**Міністерство освіти і науки України**

**Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського"**

**Факультет інформатики та обчислювальної техніки**

**Кафедра інформатики та програмної інженерії**

**Звіт**

з лабораторної роботи № 2 з дисципліни

«Проектування алгоритмів»

**«Неінформативний, інформативний та локальний пошук»**

**Виконав(ла)**

(шифр, прізвище, ім'я, по батькові)

*Титаренко Данило Олегович*

**Перевірив**

(прізвище, ім'я, по батькові)

*Сопов О. О.*

Київ 2022

Зміст

[1 Мета лабораторної роботи 3](#_Toc81070688)

[2 ЗаВдання 4](#_Toc81070689)

[3 Виконання 8](#_Toc81070690)

[3.1 Псевдокод алгоритмів 8](#_Toc81070691)

[3.2 Програмна реалізація 8](#_Toc81070692)

[3.2.1 Вихідний код 8](#_Toc81070693)

[3.2.2 Приклади роботи 8](#_Toc81070694)

[3.3 Дослідження алгоритмів 8](#_Toc81070695)

[Висновок 11](#_Toc81070696)

[Критерії оцінювання 12](#_Toc81070697)

# Мета лабораторної роботи

Мета роботи – розглянути та дослідити алгоритми неінформативного, інформативного та локального пошуку. Провести порівняльний аналіз ефективності використання алгоритмів.

# ЗаВдання

Записати алгоритм розв’язання задачі у вигляді псевдокоду, відповідно до варіанту (таблиця 2.1).

Реалізувати програму, яка розв’язує поставлену задачу згідно варіанту (таблиця 2.1) за допомогою алгоритму неінформативного пошуку **АНП**, алгоритму інформативного пошуку **АІП,** що використовує задану евристичну функцію Func, або алгоритму локального пошуку **АЛП та бектрекінгу,** що використовує задану евристичну функцію Func.

Програму реалізувати на довільній мові програмування.

**Увага!** Алгоритм неінформативного пошуку **АНП,** реалізовується за принципом «AS IS», тобто так, як є, без додаткових модифікацій (таких як перевірка циклів, наприклад).

Провести серію експериментів для вивчення ефективності роботи алгоритмів. Кожний експеримент повинен відрізнятись початковим станом. Серія повинна містити не менше 20 експериментів для кожного алгоритму. Початковий стан зафіксувати у таблиці експериментів. За проведеними серіями необхідно визначити:

* середню кількість етапів (кроків), які знадобилось для досягнення розв’язку (ітерації);
* середню кількість випадків, коли алгоритм потрапляв в глухий кут (не міг знайти оптимальний розв’язок) – якщо таке можливе;
* середню кількість згенерованих станів під час пошуку;
* середню кількість станів, що зберігаються в пам’яті під час роботи програми.

Передбачити можливість обмеження виконання програми за часом (30 хвилин) та використання пам’яті (1 Гб).

**Використані позначення:**

* **8-ферзів** – Задача про вісім ферзів полягає в такому розміщенні восьми ферзів на шахівниці, що жодна з них не ставить під удар один одного. Тобто, вони не повинні стояти в одній вертикалі, горизонталі чи діагоналі.
* **8-puzzle** – гра, що складається з 8 однакових квадратних пластинок з нанесеними числами від 1 до 8. Пластинки поміщаються в квадратну коробку, довжина сторони якої в три рази більша довжини сторони пластинок, відповідно в коробці залишається незаповненим одне квадратне поле. Мета гри – переміщаючи пластинки по коробці досягти впорядковування їх по номерах, бажано зробивши якомога менше переміщень.
* **Лабіринт** – задача пошуку шляху у довільному лабіринті від початкової точки до кінцевої з можливими випадками відсутності шляху. Структура лабіринту зчитується з файлу, або генерується програмою.
* **LDFS** – Пошук вглиб з обмеженням глибини.
* **BFS** – Пошук вшир.
* **IDS** – Пошук вглиб з ітеративним заглибленням.
* **A\*** – Пошук А\*.
* **RBFS** – Рекурсивний пошук за першим найкращим співпадінням.
* **F1** – кількість пар ферзів, які б’ють один одного з урахуванням видимості (ферзь А може стояти на одній лінії з ферзем В, проте між ними стоїть ферзь С; тому А не б’є В).
* **F2** – кількість пар ферзів, які б’ють один одного без урахування видимості.
* **H1** – кількість фішок, які не стоять на своїх місцях.
* **H2** – Манхетенська відстань.
* **H3** – Евклідова відстань.
* **COLOR** – Задача розфарбування карти самостійно обраної країни, не менше 20 регіонів (областей). Необхідно розфарбувати карту не більше ніж у 4 різні кольори. Мається на увазі приписування кожному регіону власного кольору так, щоб кольори сусідніх регіонів відрізнялись. Використовувати евристичну функцію, яка повертає кількість пар суміжних вузлів, що мають однаковий колір (тобто кількість конфліктів). Реалізувати алгоритм пошуку із поверненнями (backtracking) для розв’язання поставленої задачі. Для підвищення швидкодії роботи алгоритму використати евристичну функцію, а початковим станом вважати випадкову вершину.
* **HILL –** Пошук зі сходженням на вершину з використанням із використанням руху вбік (на 100 кроків) та випадковим перезапуском (кількість необхідних разів запуску визначити самостійно).
* **ANNEAL** – Локальний пошук із симуляцією відпалу. Робоча характеристика – залежність температури Т від часу роботи алгоритму t. Можна розглядати лінійну залежність: T = 1000 - k∙t, де k – змінний коефіцієнт.
* **BEAM** – Локальний променевий пошук. Робоча характеристика – кількість променів k. Експерименти проводи із кількістю променів від 2 до 21.
* **MRV** – евристика мінімальної кількості значень;
* **DGR** – ступенева евристика.

Таблиця 2.1 – Варіанти алгоритмів

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **№** | **Задача** | **АНП** | **АІП** | **АЛП** | **Func** |
| 1 | Лабіринт | LDFS | A\* |  | H2 |
| 2 | Лабіринт | LDFS | RBFS |  | H3 |
| 3 | Лабіринт | BFS | A\* |  | H2 |
| 4 | Лабіринт | BFS | RBFS |  | H3 |
| 5 | Лабіринт | IDS | A\* |  | H2 |
| 6 | Лабіринт | IDS | RBFS |  | H3 |
| 7 | 8-ферзів | LDFS | A\* |  | F1 |
| 8 | 8-ферзів | LDFS | A\* |  | F2 |
| 9 | 8-ферзів | LDFS | RBFS |  | F1 |
| 10 | 8-ферзів | LDFS | RBFS |  | F2 |
| 11 | 8-ферзів | BFS | A\* |  | F1 |
| 12 | 8-ферзів | BFS | A\* |  | F2 |
| 13 | 8-ферзів | BFS | RBFS |  | F1 |
| 14 | 8-ферзів | BFS | RBFS |  | F2 |
| 15 | 8-ферзів | IDS | A\* |  | F1 |
| 16 | 8-ферзів | IDS | A\* |  | F2 |
| 17 | 8-ферзів | IDS | RBFS |  | F1 |
| 18 | Лабіринт | LDFS | A\* |  | H3 |
| 19 | 8-puzzle | LDFS | A\* |  | H1 |
| 20 | 8-puzzle | LDFS | A\* |  | H2 |
| 21 | 8-puzzle | LDFS | RBFS |  | H1 |
| 22 | 8-puzzle | LDFS | RBFS |  | H2 |
| 23 | 8-puzzle | BFS | A\* |  | H1 |
| 24 | 8-puzzle | BFS | A\* |  | H2 |
| 25 | 8-puzzle | BFS | RBFS |  | H1 |
| 26 | 8-puzzle | BFS | RBFS |  | H2 |
| 27 | Лабіринт | BFS | A\* |  | H3 |
| 28 | 8-puzzle | IDS | A\* |  | H2 |
| 29 | 8-puzzle | IDS | RBFS |  | H1 |
| 30 | 8-puzzle | IDS | RBFS |  | H2 |
| 31 | COLOR |  |  | HILL | MRV |
| 32 | COLOR |  |  | ANNEAL | MRV |
| 33 | COLOR |  |  | BEAM | MRV |
| 34 | COLOR |  |  | HILL | DGR |
| 35 | COLOR |  |  | ANNEAL | DGR |
| 36 | COLOR |  |  | BEAM | DGR |

# Виконання

## Псевдокод алгоритмів

**BFS**

optional<size\_t> bfs(const Game8Puzzle& game, Logger& logger) {

logger.startTimer();

std::size\_t res = 0;

std::queue<Game8Puzzle> q;

std::unordered\_set<Game8Puzzle> used;

used.reserve(fact9);

q.push(game);

**while** (!q.empty()) **do**

**for** int i = 0 **to** q.size() **do**

logger.IncrementConditionCount();

auto& cur\_game = q.front();

**if** (used.count(cur\_game))

q.pop();

**end if**

**else**

**if** (cur\_game.IsSolved())

logger.stopTimer();

logger.setConditionInMemoryCount(used.size());

return res;

**end if**

used.insert(cur\_game);

**if** (cur\_game.moveRight())

q.push(cur\_game);

cur\_game.moveLeft();

**end if**

**if** (cur\_game.moveLeft())

q.push(cur\_game);

cur\_game.moveRight();

**end if**

**if** (cur\_game.moveUp())

q.push(cur\_game);

cur\_game.moveDown();

**end if**

**if** (cur\_game.moveDown())

q.push(cur\_game);

**end if**

q.pop();

**end else**

logger.IncrementIterationCounter();

**end for**

res++;

**end while**

logger.stopTimer();

logger.setConditionInMemoryCount(used.size());

return std::nullopt;

}

**A\*(H2)**

std::optional<std::size\_t> AStar(const Game8Puzzle& game, Logger& logger) {

logger.startTimer();

std::unordered\_map<Game8Puzzle, std::size\_t> dist;

dist.reserve(fact9);

std::multimap<std::size\_t, std::unordered\_map<Game8Puzzle, std::size\_t>::iterator> gamesSortedByF;

gamesSortedByF.insert({ 0, dist.insert({game, 0}).first });

**while** (!gamesSortedByF.empty()) **do**

logger.IncrementConditionCount();

auto cur = gamesSortedByF.begin();

Game8Puzzle cur\_game = cur->second->first;

std::size\_t cur\_dist = cur->second->second;

gamesSortedByF.erase(cur);

cout << cur\_game << '\n';

**if** (cur\_game.IsSolved())

logger.stopTimer();

logger.setConditionInMemoryCount(dist.size());

return cur\_dist;

**end if**

**if** (cur\_game.moveRight())

auto it = dist.find(cur\_game);

**if** ( it == dist.end() || it->second > cur\_dist + 1)

**if** (it == dist.end())

it = dist.insert({ cur\_game, cur\_dist + 1 }).first;

**end if**

**else**

it->second = cur\_dist + 1;

**end else**

gamesSortedByF.insert({ cur\_game.CalcManhattanDist() + cur\_dist + 1, it });

**end if**

cur\_game.moveLeft();

**end if**

**if** (cur\_game.moveLeft()) {

auto it = dist.find(cur\_game);

**if** (it == dist.end() || it->second > cur\_dist + 1)

**if** (it == dist.end())

it = dist.insert({ cur\_game, cur\_dist + 1 }).first;

**end if**

**else**

it->second = cur\_dist + 1;

**end else**

gamesSortedByF.insert({ cur\_game.CalcManhattanDist() + cur\_dist + 1, it });

**end if**

cur\_game.moveRight();

**end if**

**if** (cur\_game.moveUp())

auto it = dist.find(cur\_game);

**if** (it == dist.end() || it->second > cur\_dist + 1)

**if** (it == dist.end())

it = dist.insert({ cur\_game, cur\_dist + 1 }).first;

**end if**

**else**

it->second = cur\_dist + 1;

**end else**

gamesSortedByF.insert({ cur\_game.CalcManhattanDist() + cur\_dist + 1, it });

**end if**

cur\_game.moveDown();

**end if**

**if** (cur\_game.moveDown())

auto it = dist.find(cur\_game);

**if** (it == dist.end() || it->second > cur\_dist + 1)

**if** (it == dist.end())

it = dist.insert({ cur\_game, cur\_dist + 1 }).first;

**end if**

**else**

it->second = cur\_dist + 1;

**end else**

gamesSortedByF.insert({ cur\_game.CalcManhattanDist() + cur\_dist + 1, it });

**end if**

**end if**

logger.IncrementIterationCounter();

**end while**

logger.stopTimer();

logger.setConditionInMemoryCount(dist.size());

return std::nullopt;

}

## Програмна реалізація

### Вихідний код

#include <iostream>

#include <array>

#include <algorithm>

#include <queue>

#include <unordered\_set>

#include <unordered\_map>

#include <map>

#include <optional>

#include <chrono>

#include <fstream>

constexpr std::size\_t EMPTY = 0;

constexpr std::size\_t fact9 = 2 \* 3 \* 4 \* 5 \* 6 \* 7 \* 8 \* 9;

using namespace std;

class Logger

{

public:

Logger() = default;

inline std::size\_t getIterationCount() const

{

return m\_itercnt;

}

inline std::size\_t getTime() const

{

return m\_time;

}

inline std::size\_t getConditionsCount() const {

return m\_condcnt;

}

inline std::size\_t getConditionsInMemoryCount() const

{

return m\_condinmemcnt;

}

inline void startTimer()

{

m\_curtime = std::chrono::high\_resolution\_clock::now();

}

inline void stopTimer()

{

using namespace std::chrono;

m\_time = duration\_cast<microseconds>(high\_resolution\_clock::now() - m\_curtime).count();

}

inline void IncrementIterationCounter()

{

m\_itercnt++;

}

inline void IncrementConditionCount()

{

m\_condcnt++;

}

inline void setConditionInMemoryCount(std::size\_t cnt)

{

m\_condinmemcnt = cnt;

}

private:

std::size\_t m\_itercnt = 0, m\_time = 0, m\_condcnt = 0, m\_condinmemcnt;

std::chrono::time\_point<std::chrono::steady\_clock> m\_curtime;

};

class Game8Puzzle {

public:

Game8Puzzle() = default;

Game8Puzzle(const std::array<std::size\_t, 9>& in\_matrix) :

m\_matrix(in\_matrix) {

calcPosOfEmpty();

}

bool moveLeft();

bool moveRight();

bool moveDown();

bool moveUp();

const std::array<std::size\_t, 9>& getMatrix() const { return m\_matrix; }

std::size\_t CalcManhattanDist() const;

std::size\_t CalcWrongPuzzles() const;

bool IsSolved() const;

bool operator==(const Game8Puzzle& r) const {

return m\_matrix == r.m\_matrix;

}

private:

void calcPosOfEmpty() const;

std::array<std::size\_t, 9> m\_matrix = { 1, 2, 3, 4, 5, 6, 7, 8, EMPTY };

mutable std::size\_t posOfEmpty = 8;

};

std::ostream& operator<<(std::ostream& out, const Game8Puzzle& game) {

out << std::string(5, '\_') << '\n';

for (int i = 0; i < 3; i++) {

out << '|';

for (int j = 0; j < 3; j++) {

out << game.getMatrix()[i \* 3 + j];

if (j < 2)

out << ' ';

}

out << "|\n";

}

out << std::string(5, '\_');

return out;

}

template<>

struct std::hash<Game8Puzzle> {

std::size\_t operator()(const Game8Puzzle& game) const {

std::size\_t res = 0;

const auto& matrix = game.getMatrix();

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

res \*= 10;

res += matrix[i];

}

return res;

}

};

void Game8Puzzle::calcPosOfEmpty() const {

for (int i = 0; i < 9; i++) {

if (m\_matrix[i] == 0) {

posOfEmpty = i;

return;

}

}

}

bool Game8Puzzle::moveLeft() {

if (posOfEmpty % 3 != 2) {

std::swap(m\_matrix[posOfEmpty], m\_matrix[posOfEmpty + 1]);

posOfEmpty++;

return true;

}

return false;

}

bool Game8Puzzle::moveRight() {

if (posOfEmpty % 3 != 0) {

std::swap(m\_matrix[posOfEmpty], m\_matrix[posOfEmpty - 1]);

posOfEmpty--;

return true;

}

return false;

}

bool Game8Puzzle::moveDown() {

if (posOfEmpty > 2) {

std::swap(m\_matrix[posOfEmpty], m\_matrix[posOfEmpty - 3]);

posOfEmpty -= 3;

return true;

}

return false;

}

bool Game8Puzzle::moveUp() {

if (posOfEmpty < 6) {

std::swap(m\_matrix[posOfEmpty], m\_matrix[posOfEmpty + 3]);

posOfEmpty += 3;

return true;

}

return false;

}

std::size\_t Game8Puzzle::CalcManhattanDist() const {

std::size\_t res = 0;

for (int i = 0; i < 9; i++) {

int cur = (m\_matrix[i] + 8) % 9;

res += std::abs(cur % 3 - i % 3) + std::abs(cur / 3 - i / 3);

}

return res;

}

bool Game8Puzzle::IsSolved() const {

for (int i = 0; i < 9; i++) {

if (i != (m\_matrix[i] + 8) % 9)

return false;

}

return true;

}

optional<size\_t> bfs(const Game8Puzzle& game, Logger& logger) {

logger.startTimer();

std::size\_t res = 0;

std::queue<Game8Puzzle> q;

std::unordered\_set<Game8Puzzle> used;

used.reserve(fact9);

q.push(game);

while (!q.empty()) {

for (int i = 0, n = q.size(); i < n; i++) {

logger.IncrementConditionCount();

auto& cur\_game = q.front();

if (used.count(cur\_game)) {

q.pop();

}

else {

cout << cur\_game << '\n';

if (cur\_game.IsSolved()) {

logger.stopTimer();

logger.setConditionInMemoryCount(used.size());

return res;

}

used.insert(cur\_game);

if (cur\_game.moveRight()) {

q.push(cur\_game);

cur\_game.moveLeft();

}

if (cur\_game.moveLeft()) {

q.push(cur\_game);

cur\_game.moveRight();

}

if (cur\_game.moveUp()) {

q.push(cur\_game);

cur\_game.moveDown();

}

if (cur\_game.moveDown()) {

q.push(cur\_game);

}

q.pop();

}

logger.IncrementIterationCounter();

}

res++;

}

logger.stopTimer();

logger.setConditionInMemoryCount(used.size());

return std::nullopt;

}

std::optional<std::size\_t> AStar(const Game8Puzzle& game, Logger& logger) {

logger.startTimer();

std::unordered\_map<Game8Puzzle, std::size\_t> dist;

dist.reserve(fact9);

std::multimap<std::size\_t, std::unordered\_map<Game8Puzzle, std::size\_t>::iterator> gamesSortedByF;

gamesSortedByF.insert({ 0, dist.insert({game, 0}).first });

while (!gamesSortedByF.empty()) {

logger.IncrementConditionCount();

auto cur = gamesSortedByF.begin();

Game8Puzzle cur\_game = cur->second->first;

std::size\_t cur\_dist = cur->second->second;

gamesSortedByF.erase(cur);

cout << cur\_game << '\n';

if (cur\_game.IsSolved()) {

logger.stopTimer();

logger.setConditionInMemoryCount(dist.size());

return cur\_dist;

}

if (cur\_game.moveRight()) {

auto it = dist.find(cur\_game);

if ( it == dist.end() || it->second > cur\_dist + 1) {

if (it == dist.end())

it = dist.insert({ cur\_game, cur\_dist + 1 }).first;

else

it->second = cur\_dist + 1;

gamesSortedByF.insert({ cur\_game.CalcManhattanDist() + cur\_dist + 1, it });

}

cur\_game.moveLeft();

}

if (cur\_game.moveLeft()) {

auto it = dist.find(cur\_game);

if (it == dist.end() || it->second > cur\_dist + 1) {

if (it == dist.end())

it = dist.insert({ cur\_game, cur\_dist + 1 }).first;

else

it->second = cur\_dist + 1;

gamesSortedByF.insert({ cur\_game.CalcManhattanDist() + cur\_dist + 1, it });

}

cur\_game.moveRight();

}

if (cur\_game.moveUp()) {

auto it = dist.find(cur\_game);

if (it == dist.end() || it->second > cur\_dist + 1) {

if (it == dist.end())

it = dist.insert({ cur\_game, cur\_dist + 1 }).first;

else

it->second = cur\_dist + 1;

gamesSortedByF.insert({ cur\_game.CalcManhattanDist() + cur\_dist + 1, it });

}

cur\_game.moveDown();

}

if (cur\_game.moveDown()) {

auto it = dist.find(cur\_game);

if (it == dist.end() || it->second > cur\_dist + 1) {

if (it == dist.end())

it = dist.insert({ cur\_game, cur\_dist + 1 }).first;

else

it->second = cur\_dist + 1;

gamesSortedByF.insert({ cur\_game.CalcManhattanDist() + cur\_dist + 1, it });

}

}

logger.IncrementIterationCounter();

}

logger.stopTimer();

logger.setConditionInMemoryCount(dist.size());

return std::nullopt;

}

int main()

{

std::ofstream out("res\_table.txt");

std::array<std::size\_t, 9> matrix({ 2,1,4,

6,7,3,

5,8,EMPTY});

for (int i = 0; i < 1; i++) {

Game8Puzzle game(matrix);

std::optional<std::size\_t> sol1, sol2;

Logger logger\_bfs, logger\_astar;

sol1 = bfs(game, logger\_bfs);

sol2 = AStar(game, logger\_astar);

out << i + 1 << " condition\t" << logger\_bfs.getIterationCount() << '\t' << !sol1.has\_value() << '\t'

<< logger\_bfs.getConditionsCount() << '\t' << logger\_bfs.getConditionsInMemoryCount() << '\t';

out<< " " << '\t'

<< logger\_astar.getIterationCount() << '\t' << !sol2.has\_value() << '\t'

<< logger\_astar.getConditionsCount() << '\t' << logger\_astar.getConditionsInMemoryCount() << '\t';

/\*

for (int j = 0, n = rand(); j < n; j++) {

std::next\_permutation(matrix.begin(), matrix.end());

}

\*/

}

}

### Приклади роботи

На рисунках 3.1 і 3.2 показані приклади роботи програми для різних алгоритмів пошуку.

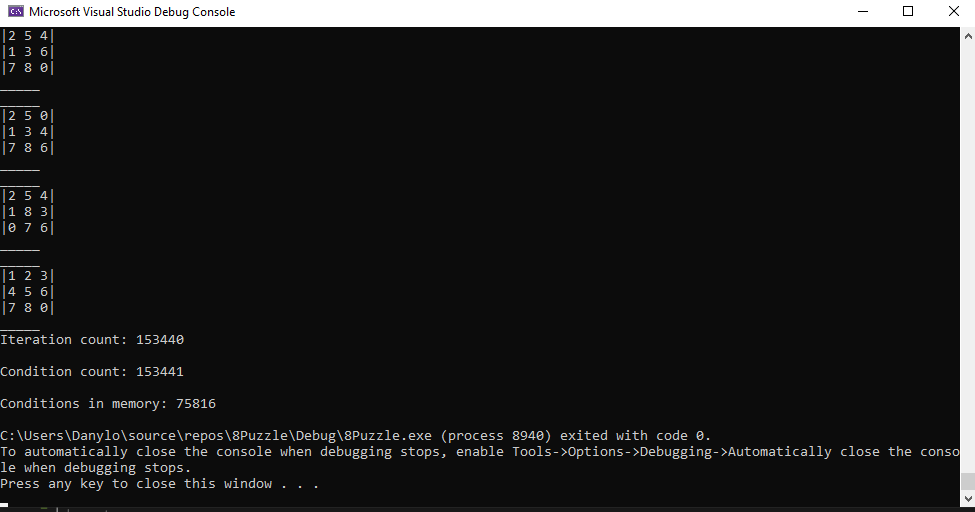


Рисунок 3.1 – Алгоритм пошук вшир



Рисунок 3.2 – Алгоритм A\* з евристикою манхетенської відстані

## Дослідження алгоритмів

В таблиці 3.1 наведені характеристики оцінювання алгоритму

Таблиця 3.1 – Характеристики оцінювання алгоритму пошуку вшир

|  |  |  |  |
| --- | --- | --- | --- |
| Початкові стани | Ітерації | Всього станів | Всього станів у пам’яті |
| Стан 1 | 153440 | 153441 | 75816 |
| Стан 2 | 58233 | 58234 | 32071 |
| Стан 3 | 158090 | 158091 | 78084 |
| Стан 4 | 129868 | 128869 | 65725 |
| Стан 5 | 34640 | 34641 | 19726 |
| Стан 6 | 36637 | 36638 | 20506 |
| Стан 7 | 16991 | 16992 | 9861 |
| Стан 8 | 166133 | 166134 | 82080 |
| Стан 9 | 122250 | 122251 | 61911 |
| Стан 10 | 205723 | 205724 | 99434 |
| Стан 11 | 158090 | 158091 | 78084 |
| Стан 12 | 382282 | 382283 | 160612 |
| Стан 13 | 129867 | 129868 | 65725 |
| Стан 14 | 270665 | 270666 | 122886 |
| Стан 15 | 310167 | 310168 | 138248 |
| Стан 16 | 247847 | 247848 | 115197 |
| Стан 17 | 32854 | 32855 | 20384 |
| Стан 18 | 57257 | 57258 | 31481 |
| Стан 19 | 200483 | 200484 | 103751 |
| Стан 20 | 15680 | 15681 | 9142 |
| **СЕРЕДНЄ** | **144360** | **144361** | **69536** |

Таблиця 3.2 – Характеристики оцінювання алгоритму A\* з евристикою манхетенської відстані

|  |  |  |  |
| --- | --- | --- | --- |
| Початкові стани | Ітерації | Всього станів | Всього станів у пам’яті |
| Стан 1 | 1291 | 1292 | 2076 |
| Стан 2 | 336 | 337 | 538 |
| Стан 3 | 854 | 855 | 1372 |
| Стан 4 | 876 | 877 | 1388 |
| Стан 5 | 494 | 495 | 797 |
| Стан 6 | 586 | 587 | 807 |
| Стан 7 | 2880 | 2881 | 4527 |
| Стан 8 | 8713 | 8714 | 13503 |
| Стан 9 | 1750 | 1751 | 2810 |
| Стан 10 | 2701 | 2702 | 4216 |
| Стан 11 | 597 | 598 | 969 |
| Стан 12 | 1995 | 1996 | 3178 |
| Стан 13 | 486 | 487 | 743 |
| Стан 14 | 1446 | 1447 | 2315 |
| Стан 15 | 930 | 931 | 1506 |
| Стан 16 | 1569 | 1570 | 2486 |
| Стан 17 | 763 | 764 | 1230 |
| Стан 18 | 1318 | 1319 | 2083 |
| Стан 19 | 3986 | 3987 | 6270 |
| Стан 20 | 1319 | 1320 | 2101 |
| **СЕРЕДНЄ** | **1744** | **1745** | **2746** |

Висновок

При виконанні даної лабораторної роботи було розглянуто алгоритми неінформативного, інформативного та локального пошуків. Особливістю моєї лабораторної роботи було дослідження алгоритму пошуку вшир та алгоритму А\* з евристикою Н2. Було проведено порівняльний аналіз ефективності алгоритмів, при якому виявлено що пошук вшир є неоптимальним алгоритмом і має в середньому більшу к-ть станів та більшу к-ть станів у пам’яті ніж А\*.

Отже, А\*(інформативний пошук) є більш ефективний ніж BFS(неінформативний)

Критерії оцінювання

За умови здачі лабораторної роботи до 23.10.2022 включно максимальний бал дорівнює – 5. Після 23.10.2022 максимальний бал дорівнює – 1.

Критерії оцінювання у відсотках від максимального балу:

* псевдокод алгоритму – 10%;
* програмна реалізація алгоритму – 60%;
* дослідження алгоритмів – 25%;
* висновок – 5%.