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) *ІП-13 Лисенко Анастасія*
(шифр, прізвище, ім'я, по батькові)

Перевірив *Сопов О.О.*
(прізвище, ім'я, по батькові)

Київ 2022

ЗМІСТ

1 МЕТА ЛАБОРАТОРНОЇ РОБОТИ

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

2 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 A*.
- **RBFS** – Рекурсивний пошук за першим найкращим співпадінням.
- **F1** – кількість пар ферзів, які б’ють один одного з урахуванням видимості (ферзь A може стояти на одній лінії з ферзем B, проте між ними стоїть ферзь C; тому A не б’є B).
- **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 T від часу роботи алгоритму t. Можна розглядати лінійну залежність: $T = 1000 - k \cdot t$, 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

3 ВИКОНАННЯ

3.1 Псевдокод алгоритмів

IDS:

```
Function Iterative-Deepening-Search(problem) returns рішення result або індикатор невдачі failure
        for depth ← 0 to ∞ do
            result ← Depth-Limited-Search(problem, depth)
            if result != cutoff then return рішення result end if
        end for

Function Depth-Limited-Search (problem, limit) returns рішення result або індикатор невдачі failure\cutoff
        Return Recursive-DLS( Make-Node (Initial-State[problem]), Problem, limit)

Function Recursive-DLS(node, problem, limit) returns рішення result або індикатор невдачі failure\cutoff
        cutoff_occurred? ← неправдиве значення
        if Goal-Test[problem](State[node]) then return Solution(node)
        else if Deptbh[node] = limit then return індикатор невдачі cutoff
        else for each спадкоємець successor in Expand(node, problem)do
            result ← Recursive-DLS(successor, problem, limit)
            if result = cutoff then cutoff_occured? ← правдиве значення
            else if result != failure then return рішення result
        if cutoff_occurred?
            Then return індикатор невдачі cutoff
        Else return індикатор невдачі failure
```

A*:

```
Function AStar(initialNode: ProblemNode): ProblemNode or NULL
    open: orderedArray(comparator(depth + conflictNum))
```

```
        closed: orderedArray(comparator(depth + conflictNum))
        open.insert(initialNode)
        while (open.length != 0) do
            current = open.extractMin()
            if (calculateConflictNum(current.placements) == 0) do
                return placements
            end if
            closed.insert(current)
            children = current.generateChildren()
            for child in children do
                if (!closed.binarySearchContains(child)) do
                    open.insert(child)
                end if
            end for
        end while
        return NULL
    end
```

3.2 Програмна реалізація

3.2.1 Вихідний код

main.py

```
from helperFunctions import getInput, generateBoard, printBoard
import time
from Algorithms import Algorithms

size, chosenAlgo = getInput()
state = generateBoard(size)
print("\n------------Initial state-------------\n")
printBoard(state)

algo = Algorithms(state, size)
if chosenAlgo == 'i':
    algo.IDS()
    solution = algo.solutions
    currTime = time.time() - algo.timerStart
    print("\n------------The solution-------------\n")
    printBoard(solution)
    print(f"\n Iterations: {algo.iterAmn}\n Nodes Created:
{algo.nodesCreatedAmn}\n Nodes Saved: {algo.nodesSavedAmn}\n
Amount of Dead-ends: {algo.deadEnds}\n Time Spent: {currTime}\n")
```

```
if chosenAlgo == 'a':
    algo.A_star()
    solution = algo.solutions
    currTime = time.time() - algo.timerStart
    print("\n------------The solution-------------\n")
    printBoard(solution)
    print(f"\n Iterations: {algo.iterAmn}\n Nodes Created:
{algo.nodesCreatedAmn}\n Nodes Saved: {algo.nodesSavedAmn}\n
Amount of Dead-ends: {algo.deadEnds}\n Time Spent: {currTime}\n")
```

Node.py

```
import sys

class Node:
    def __init__(self, state, depth:int, node):
        self.state = state
        self.depth = depth
        self.parent = node
        self.bytesUsed = 0

    def AddBytesUsed(self, boardSize):
        self.bytesUsed +=sys.getsizeof(int())*(boardSize + 1)

    def getState(self):
        return self.state

    def getDepth(self):
        return self.depth

    def getParent(self):
        return self.parent
```

Algorithms.py

```
from copy import deepcopy
import time
from Node import Node
from State import State
from queue import PriorityQueue
from PrioritizedItem import PrioritizedItem
from helperFunctions import printBoard
from constants import GBToBytes, ThirtyMinutes

class Algorithms:

    def __init__(self, initial_state, size):

        self.root = Node(initial_state, 0, None)
        self.path = []
        self.boardSize = size
        self.iterAmn = 0
        self.nodesCreatedAmn = 1
```

```
        self.nodesSavedAmn = 1
        self.root.AddBytesUsed(self.boardSize)
        self.timerStart = time.time()
        self.deadEnds = 0

    def Expand(self, node):
        currDepth = node.getDepth()
        children = []
        if currDepth == self.boardSize:
            return children
        for col in range(0, self.boardSize):
            newBoard = deepcopy(node.getState())
            newBoard.posititonQueen(currDepth, col)
            children.append(Node(newBoard, currDepth+1, node))
        return children


    def DLS(self, node:Node, lim, currNodesInMemory):

        self.iterAmn+=1
        isCutoff = False
        if node.getState().isReady():
            self.solutions:State = deepcopy(node.getState())
            self.__getStates(node)

            return 0
        if node.getDepth() == lim:
            currNodesInMemory-=1
            return 2

        children = self.Expand(node)

        self.nodesCreatedAmn += len(children)
        currNodesInMemory += len(children)
        if currNodesInMemory > self.nodesSavedAmn:
            self.nodesSavedAmn = currNodesInMemory
        currTime = time.time() - self.timerStart
        if currNodesInMemory * self.root.bytesUsed > GBToBytes or
currTime > ThirtyMinutes:
            self.deadEnds+=1
            return 1
        for i in range(0, len(children)):
            result = self.DLS(children[i], lim, currNodesInMemory)
            if result == 2:
                isCutoff = True
            else:
                if result != 1:
                    return result

        currNodesInMemory-=1
        if isCutoff:
            return 2
        else:
```

```
            return 1


    def IDS(self):
        for i in range(self.boardSize):
            currNodeSaved = 0
            result = self.DLS(self.root, i, currNodeSaved)
            if result == 0:
                return True
            else:
                if result == 1:
                    return False
        return False

    def A_star(self):
        queue = PriorityQueue(0)
        queue.put(PrioritizedItem(self.root.getState().F1(),
self.root))
        while queue.qsize() >0:
            self.iterAmn+=1
            curr = queue.get()
            if curr.item.getState().isReady():
                self.solutions: State =
deepcopy(curr.item.getState())
                self.__getStates(curr.item)
                return True
            children = self.Expand(curr.item)
            self.nodesCreatedAmn += len(children)
            currTime = time.time() - self.timerStart
            if queue.qsize() * self.root.bytesUsed > GBToBytes or
currTime > ThirtyMinutes:
                self.deadEnds += 1
                return 1
            for i in range(len(children)):

queue.put(PrioritizedItem(children[i].getState().F1(),
children[i]))
            if queue.qsize() > self.nodesSavedAmn:
                self.nodesSavedAmn = queue.qsize()

        return False


    def __getStates(self, node):
        lst = []
        while node.getParent() is not None:
            lst.append(node.getState())
            node = node.getParent()
        print("\n------------Solution start-------------\n")
        for i in range(len(lst)-1, -1, -1):
            printBoard(lst[i])
        print("\n------------Solution end-------------\n")
```

PrioritizedItem.py

```
from dataclasses import dataclass, field
from Node import Node

@dataclass(order=True)
class PrioritizedItem:
    priority: int
    item:Node=field(compare=False)
```

State.py

```
     class State:
    def __init__(self, n):
        self.boardSize = n
        self.board = [0]*self.boardSize

    def getBoard(self):
        return self.board

    def isReady(self):
        for i in range(1, self.boardSize):
            for j in range (i-1, -1, -1):
                if self.board[i] == self.board[j]:
                    return False
                differance = i - j
                if self.board[i] - differance == self.board[j] or
self.board[i] + differance == self.board[j]:
                    return False
        return True

    def isByRules(self, row, col):
        for i in range(col-1, -1, -1):
            if row == self.board[i]:
                return False
            differance = col - i
            if row - differance == self.board[i] or row +
differance == self.board[i]:
                return False
        return True

    def posititonQueen(self, row, col):
        self.board[row] = col

    def F1(self):
        result = 0
        for i in range(self.boardSize):
            foundOnRow = False
            foundOnMainDiag = False
            foundOnSideDiag = False
            for j in range(i+1, self.boardSize):
```

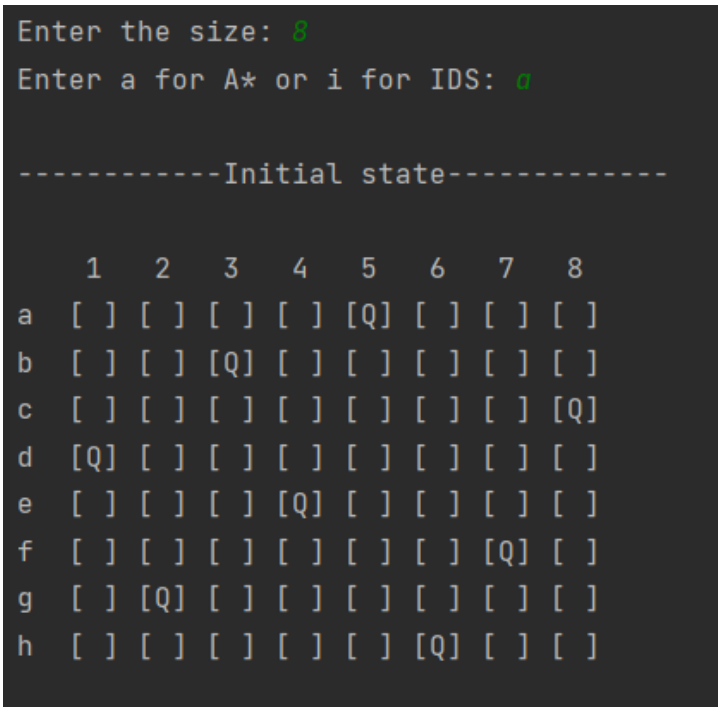
```
                if not foundOnRow and self.board[j] ==
self.board[i]:
                    foundOnRow = True
                    result+=1
                if not foundOnMainDiag and i - self.board[i] == j
- self.board[j]:
                    foundOnMainDiag = True
                    result+=1
                if not foundOnSideDiag and i + self.board[i] == j
+ self.board[j]:
                    foundOnSideDiag = True
                    result+=1
        return result


helperFunctions.py

from State import State
import random
import string

def getInput():
    n = int(input("Enter the size: "))
    chosenAlgo = input("Enter a for A* or i for IDS: ")
    while chosenAlgo != 'a' and chosenAlgo != 'i':
        chosenAlgo = input("Enter correct letter: ")
    return n, chosenAlgo

def generateBoard(size):
    state = State(size)
    state.board = random.sample(range(0, size), size)

    return state

def printBoard(state):
    alphabet = list(string.ascii_lowercase)
    for k in range(0, state.boardSize+1):
        if k == 0:
            print(' ', end='   ')
        else:
            print(k, end='   ')
    print('')
    for i in range(0, state.boardSize):
        if i < (len(alphabet)-1):
            print(alphabet[i], end='  ')
        else:
            print(alphabet[i%len(alphabet)], end=' ')
        for j in range(0, state.boardSize):
            if state.board[i] == j:
                print('[Q]', end=' ')
            else:
                print('[ ]', end=' ')
        print('')
```

constants.py

```
GBToBytes = 1024 ** 3
ThirtyMinutes = 30*60000
```

3.2.2 Приклади роботи

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



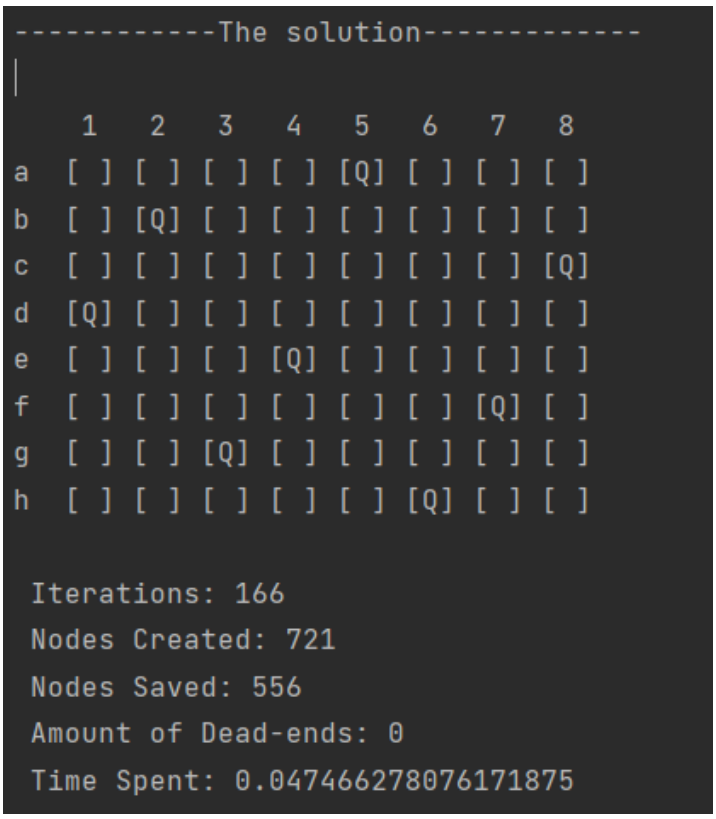
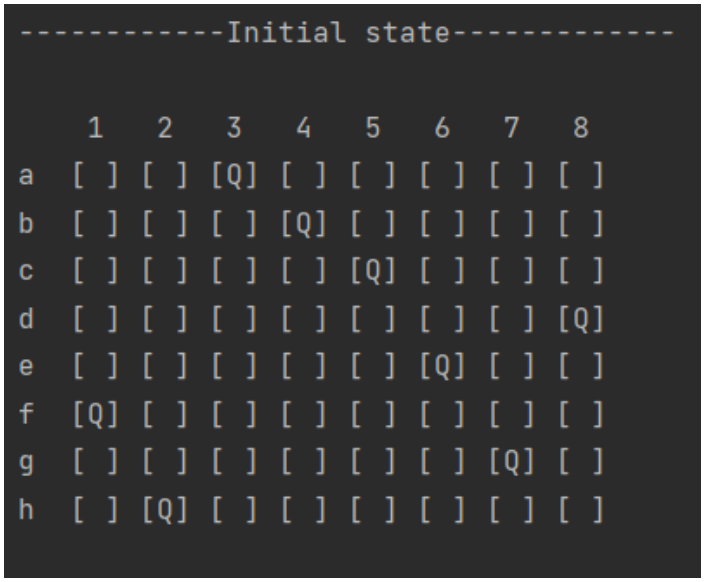


Рисунок 3.1 – Алгоритм A*



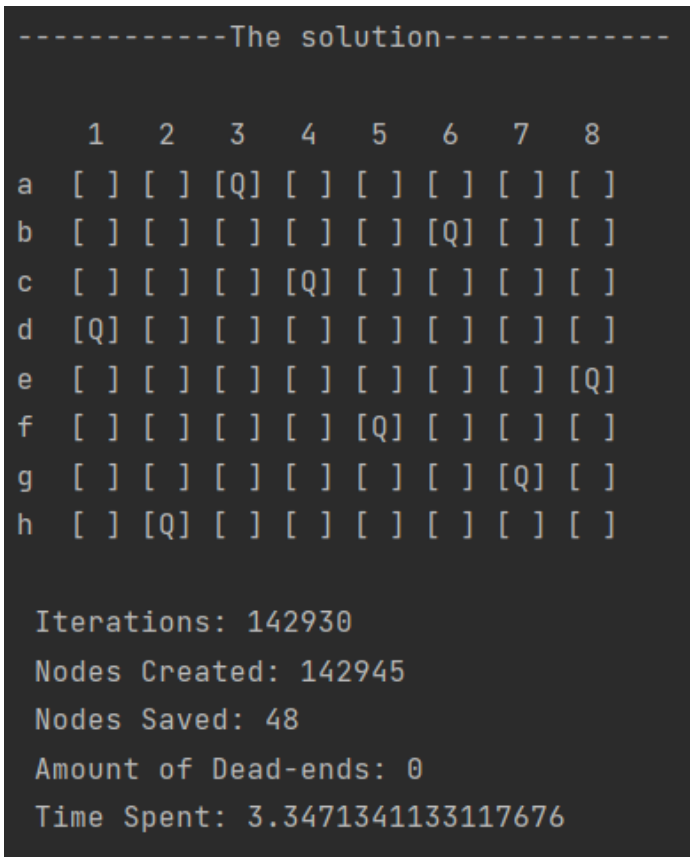


Рисунок 3.2 – Алгоритм IDS

3.3 Дослідження алгоритмів

В таблиці 3.1 наведені характеристики оцінювання алгоритму A*, для задачі про 8 ферзів для 20 початкових станів.

Таблиця 3.1 – Характеристики оцінювання алгоритму A*

№	Початкові стани	Ітерації	К-сть гл. кутів	Всього станів	Всього станів у пом'яті
1	[4, 6, 7, 2, 0, 5, 3, 1]	1128	0	3697	2570
2	[6, 7, 1, 3, 4, 2, 0, 5]	2426	0	5929	3504
3	[6, 3, 7, 5, 4, 2, 0, 1]	9402	0	12617	3246
4	[0, 3, 6, 4, 7, 1, 2, 5]	814	0	2385	1572
5	[4, 7, 5, 6, 0, 2, 3, 1]	41158	0	59033	17876
6	[0, 7, 6, 2, 3, 5, 4, 1]	27497	0	42353	14860
7	[7, 3, 2, 0, 5, 1, 4, 6]	250	0	1193	944
8	[6, 5, 2, 4, 7, 3, 1, 0]	65	0	401	337
9	[1, 5, 6, 3, 7, 2, 0, 4]	49613	0	73393	23781
10	[4, 6, 3, 1, 0, 5, 7, 2]	23	0	177	155
11	[2, 7, 0, 6, 3, 1, 5, 4]	34	0	265	232
12	[4, 2, 7, 5, 6, 3, 0, 1]	111	0	297	187
13	[6, 0, 3, 7, 5, 4, 2, 1]	7902	0	13849	5948
14	[5, 0, 1, 7, 2, 4, 3, 6]	1639	0	3289	1651
15	[2, 5, 3, 0, 1, 4, 7, 6]	8068	0	11873	3834
16	[7, 5, 1, 6, 2, 4, 3, 0]	6145	0	8817	2686
17	[1, 7, 5, 2, 6, 4, 3, 0]	18771	0	34777	16009
18	[2, 3, 4, 6, 0, 7, 1, 5]	22428	0	43073	20646
19	[4, 7, 6, 5, 2, 1, 0, 3]	8	0	57	50
20	[1, 2, 6, 4, 5, 0, 7, 3]	534	0	1425	892

В таблиці 3.2 наведені характеристики оцінювання алгоритму IDS для задачі про 8 ферзів для 20 початкових станів.

Таблиця 3.3 – Характеристики оцінювання алгоритму IDS

№	Початкові стани	Ітерації	К-сть гл. кутів	Всього станів	Всього станів у пом’яті
1	[7, 3, 4, 0, 6, 1, 5, 2]	445	0	449	24
2	[1, 3, 4, 5, 6, 7, 0, 2]	245231	0	245241	48
3	[6, 7, 5, 2, 1, 4, 3, 0]	1100555	0	1100569	56
4	[6, 1, 4, 3, 5, 2, 0, 7]	1485548	0	1485569	56
5	[1, 2, 6, 0, 5, 3, 7, 4]	17769	0	17785	40
6	[2, 1, 7, 0, 6, 4, 3, 5]	121569	0	121585	48
7	[0, 7, 1, 4, 6, 2, 3, 5]	121569	0	121585	48
8	[2, 6, 4, 3, 1, 5, 7, 0]	1100555	0	1100569	56
9	[6, 3, 5, 4, 0, 7, 1, 2]	584585	0	584601	56
10	[6, 2, 5, 3, 0, 7, 1, 4]	476	0	481	24
11	[2, 4, 1, 5, 7, 3, 0, 6]	199467	0	199481	48
12	[0, 7, 4, 2, 3, 1, 6, 5]	972534	0	972553	56
13	[7, 4, 0, 5, 3, 2, 1, 6]	227146	0	227161	48
14	[6, 1, 5, 4, 0, 3, 7, 2]	25035	0	25049	40
15	[3, 1, 6, 5, 2, 7, 0, 4]	20444	0	20457	40
16	[2, 5, 4, 7, 3, 0, 6, 1]	142930	0	142945	48
17	[7, 0, 3, 1, 6, 4, 5, 2]	73754	0	73769	48
18	[6, 2, 4, 3, 7, 0, 5, 1]	25793	0	25801	40
19	[0, 6, 1, 7, 4, 3, 5, 2]	9221	0	9233	40
20	[4, 1, 5, 2, 0, 7, 6, 3]	115968	0	115985	48

ВИСНОВОК

При виконанні даної лабораторної роботи було розглянуто та досліджено алгоритми неінформативного, інформативного та локального пошуку. Проведено порівняльний аналіз ефективності використання алгоритмів.

При виконанні свого варіанту роботи я на практиці навчилася працювати з алгоритмами IDS та A* та вирішувати задачу про 8 ферзів. При порвняльному аналізі було проведено 20 експериментів, завдяки чому ми бачимо, що в середньому IDS робить більше ітерацій та відповідно створює більше станів. A* в свою чергу в середньому зберігає більше станів у пам’яті. Задача про 8 ферзів не має глухих кутів.

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%.