**Министерство образования и науки Украины**

**Национальный технический университет Украины "Киевский политехнический институт имени Игоря Сикорского"**

**Факультет информатики и вычислительной техники**

**Кафедра автоматизированных систем обработки**

**информации и управления**

**ОТЧЕТ**

по лабораторной работе № 4 по дисциплине

«Проектирование и анализ вычислительных алгоритмов»

„ **Эвристические алгоритмы** ”

**Выполнил**

(шифр, фамилия, имя, отчество)

*ІП-73, Іващук Владислав*

**Проверил**

(фамилия, имя, отчество )

*Головченко М.Н.*

Киев 2018

СОДЕРЖАНИЕ

1 ЦЕЛЬ ЛАБОРАТОРНОЙ РАБОТЫ 3

2 ЗАДАНИЕ 4

3 ВЫПОЛНЕНИЕ 7

3.1 Псевдокод алгоритма 7

3.2 Анализ временной сложности 7

3.3 Программная реализация алгоритма 7

*3.3.1* *Исходный код 7*

*3.3.2* *Примеры работы 7*

3.4 Испытания алгоритма 8

*3.4.1* *Временные оценочные характеристики 8*

*3.4.2* *Графики зависимости временных оценочных характеристик и времени поиска от размерности структур 9*

ВЫВОДЫ 10

КРИТЕРИИ ОЦЕНИВАНИЯ **11**

1. ЦЕЛЬ ЛАБОРАТОРНОЙ РАБОТЫ

Цель работы – изучить основные подходы к формализации эвристических алгоритмов и решению типовых задач с их помощью.

1. ЗАДАНИЕ

Выбрать 15 городов в стране согласно варианту (таблица 2.1) и записать для них кратчайшее расстояние по дороге, в случае прямого сообщения между ними и расстояние по прямой в отдельные таблицы. Для определения расстояний рекомендуется использовать интернет сервисы (например Google Maps).

Записать алгоритм методов Жадный поиск и поиск А\* для задачи нахождения кратчайшего пути между парами вершин в транспортной сети (неориентированном графе)

Разработать программу, которая будет находить кратчайшие маршруты между каждой парой городов. В качестве методов нахождения маршрутов выбрать Жадный поиск и поиск А\*. В качестве эвристики выбрать расстояние по прямой.

Ответ выводить в виде (Город1-Город2 Расстояние: 234км Маршрут: Город1 → Город3 → Город4 → Город2).

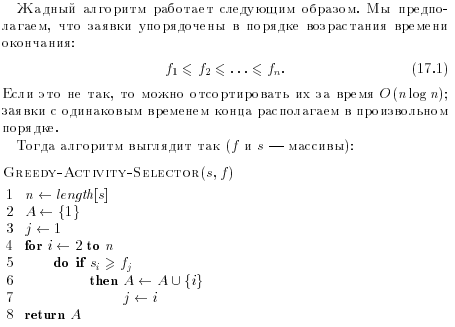
Сделать обобщенный вывод по лабораторной работе, в котором оценить качество алгоритмов.

Таблица 2.1 – Варианты

|  |  |
| --- | --- |
| **№** | **Вариант** |
| 1 | Индонезия |
| 2 | Австралия |
| 3 | Австрия |
| 4 | Азербайджан |
| 5 | Испания |
| 6 | Албания |
| 7 | Алжир |
| 8 | Италия |
| 9 | Ангола |
| 10 | ОАЭ |
| 11 | **Аргентина** |
| 12 | Армения |
| 13 | Мексика |
| 14 | Афганистан |
| 15 | Молдавия |
| 16 | Бангладеш |
| 17 | Барбадос |
| 18 | Польша |
| 19 | Беларусь |
| 20 | Португалия |
| 21 | Бельгия |
| 22 | Сербия |
| 23 | Болгария |
| 24 | Словакия |
| 25 | Норвегия |
| 26 | Нидерланды |
| 27 | Перу |
| 28 | Сингапур |
| 29 | Таиланд |
| 30 | Турция |

1. ВЫПОЛНЕНИЕ
   1. Псевдокод алгоритм

Жадный алгоритм



А\*

function A\*(start, goal, f)

var closed = the empty set

var open = make\_queue(f)

enqueue(open, path(start))

while open is not empty

var p = remove\_first(open)

var x == the last node of p

if x in closed

continue

if x == goal

return p

add(closed, x)

foreach y in successors(x)

enqueue(open, add\_to\_path(p, y))

return failure

* 1. Входные данные задачи

В таблице 3.1 приведены расстояния между городами по дороге, если между ними есть прямое сообщение.

Таблица 3.1 – Расстояния между городами по дороге

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Cordoba | Rosario | Santa Fe | Mendoza | Buenos Aires | La Plata | Mar del Plata | Necochea | Bahia Blanca | Santa Rosa | San Carlos de Bariloche | Trelew | San Miguel de Tacuman | Salta | Villa Maria |
| Cordoba | 0 | 321 | 199 | 325 | 28 | 75 | 412 | 25 | 52 | 15 | 115 | 113 | 52 | 564 | 59 |
| Rosario | 256 | 0 | 97 | 19 | 82 | 65 | 139 | 82 | 918 | 112 | 720 | 151 | 154 | 90 | 50 |
| Santa Fe | 39 | 12 | 0 | 87 | 178 | 29 | 110 | 46 | 55 | 110 | 80 | 70 | 146 | 156 | 452 |
| Mendoza | 132 | 759 | 287 | 0 | 113 | 60 | 83 | 47 | 636 | 57 | 241 | 116 | 82 | 70 | 36 |
| Buenos Aires | 78 | 92 | 178 | 133 | 0 | 149 | 194 | 156 | 127 | 133 | 99 | 228 | 534 | 832 | 125 |
| La Plata | 524 | 65 | 29 | 80 | 149 | 0 | 103 | 31 | 26 | 95 | 51 | 83 | 118 | 129 | 31 |
| Mar del Plata | 135 | 179 | 110 | 93 | 194 | 103 | 0 | 71 | 98 | 33 | 120 | 80 | 161 | 122 | 76 |
| Necochea | 69 | 92 | 46 | 47 | 156 | 31 | 71 | 0 | 333 | 65 | 60 | 71 | 123 | 116 | 12 |
| Bahia Blanca | 522 | 148 | 55 | 836 | 127 | 291 | 98 | 333 | 0 | 80 | 230 | 152 | 94 | 102 | 23 |
| Santa Rosa | 86 | 112 | 110 | 67 | 163 | 116 | 733 | 65 | 80 | 0 | 97 | 251 | 129 | 89 | 63 |
| San Carlos de Bariloche | 225 | 50 | 280 | 141 | 99 | 651 | 120 | 260 | 30 | 497 | 0 | 253 | 67 | 90 | 150 |
| Trelew | 443 | 251 | 70 | 116 | 228 | 586 | 80 | 271 | 102 | 151 | 138 | 0 | 194 | 199 | 83 |
| San Miguel de Tacuman | 252 | 54 | 146 | 92 | 934 | 218 | 261 | 123 | 94 | 129 | 67 | 198 | 0 | 68 | 113 |
| Salta | 64 | 90 | 156 | 70 | 88 | 129 | 122 | 116 | 103 | 89 | 90 | 179 | 364 | 0 | 107 |
| Villa Maria | 159 | 59 | 59 | 936 | 165 | 31 | 79 | 112 | 18 | 635 | 250 | 83 | 113 | 108 | 0 |

В таблице 3.2 приведены расстояния между городами по прямой.

Таблица 3.2 – Расстояния между городами по прямой

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Cordoba | Rosario | Santa Fe | Mendoza | Buenos Aires | La Plata | Mar del Plata | Necochea | Bahia Blanca | Santa Rosa | San Carlos de Bariloche | Trelew | San Miguel de Tacuman | Salta | Villa Maria |
| Cordoba | 0 | 32 | 99 | 32 | 88 | 75 | 115 | 69 | 52 | 86 | 525 | 143 | 52 | 564 | 59 |
| Rosario | 132 | 0 | 97 | 59 | 82 | 65 | 139 | 82 | 948 | 112 | 720 | 151 | 154 | 90 | 50 |
| Santa Fe | 99 | 97 | 0 | 87 | 178 | 29 | 110 | 46 | 55 | 110 | 80 | 70 | 146 | 156 | 452 |
| Mendoza | 232 | 759 | 287 | 0 | 113 | 60 | 83 | 47 | 636 | 57 | 241 | 116 | 82 | 70 | 36 |
| Buenos Aires | 88 | 82 | 178 | 113 | 0 | 149 | 194 | 156 | 127 | 163 | 99 | 228 | 534 | 88 | 145 |
| La Plata | 475 | 65 | 29 | 60 | 149 | 0 | 103 | 31 | 26 | 95 | 51 | 86 | 118 | 129 | 31 |
| Mar del Plata | 115 | 139 | 110 | 83 | 194 | 103 | 0 | 71 | 98 | 33 | 120 | 80 | 161 | 122 | 76 |
| Necochea | 69 | 82 | 46 | 47 | 156 | 31 | 71 | 0 | 333 | 65 | 60 | 71 | 123 | 116 | 12 |
| Bahia Blanca | 52 | 248 | 55 | 836 | 127 | 26 | 98 | 333 | 0 | 80 | 230 | 102 | 94 | 103 | 23 |
| Santa Rosa | 86 | 112 | 110 | 57 | 163 | 95 | 733 | 65 | 80 | 0 | 97 | 101 | 129 | 89 | 63 |
| San Carlos de Bariloche | 125 | 20 | 280 | 141 | 99 | 651 | 120 | 260 | 30 | 497 | 0 | 132 | 67 | 90 | 150 |
| Trelew | 143 | 151 | 70 | 116 | 228 | 486 | 80 | 271 | 102 | 101 | 132 | 0 | 194 | 179 | 83 |
| San Miguel de Tacuman | 252 | 54 | 146 | 82 | 934 | 118 | 161 | 123 | 94 | 129 | 67 | 194 | 0 | 64 | 113 |
| Salta | 64 | 90 | 156 | 70 | 88 | 129 | 122 | 116 | 103 | 89 | 90 | 179 | 364 | 0 | 107 |
| Villa Maria | 159 | 50 | 52 | 936 | 145 | 31 | 76 | 112 | 23 | 635 | 250 | 83 | 113 | 107 | 0 |

* 1. Программная реализация
     1. Исходный код

#include <iostream>

#include <map>

#include <vector>

#include <random>

**using namespace** std;

**class** Town{

**public**:

string name\_;

Town(string name){

name\_ = name;

}

**bool operator**< (**const** Town& userObj) **const**

{

**if**(userObj.name\_.size() < **this**->name\_.size())

**return true**;

}

};

**class** Node{

**public**:

Town \* t;

Node(Town \*t\_){

t = t\_;

}

map<Town\*, **int**> neighbors\_;

**void** add(Town \*t, **int** length){

**this**->neighbors\_.insert(pair<Town\*,**int**>(t,length));

}

**int** get\_l(Town \*t){

**if**(**this**->neighbors\_.find(t) != **this**->neighbors\_.end()) {

**return this**->neighbors\_[t];

}

}

};

**class** Greedy{

**private**:

**int** path\_ = 0;

vector<Town\*> Greedy\_algo(Town \* t1, Town \* t2, vector<Node> graph){

vector<Town\*> path;

Town \* curr\_t, \* min\_t;

curr\_t = t1;

path.push\_back(t1);

**while** (curr\_t != t2){

Node \* curr\_n = &graph[index\_find(graph,curr\_t)];

**auto** it = curr\_n->neighbors\_.begin();

min\_t = it->first;

**for**(; it != curr\_n->neighbors\_.end(); it++ ) {

**if**(graph[index\_find(graph,it->first)].get\_l(t2) < graph[index\_find(graph,min\_t)].get\_l(t2)){

min\_t = it->first;

}

}

path\_ += graph[index\_find(graph, curr\_t)].get\_l(min\_t);

curr\_t = min\_t;

path.push\_back(min\_t);

}

**return** path;

}

**public**:

Greedy(){}

**void** greedy(Town \*t1, Town \*t2, vector<Node> graph){

vector<Town \*> path\_greedy = Greedy\_algo(t1, t2, graph);

cout << "\n[Greedy] " << t1->name\_ << "-" << t2->name\_ << ". Distance: " << calculate\_path() << ". Path: ";

**for** (**auto** it = path\_greedy.begin(); it != path\_greedy.end(); ) {

cout << (\*it)->name\_;

**if** (++it != path\_greedy.end()) cout << "->";

}

}

**int** calculate\_path(){

**return** path\_;

}

};

**class** AStar{

**private**:

**int** path\_ = 0;

vector<Town\*> AStar\_algo(Town \* t1, Town \* t2, vector<Node> graph) {

vector<Town \*> path;

Town \*curr\_t, \*min\_t;

curr\_t = t1;

path.push\_back(t1);

**while** (curr\_t != t2) {

Node \*curr\_n = &graph[index\_find(graph, curr\_t)];

**auto** it = curr\_n->neighbors\_.begin();

it++;

min\_t = it->first;

**for** (; it != curr\_n->neighbors\_.end(); it++) {

**int** l = (graph[index\_find(graph, it->first)].get\_l(t2) + curr\_n->get\_l(it->first));

**int** r = (graph[index\_find(graph, min\_t)].get\_l(t2) + curr\_n->get\_l(min\_t));

**if** (l <= r) {

min\_t = it->first;

}

}

path\_ += graph[index\_find(graph, curr\_t)].get\_l(min\_t);

curr\_t = min\_t;

path.push\_back(min\_t);

}

**return** path;

}

**public**:

AStar(){}

**void** astar(Town \*t1, Town \*t2, vector<Node> graph){

vector<Town \*> path\_astar = AStar\_algo(t1, t2, graph);

cout << "\n[A\*] " << t1->name\_ << "-" << t2->name\_ << ". Distance: " << calculate\_path() << ". Path: ";

**for** (**auto** it = path\_astar.begin(); it != path\_astar.end(); ) {

cout << (\*it)->name\_;

**if** (++it != path\_astar.end()) cout << "->";

}

}

**int** calculate\_path(){

**return** path\_;

}

};

**int** index\_find(vector<Node>, Town \*);

**int** main() {

Town towns[] = {Town("Cordoba"),

Town("Rosario"),

Town("Santa Fe"),

Town("Mendoza"),

Town("Buenos Aires"),

Town("La Plata"),

Town("Mar del Plata"),

Town("Necochea"),

Town("Bahia Blanca"),

Town("Santa Rosa"),

Town("San Carlos de Bariloche"),

Town("Trelew"),

Town("San Miguel de Tacuman"),

Town("Salta"),

Town("Villa Maria")};

vector<Node> graph;

**for** (**int** i = 0; i < 15; i++) {

graph.emplace\_back(Node(&towns[i]));

graph[i].add(&towns[i], 0);

**for** (**int** j = 1; j < 15; j++) {

graph[i].add(&towns[j], rand() % (j \* 10) + (j - 1) \* 10 + 1);

}

}

AStar a;

Greedy g;

a.astar(&towns[0],&towns[7], graph);

g.greedy(&towns[0],&towns[7], graph);

**return** 0;

}

**int** index\_find(vector<Node> graph, Town \* curr){

**for** (**int** i = 0; i < graph.size(); i++){

**if** (graph[i].t == curr){

**return** i;

}

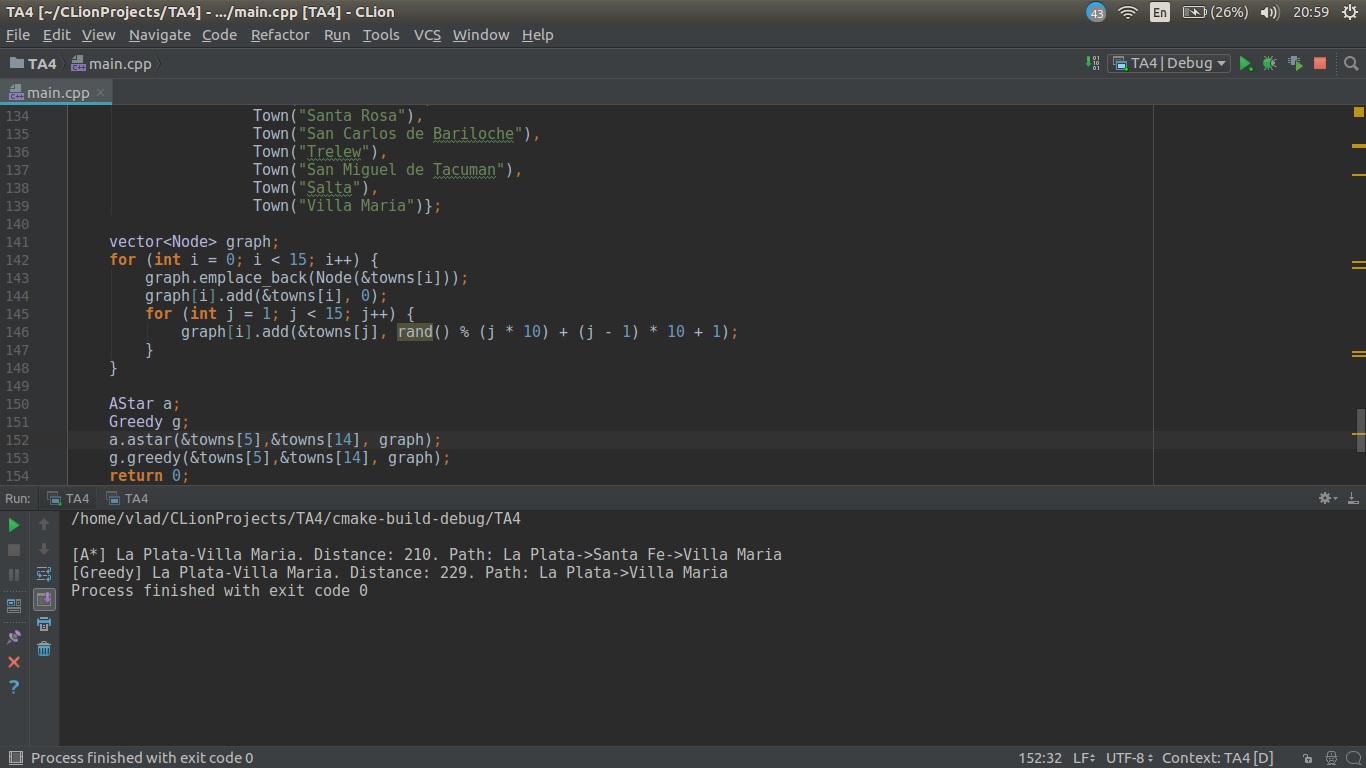
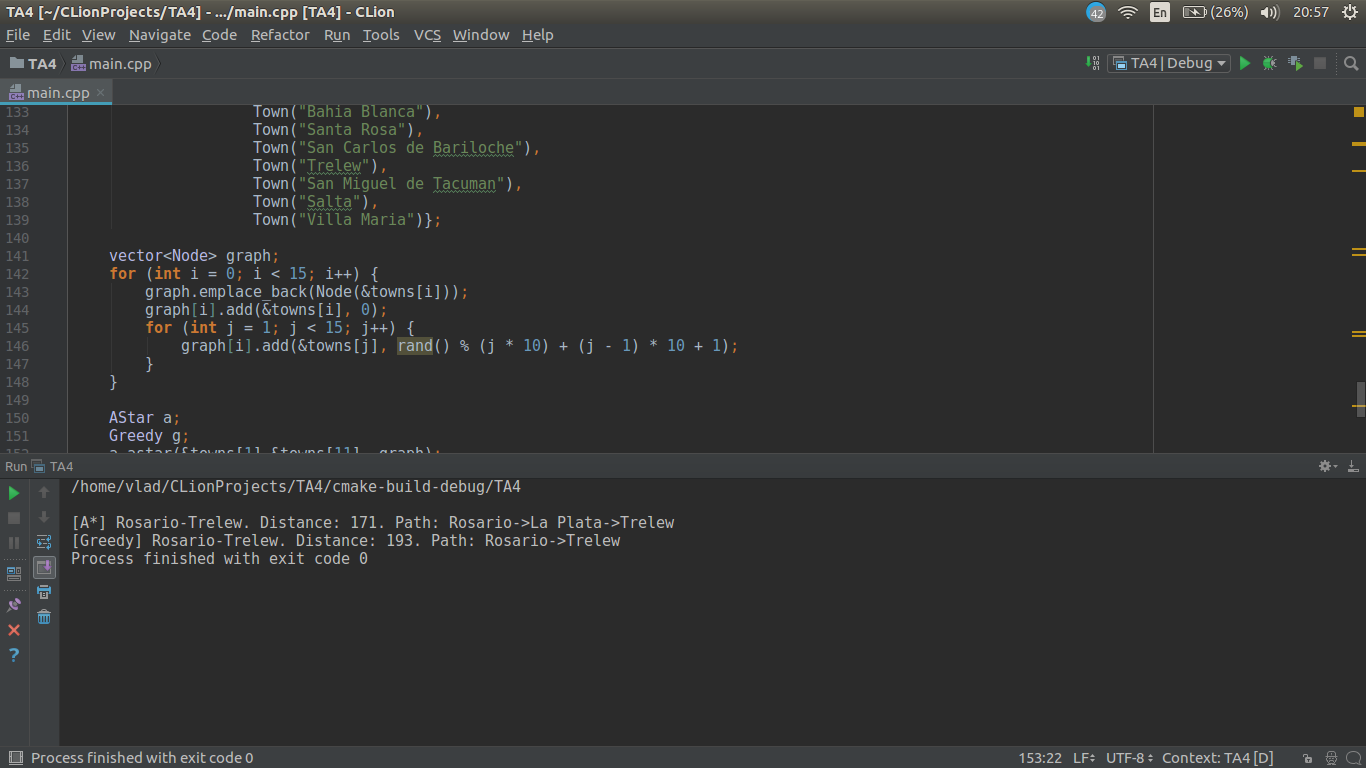
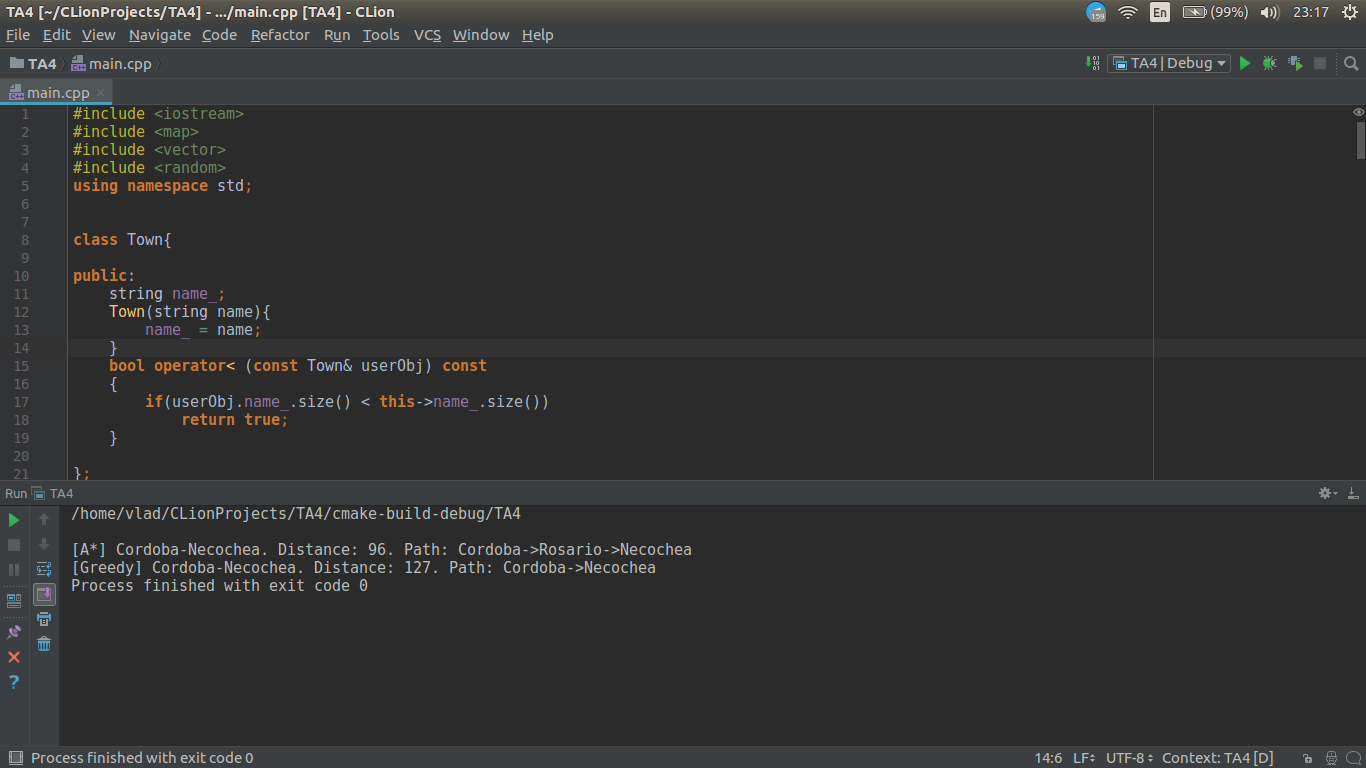
}

}

* + 1. Примеры работы

На рисунках 3.1 и 3.2 показаны примеры работы программы для разных алгоритмов поиска.

Рисунок 3.1 – Жадный поиск Рисунок 3.2 – Поиск А\*



1. ВЫВОДЫ

В рамках данной лабораторной работы я изучил основные подходы к формализации эвристических алгоритмов и приобрел навыки решения типовых задач с их помощью. Были реализованы два алгоритма: жадный алгоритм и А\*. Также я протестировал их на нахождение кратчайшего пути между городами Аргентины, данные о которых хранились в заранее заготовленных таблицах.

После проведенных опытов, с уверенностью можно сказать, что жадный поиск по первому наилучшему совпадению напоминает поиск в глубину в том отношении, что этот алгоритм предпочитает на пути к цели постоянно следовать по единственному пути, но возвращается к предыдущим узлам после попадания в тупик. Данный алгоритм страдает от тех же недостатков, что и алгоритм поиска в глубину: он не является оптимальным, к тому же он — не полный (поскольку способен отправиться по бесконечному пути, да так и не вернуться, чтобы опробовать другие возможности). При этом в наихудшем случае оценки временной и пространственной сложности составляют О (nm) , где m — максимальная глубина пространства поиска.

Временна́я сложность алгоритма A\* зависит от эвристики. В худшем случае, число вершин, исследуемых алгоритмом, растёт экспоненциально по сравнению с длиной оптимального пути, но сложность становится полиномиальной, когда эвристика удовлетворяет следующему условию: |h ( n ) − h ∗ ( n ) | ≤ O (logh ∗ (n)); где h\* — оптимальная эвристика, то есть точная оценка расстояния из вершины x к цели. Другими словами, ошибка h(n) не должна расти быстрее, чем логарифм от оптимальной эвристики.

В заключении можно сказать, что за счет своей скорости и гибкости оба алгоритма могут быть очень полезными в практике, а в некторых случаях даже незаменимыми.