**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-*

**Перевірив**

(прізвище, ім'я, по батькові)

*Головченко М.М.*

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

ФУНКЦІЯ BFS

черга = Черга()

черга.додати(початковий стан)

ПОВТОРИТИ

ПОКИ не черга.порожня()

стан = черга.вилучити()

ЯКЩО не конфлікт(стан)

ПОВЕРНУТИ стан

ВСЕ ЯКЩО

ДЛЯ стан В отримати\_нащадків(стан)

черга.додати(стан)

ВСЕ ПОВТОРИТИ

RBFS

ФУНКЦІЯ RBFS START

ПОВЕРНУТИ RBFS\_SEARCH(початковий стан, \*нескінченність, 0)

ФУНКЦІЯ RBFS MAIN

ЯКЩО не конфлікт(стан)

ПОВЕРНУТИ стан

ВСЕ ЯКЩО

стани = отримати\_нащадків(стан)

оцінки\_станів = массив(стани.розмір)

ПОВТОРИТИ

ДЛЯ ***i*** ВІД 0 ДО стани.розмір

оцінки\_станів[i] = оцінити\_стан(стани[i]) + d

ВСЕ ПОВТОРИТИ

ПОВТОРИТИ

ПОКИ Істина

найкращий = знайти\_мінімальне(оцінки\_станів)

ЯКЩО найкращий > ліміт\_оцінки

ПОВЕРНУТИ невдача, найкращий

ВСЕ ЯКЩО

альтернатива = отримати\_альтернативу(оцінки\_станів)

результат, оцінки\_станів[індекс\_найкращого] =

RBFS\_MAIN(стани[індекс\_найкращого], знайти\_мінімальне(альтернатива,ліміт\_оцінки),

глибина + 1)

ЯКЩО результат

ПОВЕРНТИ результат

ВСЕ ЯКЩО

ВСЕ ПОВТОРИТИ

ФУНКЦІЯ ОЦІНИТИ СТАН

лічильник = 0

ПОВТОРИТИ

ДЛЯ і ВІД 0 ДО кількість ферзів

ПОВТОРИТИ

ДЛЯ j ВІД і

a , b = ферзі[i]

c, d = ферзі[j]

ЯКЩО а==с АБО abs(a - c) == abs(b - d)

лічильник += 1

ВСЕ ЯКЩО

ВСЕ ПОВТОРИТИ

ПОВЕРНУТИ лічильник

ВСЕ ПОВТОРИТИ

ФУНКЦІЯ КОНФЛІКТ

ПОВТОРИТИ

ДЛЯ і від 1 до кількість\_ферзів

ПОВТОРИТИ

ДЛЯ і ВІД 0 ДО і

ЯКЩО а==с АБО abs(a - c) == abs(b - d)

ПОВЕРНУТИ істина

ВСЕ ЯКЩО

ВСЕ ПОВТОРИТИ

ВСЕ ПОВТОРИТИ

ПОВЕРНУТИ хиба

## Програмна реалізація

### Вихідний код

import random

from queue import Queue

import time

import psutil

import os

from math import inf

class Game:

    def \_\_init\_\_(self) -> None:

        self.queens = [

            (0,0),

            (0,1),

            (0,2),

            (0,3),

            (0,4),

            (0,5),

            (0,6),

            (0,7),

        ]

    def shuffle\_queens(self):

        self.queens = []

        for i in range(8):

            self.queens.append((random.randint(0,7),i))

    def showField(self, queens):

        print('-------------------------------------------------')

        for i in range(8):

            print('|  ' , end='')

            for j in range(8):

                if self.queen\_pos(queens,j,i):

                    print(str(1) + '  |  ' , end='')

                else:

                    print(str(0) + '  |  ' , end='')

            print()

            print('-------------------------------------------------')

    def queen\_pos(self,queens, x, y):

        for queen in queens:

            if x == queen[0] and y == queen[1]:

                return True

        return False

#

    def findSolution\_RBFS(self):

        return self.RBFS\_Search(self.queens, inf , 0)[0]

    def RBFS\_Search(self, queens, f\_limit, d):

        if not self.conflict(queens):

            return (queens, 0)

        moves = self.get\_moves(queens)

        f = [0 for i in range(len(moves))]

        for i in range(len(moves)):

            f[i] = self.f2(moves[i]) + d

        while True:

            best = min(f)

            if best > f\_limit:

                return (False, best)

            alternative = self.get\_alternative(f)

            result , f[f.index(best)]= self.RBFS\_Search( moves[ f.index(best) ] , min ( alternative, f\_limit ), d+1 )

            if result:

                return (result, 0)

    def findSolution\_RBFS\_with\_limits(self):

        return self.RBFS\_Search(self.queens, inf , 0, time.time())[0]

    def RBFS\_Search\_with\_limits(self, queens, f\_limit, d, time):

        if time.time() - t0 > 1800 or psutil.Process(os.getpid()).memory\_info().rss > 1024\*\*3:

                print('task too complex')

                return (False, 0)

        if not self.conflict(queens):

            return (queens, 0)

        moves = self.get\_moves(queens)

        f = [0 for i in range(len(moves))]

        for i in range(len(moves)):

            f[i] = self.f2(moves[i]) + d

        while True:

            best = min(f)

            if best > f\_limit:

                return (False, best)

            alternative = self.get\_alternative(f)

            result , f[f.index(best)]= self.RBFS\_Search( moves[ f.index(best) ] , min ( alternative, f\_limit ), d+1 )

            if result:

                return (result, 0)

    def get\_alternative(self, f):

        copy = f.copy()

        copy.pop( f.index( min(f) ) )

        return min(copy)

    def f2(self, queens):

        counter = 0

        for i in range(1, len(queens)):

            for j in range(0, i):

                a, b = queens[i]

                c, d = queens[j]

                if a == c or b == d or abs(a - c) == abs(b - d):

                    counter += 1

        return counter

    def findSolution\_BFS\_with\_limits(self):

        quene = Queue()

        t0 = time.time()

        quene.put(self.queens)

        while not quene.empty():

            if time.time() - t0 > 1800 or psutil.Process(os.getpid()).memory\_info().rss > 1024\*\*3:

                print('task too complex')

                return False

            queens = quene.get()

            if not self.conflict(queens):

                return queens

            for move in self.get\_moves(queens):

                quene.put(move)

    def findSolution\_BFS(self):

        quene = Queue()

        t0 = time.time()

        quene.put(self.queens)

        while not quene.empty():

            if time.time() - t0 > 600:

                return False

            queens = quene.get()

            if not self.conflict(queens):

                return queens

            for move in self.get\_moves(queens):

                quene.put(move)

    def get\_moves(self, queens):

        moves = []

        for i in range(len(queens)):

            for j in range(queens[i][0]):

                initital\_state = queens.copy()

                initital\_state[i] = (j , initital\_state[i][1])

                moves.append(initital\_state)

            for j in range(queens[i][0]+1, 8):

                initital\_state = queens.copy()

                initital\_state[i] = (j , initital\_state[i][1])

                # print(initital\_state)

                moves.append(initital\_state)

        return moves

    def conflict(self, queens):

        for i in range(1, len(queens)):

            for j in range(0, i):

                a, b = queens[i]

                c, d = queens[j]

                if a == c or b == d or abs(a - c) == abs(b - d):

                    return True

        return False

game = Game()

game.shuffle\_queens()

print("BFS Search:")

game.showField(game.queens)

t0 = time.time()

game.findSolution\_BFS()

print('time: ', t0-time.time())

game.showField(game.queens)

print('-----')

game.shuffle\_queens()

print("RBFS Search:")

game.showField(game.queens)

t0 = time.time()

game.findSolution\_BFS()

print('time: ', t0-time.time())

game.showField(game.queens)

### Приклади роботи

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

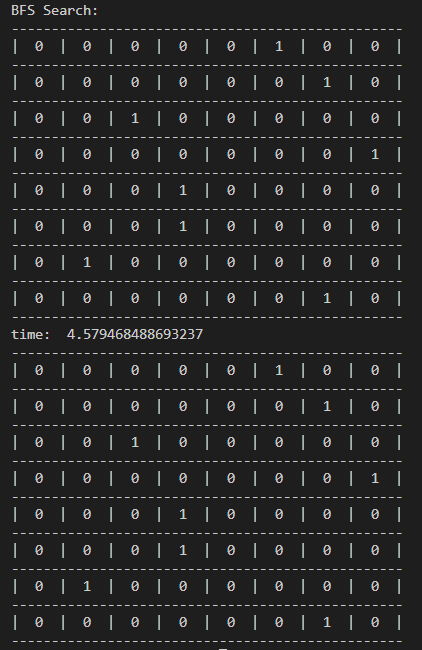
z

Рисунок 3.1 – Алгоритм BFS

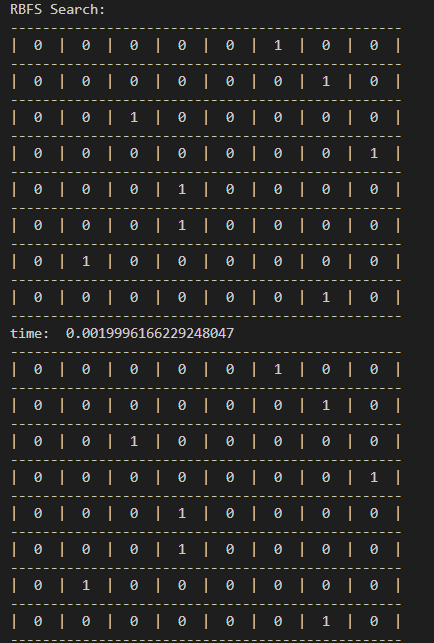


Рисунок 3.2 – Алгоритм RBFS

## Дослідження алгоритмів

В таблиці 3.1 наведені характеристики оцінювання алгоритму Назва алгоритму, задачі Назва задачі для 20 початкових станів.

Таблиця 3.1 – Характеристики оцінювання алгоритму…

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Початкові стани | Ітерації | К-сть гл. кутів | Всього станів | Всього станів у пом’яті |
| Стан 1 | 579 821 | 0 | 32 469 921 | 31 890 100 |
| Стан 2 | 1 670 137 | 0 | 93 527 617 | 91 857 480 |
| Стан 3 | 45 573 | 0 | 2 552 033 | 2 506 460 |
| Стан 4 | 905 167 | 0 | 50 689 297 | 49 784 130 |
| Стан 5 | 393 555 | 0 | 22 039 025 | 21 645 470 |
| Стан 6 | 940 405 | 0 | 52 662 625 | 51 722 220 |
| Стан 7 | 942 132 | 0 | 52 759 337 | 51 817 205 |
| Стан 8 | 745 301 | 0 | 41 736 801 | 40 991 500 |
| Стан 9 | 265 656 | 0 | 14 876 681 | 14 611 025 |
| Стан 10 | 19 703 | 0 | 1 103 313 | 1 083 610 |
| Стан 11 | 29 380 | 0 | 1 645 225 | 1 615 845 |
| Стан 12 | 14 099 | 0 | 789 489 | 775 390 |
| Стан 13 | 17 679 | 0 | 989 969 | 972 290 |
| Стан 14 | 1 112 383 | 0 | 62 293 393 | 61181010 |
| Стан 15 | 986 772 | 0 | 55 259 177 | 54 272 405 |
| Стан 16 | 70 780 | 0 | 3 963 625 | 3 892 845 |
| Стан 17 | 905 910 | 0 | 50 730 905 | 49 824 995 |
| Стан 18 | 731 855 | 0 | 40 983 825 | 40 251 970 |
| Стан 19 | 910 313 | 0 | 50 977 473 | 50 067 160 |
| Стан 20 | 744 903 | 0 | 41 714 513 | 40 969 610 |
| Середнє значення | 601 576 | 0 | 33 688 212 | 33 086 636 |

В таблиці 3.2 наведені характеристики оцінювання алгоритму Назва алгоритму, задачі Назва задачі для 20 початкових станів.

Таблиця 3.3 – Характеристики оцінювання алгоритму…

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Початкові стани | Ітерації | К-сть гл. кутів | Всього станів | Всього станів у пом’яті |
| Стан 1 | 7 | 1 | 392 | 280 |
| Стан 2 | 8 | 2 | 448 | 280 |
| Стан 3 | 16 | 9 | 896 | 336 |
| Стан 4 | 39 | 33 | 2 184 | 280 |
| Стан 5 | 30 | 24 | 1 680 | 280 |
| Стан 6 | 12 | 6 | 672 | 280 |
| Стан 7 | 8 | 3 | 448 | 224 |
| Стан 8 | 91 | 84 | 5 096 | 336 |
| Стан 9 | 6 | 2 | 336 | 168 |
| Стан 10 | 8 | 1 | 448 | 336 |
| Стан 11 | 151 | 144 | 8 456 | 336 |
| Стан 12 | 5 | 0 | 280 | 224 |
| Стан 13 | 6 | 2 | 336 | 168 |
| Стан 14 | 9 | 3 | 5 049 | 280 |
| Стан 15 | 141 | 134 | 7 896 | 336 |
| Стан 16 | 14 | 7 | 784 | 336 |
| Стан 17 | 34 | 28 | 1 904 | 280 |
| Стан 18 | 44 | 38 | 2 464 | 280 |
| Стан 19 | 143 | 137 | 8 008 | 280 |
| Стан 20 | 5 | 0 | 280 | 224 |
| Середнє значення | 38 | 32 | 2 402 | 277 |

Висновок

При виконанні даної лабораторної роботи було розглянуто та досліджено алгоритми неінформативного та інформативного пошуку (BFS та RBFS). Провести порівняльний аналіз ефективності використання алгоритмів. Викоритсано евристичну функцію. Розроблено алгоритм перевірки чи розв`язаний стан задачі.

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%.