new ProcessCity[cities.getSize()];  string\* indices = new string[cities.getSize()];    for (int i = 0; i < cities.getSize(); i++) {  indices[i] = (\*adjancencyList[i])[0].name;  processCity[i].name = (\*adjancencyList[i])[0].name;  processCity[i].isVisited = false;  processCity[i].disatnce = infinity;  if ((\*adjancencyList[i])[0].name == source)  processCity[i].disatnce = 0;  }    while (!allVisited(processCity, cities.getSize())) {  int curNodeIndex = minDistance(processCity, cities.getSize());    for (int i = 1; i < (\*adjancencyList[curNodeIndex]).getSize(); i++) {    if (processCity[getIndex(indices, (\*adjancencyList[curNodeIndex])[i].name)].isVisited == false &&  processCity[curNodeIndex].disatnce + (\*adjancencyList[curNodeIndex])[i].cost <  processCity[getIndex(indices, (\*adjancencyList[curNodeIndex])[i].name)].disatnce) {  processCity[getIndex(indices, (\*adjancencyList[curNodeIndex])[i].name)].disatnce =  processCity[curNodeIndex].disatnce + (\*adjancencyList[curNodeIndex])[i].cost;  }  else  continue;  }  processCity[curNodeIndex].isVisited = true;  }    int\* ver = new int[cities.getSize()];  int end = getIndex(indices, destination);  int start = getIndex(indices, source);  ver[0] = end;  int k = 1;  int weight = processCity[end].disatnce;  while (end != start) {  for (int i = 0; i < cities.getSize(); i++) {  if (isConnected(indices, adjancencyList, i, end) != -1) {  int temp = weight - isConnected(indices, adjancencyList, i, end);  if (temp == processCity[i].disatnce) {  weight = temp;  end = i;  ver[k] = i;  k++;  }  }  }  }  cout << endl;  cout << endl;  cout << endl;  for (int i = k - 1; i > 0; i--)  cout << indices[ver[i]] << " -> ";  cout << indices[ver[0]];  cout << endl;  cout << endl;  cout << endl;  end = getIndex(indices, destination);  cout << "Эффективная стоимость:" << processCity[end].disatnce << '\n';  } |
| UnitTest1.cpp |
| #include "pch.h"  #include "CppUnitTest.h"  #include "../3laba/list.h"  using namespace Microsoft::VisualStudio::CppUnitTestFramework;  namespace UnitTest1  {  TEST\_CLASS(UnitTest1)  {  public:  TEST\_METHOD(push\_test)  {  list<int> list;  list.push(1);  list.push(2);  Assert::IsTrue(list[1] == 2);  }  TEST\_METHOD(pop\_test)  {  list<int> list;  list.push(1);  list.push(2);  list.push(3);  list.pop();  Assert::IsTrue(list[1] == 2);  }  };  } |

Вывод:

В ходе лабораторной работы научился работать с алгоритмом Дейкстры на графах.