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-15 Мєшков Андрій* _____
(шифр, прізвище, ім'я, по батькові)

Перевірив *Головченко М. М.* _____
(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).

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

Варіант 18

Лабіринт, пошук з обмеженням глибини, пошук A* з евристикою евклідова відстань

3.1 Псевдокод алгоритмів

3.1.1 Пошук з обмеженням глибини

```
procedure LDFS(maze, x, y, start, end)
      limit =  x*y//3
      m_copy = m
      dir_map = self.new_map(m)
      priority = ["bottom", "right", "top", "left"]
      path = [start]
      cells = [start]
      movement = [start]
      while not len(movement) > 0 do
            curr= movement.pop()
            if curr == end then
                  break
            end if
            for d in priority do
                  if dir_map[curr][d] == TRUE then
                        if d=='right' then
                              nextcell=(curr[0],curr[1]+1)
                        end if
                        if d=='left' then
                              nextcell=(curr[0],curr[1]-1)
                        end if
                        if d=='top' then
                              nextcell=(curr[0]-1,curr[1])
                        end if
                        if d=='bottom' then
                              nextcell=(curr[0]+1,curr[1])
                        end if
                        cells.append(nextStep)
                        movement.append(nextStep)
                        path.append(nextStep)
                  end if
            end for
      return FALSE
end procedure LDFS
```

3.1.2 A*

```
procedure AStar(maze, x, y, start, end)
      m_copy = m
      dir_map = self.new_map(m)
      priority = ["bottom", "right", "top", "left"]
      q_score = {cell: INFINITY FOR cell IN self.grid(x, y)}
      g_score[start] = 0
      f_score = {cell: INFINITY FOR cell IN self.grid(x, y)}
      f_score[start] = self.h(start, end)
      border = PriorityQueue()
      border.put((self.h(start, end), self.h(start, end), start))
      aPath = {}
      states = []
      while not border.empty() do
            curr = border.get()[2]
            if curr == end then
                  break
            for d in priority do
                  if dir_map[curr][d] == TRUE then
                        if d=='right' then
                              nextcell=(curr[0],curr[1]+1)
                        end if
                        if d=='left' then
                              nextcell=(curr[0],curr[1]-1)
                        end if
                        if d=='top' then
                              nextcell=(curr[0]-1,curr[1])
                        end if
                        if d=='bottom' then
                              nextcell=(curr[0]+1,curr[1])
                        end if
                        tent_g_score = g_score[curr] + 1
                        tent_f_score = tent_g_score + self.h(curr, end)
                        if tent_f_score < f_score[nextcell] then
                              g_score[nextcell] = tent_g_score
                              f_score[nextcell] = tent_f_score
                              open.put((tent_f_score, nextcell))
                              came_from[nextcell] = curr
                        end if
                  end if
            end for
      end while
      return FALSE
```

```
        end procedure AStar(maze)


        procedure h(a, b)
              x1,y1 = a
              x2,y2 = b
              return SQRT((x1 – x2)**2 + (y1 – y2)**2)
        end procedure h
```

Програмна реалізація

3.1.3 Вихідний код

```
class LDFS:
   def __init__(self):
      self.iterat = 0
      self.state = 1
      self.mem_st = 0
      self.stop = 0

   def search(self, m, x, y, start, end):
      self.iterat = 0
      self.mem_st = 0
      self.stop = 0
      self.state = 0
      limit =  x*y//3
      m_copy = m
      dir_map = self.new_map(m)
      priority = ["bottom", "right", "top", "left"]
      path = [start]
      cells = [start]
      movement = [start]
      memory = [start]
      while len(movement) > 0:
         self.iterat+=1
         curr = movement.pop()
         if curr == end:
            break
         if len(movement) > self.mem_st:
            self.mem_st = len(movement)
         for direction in priority:
            if dir_map[curr][direction]:
               if direction == "left":
                  nextStep = (curr[0], curr[1] - 1)
               elif direction == "top":
                  nextStep = (curr[0] - 1, curr[1])
               elif direction == "right":
                  nextStep = (curr[0], curr[1] + 1)
               else:
                  nextStep = (curr[0] + 1, curr[1])
               if nextStep in cells:
                  memory.append(nextStep)
                  continue
               cells.append(nextStep)
```

```
                movement.append(nextStep)
                path.append(nextStep)
                memory.append(nextStep)
                if len(path) - 1 == limit:
                    self.stop+=1
                if self.iterat > 1 and m_copy[curr[0]][curr[1]] != 's':
                    m_copy[curr[0]][curr[1]] = 'P'
        self.state += printPath(path)
        self.mem_st += len(movement)
        printMaze(m_copy, x, y)



    def new_map(self, m):
        dir_map = {}
        for i in range(len(m)):
            for j in range(len(m[i])):
                x = i
                y = j
                curr = (x, y)
                dir_map[curr] = {"left": self.is_dir(m, i, j-1),
                                 "top": self.is_dir(m, i-1, j),
                                 "right": self.is_dir(m, i, j+1),
                                 "bottom": self.is_dir(m, i+1, j)}
        return dir_map

    def is_dir(self, m, x, y):
        if x>=len(m) or x<0 or y>=len(m[x]) or y<0:
            return False
        return True if m[x][y] == "c" or m[x][y] == "E" else False


class AStar:
    def __init__(self):
        self.iterat = 0
        self.state = 0
        self.mem_st = 0

    def grid(self, rows, cols):
        grid=[]
        maze_map={}
        y=0
        for n in range(cols):
            x = 1
            y = 1+y
            for m in range(rows):
                grid.append((x,y))
                maze_map[x,y]={'right':0,'left':0,'top':0,'bottom':0}
                x = x + 1
        return grid

    def h(self, cell1, cell2):
        x1,y1=cell1
        x2,y2=cell2
        return math.sqrt((x1-x2)**2 + (y1-y2)**2)

    def search(self, m, x, y, start, end):
        self.iterat = 0
        self.state = 0
        self.mem_st = 0
```

```
    m_copy = m
    dir_map = self.new_map(m)
    priority = ["bottom", "right", "top", "left"]
    g_score = {cell:float('inf') for cell in self.grid(x, y)}
    g_score[start]=0
    f_score={cell:float('inf') for cell in self.grid(x, y)}
    f_score[start] = self.h(start, end)
    border=PriorityQueue()
    border.put((self.h(start, end), self.h(start, end), start))
    aPath = {}
    states = []
    while not border.empty():
        self.iterat += 1
        if border.qsize() > self.mem_st:
            self.mem_st = border.qsize()
        curr = border.get()[2]
        if curr not in states:
            states.append(curr)
        if curr == end:
            self.state = len(states)
            break
        for d in priority:
            if dir_map[curr][d]==True:
                if d=='right':
                    nextcell=(curr[0],curr[1]+1)
                if d=='left':
                    nextcell=(curr[0],curr[1]-1)
                if d=='top':
                    nextcell=(curr[0]-1,curr[1])
                if d=='bottom':
                    nextcell=(curr[0]+1,curr[1])
                temp_g_score=g_score[curr]+1
                temp_f_score=temp_g_score+self.h(curr,end)
                if temp_f_score < f_score[nextcell]:
                    g_score[nextcell]= temp_g_score
                    f_score[nextcell]= temp_f_score
                    border.put((temp_f_score,self.h(nextcell,end),nextcell))
                    aPath[nextcell]=curr
    fwdPath={}
    cell=end
    while cell!=start:
        fwdPath[aPath[cell]]=cell
        cell=aPath[cell]
    printPath(fwdPath)
    for i in fwdPath:
        if m_copy[i[0]][i[1]] != 'S' and m_copy[i[0]][i[1]] != 'E':
            m_copy[i[0]][i[1]] = 'P'
    printMaze(m_copy, x, y)

def new_map(self, m):
    dir_map = {}
    for i in range(len(m)):
        for j in range(len(m[i])):
            x = i
            y = j
            curr = (x, y)
            dir_map[curr] = {"left": self.is_dir(m, i, j-1),
                             "top": self.is_dir(m, i-1, j),
                             "right": self.is_dir(m, i, j+1),
```

```
                    "bottom": self.is_dir(m, i+1, j)}
        return dir_map

    def is_dir(self, m, x, y):
        if x>=len(m) or x<0 or y>=len(m[x]) or y<0:
            return False
        return True if m[x][y] == "c" or m[x][y] == "E" else False
```

3.1.4 Приклади роботи

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

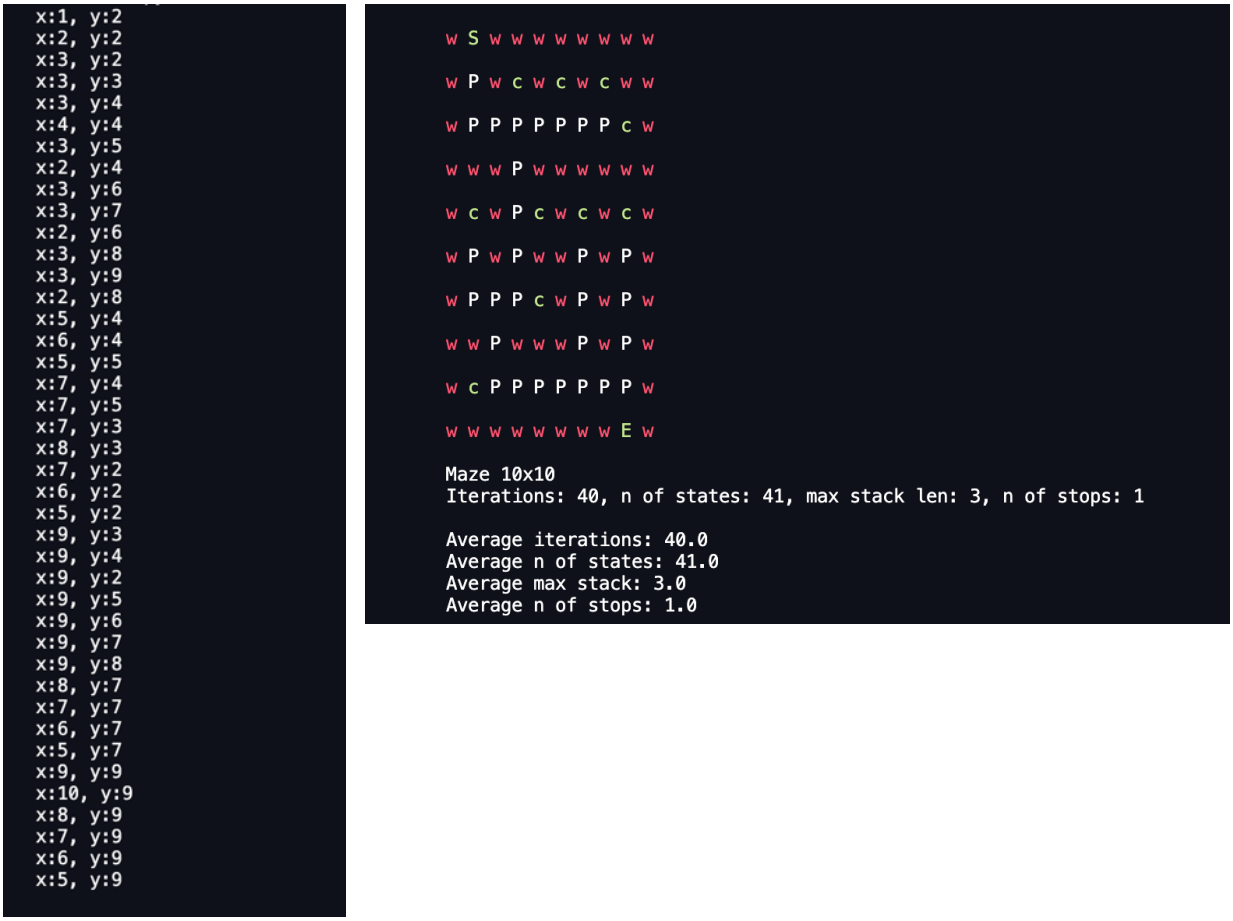

Рисунок 3.1 – Алгоритм пошуку з обмеженням глибини

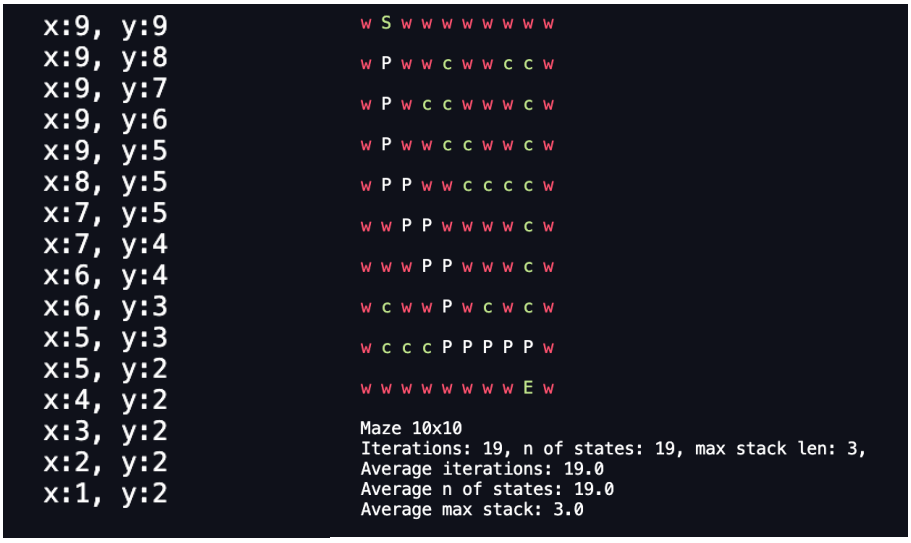

Рисунок 3.2 – Алгоритм пошуку A*

3.2 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	К-ть глухих кутів
Стан 1	149	152	8	1
Стан 2	168	169	11	1
Стан 3	122	126	12	0
Стан 4	88	92	10	0
Стан 5	132	138	16	1
Стан 6	98	101	7	0
Стан 7	104	109	12	0
Стан 8	101	105	9	0
Стан 9	162	163	8	1
Стан 10	115	118	10	1
Стан 11	166	168	9	1
Стан 12	147	149	11	1

Стан 13	132	140	16	0
Стан 14	143	145	10	0
Стан 15	115	118	9	1
Стан 16	159	161	10	1
Стан 17	167	167	8	0
Стан 18	125	129	11	0
Стан 19	149	155	19	1
Стан 20	130	134	10	1
СЕРЕДНЄ	134	137	11	1

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

Таблиця 3.3 – Характеристики оцінювання пошуку A*

Початкові стани	Ітерації	Всього станів	Всього станів у пом’яті
Стан 1	73	73	11
Стан 2	59	59	5
Стан 3	79	79	9
Стан 4	98	98	12
Стан 5	111	111	15
Стан 6	104	104	13
Стан 7	172	172	11
Стан 8	82	82	11
Стан 9	69	69	8
Стан 10	80	80	7
Стан 11	163	163	11
Стан 12	85	85	13
Стан 13	66	66	9
Стан 14	96	96	9

Стан 15	121	121	8
Стан 16	70	70	11
Стан 17	113	113	13
Стан 18	74	74	12
Стан 19	69	69	10
Стан 20	87	87	10
СЕРЕДНЄ	94	94	10

ВИСНОВОК

При виконанні даної лабораторної роботи було розглянуто алгоритми пошуку в глибину з обмеженням глибини та A* 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, також кількість глухих кутів для першого алгоритму.

Зроблено висновок, що пошук з обмеженням глибини є неповним і неоптимальним алгоритмом, коли пошук A* є повним і є оптимальним допоки еврестична функція є прийнятною. Було отримано, що пошук A* здійснює менше ітерацій, зберігає можливо менше станів у пам’яті, але розгортає можливо більше станів.

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%.