**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)

*ІП-12 Мельник Михайло*

**Перевірив**

(прізвище, ім'я, по батькові)

*Сопов О.О.*

Київ 2022

Зміст

[1 Мета лабораторної роботи 3](#_Toc23720)

[2 ЗаВдання 4](#_Toc23276)

[3 Виконання 7](#_Toc29273)

[3.1 Псевдокод алгоритмів 7](#_Toc1172)

[3.1.1 Пошук з обмеженням глибини 7](#_Toc22223)

[3.1.2 A\* 8](#_Toc19983)

[3.2 Програмна реалізація 9](#_Toc17516)

[3.2.1 Вихідний код 9](#_Toc6953)

[3.2.2 Приклади роботи 11](#_Toc16095)

[3.3 Дослідження алгоритмів 12](#_Toc7081)

[Висновок 15](#_Toc11896)

[Критерії оцінювання 16](#_Toc14260)

# 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).

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

Варіант 18

Лабірин, пошук з обмеженням глибини, пошук А\* з евристикою евклідова відстань

## Псевдокод алгоритмів

### Пошук з обмеженням глибини

PROCEDURE LDFS\_Search(maze, limit)

start := (maze.rows, maze.cols)

stack = stack()

stack.append((start, 0))

visited = dict()

WHILE NOT stack.empty() DO

(curr, c\_l) = stack.pop()

IF c\_l == limit THEN CONTINUE

IF curr == (1, 1) THEN

visited[curr] = c\_l

RETURN Reconstruct(visited)

IF curr IN visited THEN CONTINUE

visited[curr] = c\_l

stack = stack + Expand(maze, curr, c\_l)

RETURN FALSE

END PROCEDURE LDfS\_Search

PROCEDURE Reconstruct(visited)

c, c\_l = visited.pop()

path = list()

path.append(c)

n, n\_l = visited.pop()

WHILE NOT visited.empty() DO

IF c\_l - n\_l == 1 THEN

path.append(n)

c, c\_l = n, n\_l

n, n\_l = visited.pop()

path.append(n)

reverse(path)

RETURN path

END PROCEDURE Reconstruct

PROCEDURE Expand(maze, curr, c\_l)

neighbours = list()

FOR d IN mazeDirections DO

IF no wall in direction d THEN

calculate neighbour

neighbours.append((neighbour, c\_l + 1))

RETURN neighbours

END PROCEDURE Expand

### A\*

PROCEDURE Search(maze)

start = (maze.rows, maze.cols)

q\_score = {cell: INFINITY FOR cell IN maze}

g\_score[start] = 0

f\_score = {cell: INFINITY FOR cell IN maze}

f\_score[start] = h(start, (1, 1))

open = PriorityQueue()

open.put((self.\_\_h(start, (1, 1)), start)0

came\_from = dict()

WHILE NOT open.empty() DO

curr = open.get()[1]

if curr == (1, 1) THEN

RETURN Reconstruct(came\_from, start)

FOR neighbour in Expand(maze, curr) DO

tent\_g\_score = g\_score[curr] + 1

tent\_f\_score = tent\_g\_score + h(neighbour, (1, 1))

IF tent\_f\_score < f\_score[neighbour] THEN

g\_score[neighbour] = tent\_g\_score

f\_score[neighbour] = tent\_f\_score

open.put((tent\_f\_score, neighbour))

came\_from[neighbour] = curr

RETURN FALSE

END PROCEDURE Search(maze)

PROCEDURE h(a, b)

RETURN SQRT((a[0] – b[0])\*\*2 + (a[1] – b[1])\*\*2)

END PROCEDURE h

PROCEDURE Expand(maze, curr)

neighbours = list()

FOR d IN mazeDirections DO

IF no wall in direction d THEN

calculate neighbour

neighbours.append(neighbour)

RETURN neighbours

END PROCEDURE Expand

PROCEDURE Reconstruct(came\_from, start)

path = dict()

curr = (1, 1)

WHILE curr != start DO

path[came\_from[curr]] = curr

curr = came\_from[curr]

RETURN path

END PROCEDURE Reconstruct

## Програмна реалізація

### Вихідний код

class LDFSearcher:

def \_\_init\_\_(self):

pass

@staticmethod  
def \_expand(m, curr, c\_l):  
 directions = {"E": lambda x: (x[0], x[1] + 1),  
 "W": lambda x: (x[0], x[1] - 1),  
 "N": lambda x: (x[0] - 1, x[1]),  
 "S": lambda x: (x[0] + 1, x[1])}  
 neighbours = []  
 for d in "ENSW":  
 if m.maze\_map[curr][d]:  
 neighbours += (directions[d](curr), c\_l+1),  
 return neighbours  
  
@staticmethod  
def \_reconstruct(visited):  
 c, c\_l = visited.popitem() # cell and level  
 path = [c]  
 n, n\_l = visited.popitem()  
 while visited:  
 if c\_l - n\_l == 1: # if delta levels == 1 means the very found path  
 path += n,  
 c, c\_l = n, n\_l  
 n, n\_l = visited.popitem()

path == n,  
 return list(reversed(path))

def classic\_search(self, m, limit):  
 *"""Algorithm with no features tracked"""* start = (m.rows, m.cols)  
 stack = [(start, 0)]  
 visited = {}  
 while stack:  
 curr, c\_l = stack.pop() # cell and level  
 if c\_l == limit:  
 continue  
 if curr == (1, 1):  
 visited[curr] = c\_l  
 return self.\_reconstruct(visited)  
 if curr in visited:  
 continue  
 visited[curr] = c\_l  
 stack += self.\_expand(m, curr, c\_l)  
 return []

class AStarSearcher:

def \_\_init\_\_(self):

pass

@staticmethod  
def \_\_h(a, b):  
 return math.sqrt((a[0]-b[0])\*\*2 + (a[1] - b[1])\*\*2)  
  
@staticmethod  
def \_expand(m, curr):  
 directions = {"E": lambda x: (x[0], x[1] + 1),  
 "W": lambda x: (x[0], x[1] - 1),  
 "N": lambda x: (x[0] - 1, x[1]),  
 "S": lambda x: (x[0] + 1, x[1])}  
 neighbours = []  
 for d in "ENSW":  
 if m.maze\_map[curr][d]:  
 neighbours += directions[d](curr),  
 return neighbours  
  
@staticmethod  
def \_reconstruct(came\_from, start):  
 path = {}  
 curr = (1, 1)  
 while curr != start:  
 path[came\_from[curr]] = curr  
 curr = came\_from[curr]  
 return path

def classic\_search(self, m):  
 *"""Algorithm with no features tracked"""* start = (m.rows, m.cols)  
  
 g\_score = {cell: float('inf') for cell in m.grid}  
 g\_score[start] = 0  
  
 f\_score = {cell: float('inf') for cell in m.grid}  
 f\_score[start] = self.\_\_h(start, (1, 1))  
  
 open = PriorityQueue()  
 open.put((self.\_\_h(start, (1, 1)), start))  
  
 came\_from = {}  
  
 while not open.empty():  
  
 curr = open.get()[1]  
 if curr == (1, 1):  
 return self.\_reconstruct(came\_from, start)  
  
 for neighbour in self.\_expand(m, curr):  
 tentative\_g\_score = g\_score[curr] + 1  
 tentative\_f\_score = tentative\_g\_score + self.\_\_h(neighbour, (1, 1))  
  
 if tentative\_f\_score < f\_score[neighbour]:  
 g\_score[neighbour] = tentative\_g\_score  
 f\_score[neighbour] = tentative\_f\_score  
 open.put((tentative\_f\_score, neighbour))  
 came\_from[neighbour] = curr  
  
 return []

### Приклади роботи

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

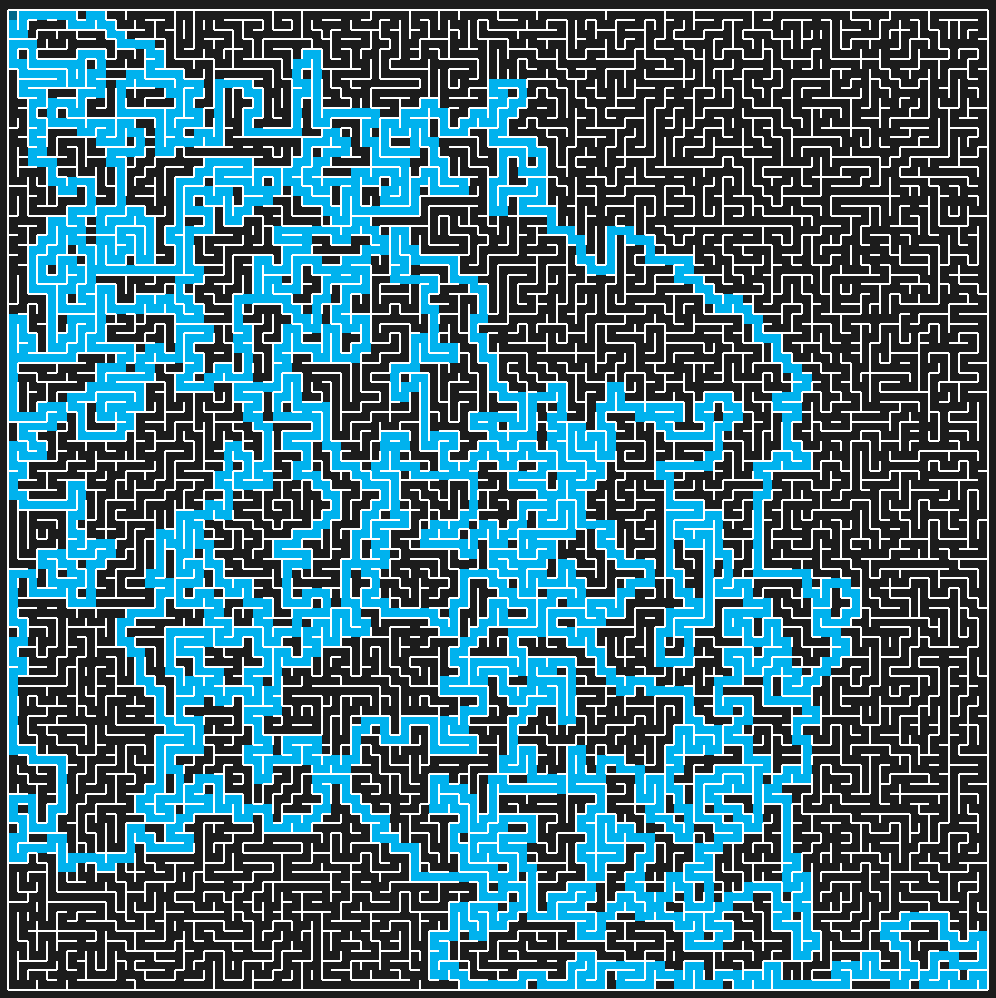
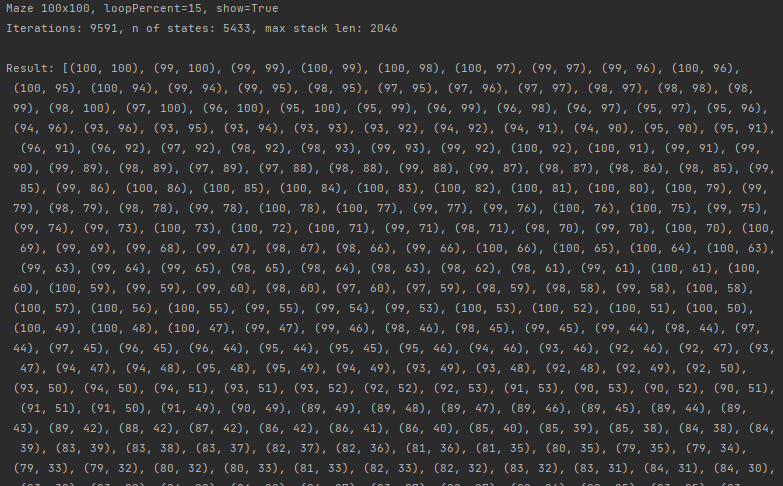


Рисунок 3.1 – Алгоритм пошуку з обмеженням глибини

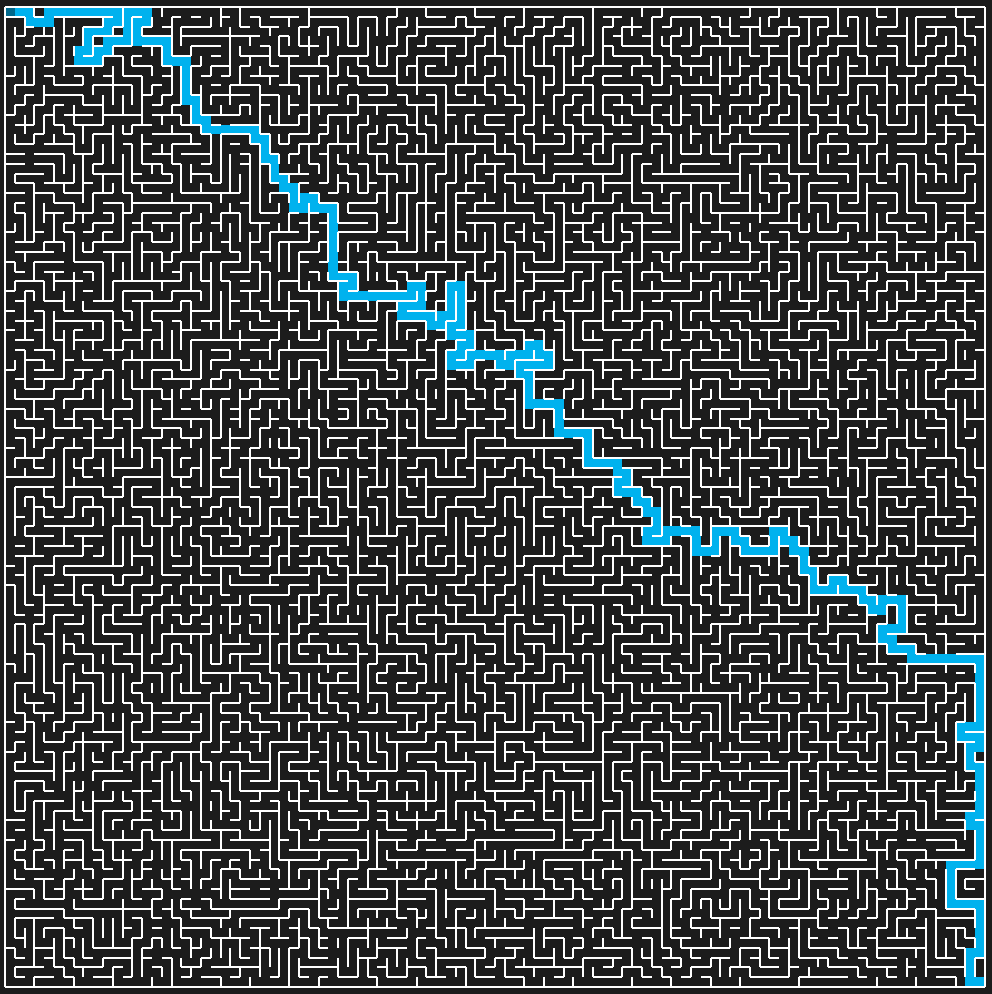
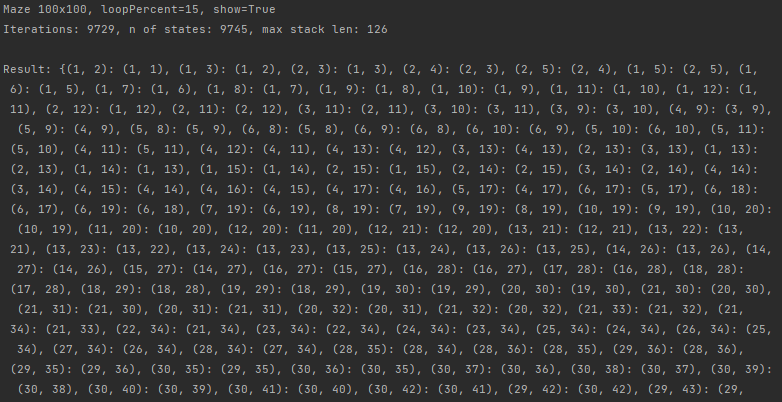


Рисунок 3.2 – Алгоритм пошуку А\*

## Дослідження алгоритмів

В таблиці 3.1 наведені характеристики оцінювання алгоритму пошуку з обмеженням глибини задачі лабіринту для 20 початкових станів.

Таблиця 3.1 – Характеристики оцінювання алгоритму пошуку з обмеженням глибини

|  |  |  |  |
| --- | --- | --- | --- |
| Початкові стани | Ітерації | Всього станів | Всього станів у пом’яті |
| Стан 1 | 1711 | 1050 | 544 |
| Стан 2 | 17382 | 8679 | 2436 |
| Стан 3 | 5175 | 3047 | 1332 |
| Стан 4 | 18244 | 8929 | 1764 |
| Стан 5 | 8229 | 4673 | 1786 |
| Стан 6 | 2726 | 1590 | 682 |
| Стан 7 | 1357 | 870 | 534 |
| Стан 8 | 6064 | 3456 | 1348 |
| Стан 9 | 1793 | 1099 | 560 |
| Стан 10 | 2094 | 1244 | 577 |
| Стан 11 | 3944 | 2244 | 857 |
| Стан 12 | 7870 | 4377 | 1464 |
| Стан 13 | 5682 | 3087 | 907 |
| Стан 14 | 7389 | 4251 | 1772 |
| Стан 15 | 6262 | 3315 | 830 |
| Стан 16 | 19494 | 9354 | 2410 |
| Стан 17 | 8512 | 4818 | 1774 |
| Стан 18 | 3105 | 1855 | 870 |
| Стан 19 | 7683 | 4395 | 1720 |
| Стан 20 | 1783 | 1076 | 542 |
| СЕРЕДНЄ | 6825 | 3680 | 1235 |

В таблиці 3.2 наведені характеристики оцінювання алгоритму пошуку А\*, задачі Лабіринту для 20 початкових станів.

Таблиця 3.3 – Характеристики оцінювання пошуку А\*

|  |  |  |  |
| --- | --- | --- | --- |
| Початкові стани | Ітерації | Всього станів | Всього станів у пом’яті |
| Стан 1 | 8843 | 8900 | 102 |
| Стан 2 | 8369 | 8442 | 102 |
| Стан 3 | 8157 | 8239 | 101 |
| Стан 4 | 8164 | 8247 | 117 |
| Стан 5 | 9077 | 9112 | 111 |
| Стан 6 | 8710 | 8764 | 128 |
| Стан 7 | 8595 | 8615 | 111 |
| Стан 8 | 9302 | 9353 | 117 |
| Стан 9 | 8639 | 8700 | 112 |
| Стан 10 | 8382 | 8436 | 117 |
| Стан 11 | 8703 | 8783 | 103 |
| Стан 12 | 9789 | 9787 | 94 |
| Стан 13 | 9353 | 9380 | 117 |
| Стан 14 | 9665 | 9695 | 99 |
| Стан 15 | 9406 | 9430 | 100 |
| Стан 16 | 8522 | 8605 | 108 |
| Стан 17 | 7339 | 7425 | 109 |
| Стан 18 | 8000 | 8076 | 131 |
| Стан 19 | 7137 | 7255 | 125 |
| Стан 20 | 9957 | 9953 | 98 |
| СЕРЕДНЄ | 8705 | 8760 | 110 |

Висновок

При виконанні даної лабораторної роботи було розглянуто алгоритми пошуку в глибину з обмеженням глибини та А\* для задачі лабіринту, було здійснено програмну реалізацію цих алгоритмів. Було здійснено 20 експерементів для кожного із алгоритмів і зафіксовано кількість ітерацій, кількість пройдених станів та максимальну кількість станів у пам’яті.

Зроблено висновок, що пошук з обмеженням глибини є неповним і неоптимальним алгоритмом, коли пошук А\* є повним і оптимальним допоки еврестична функція є прийнятною. Було отримано, що пошук А\* має меншу кількість станів у пам’яті, але в середньому розгортає більше станів та здійснює більше ітерацій.

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%.