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-35 Адаменко Арсен Богданович _____
(шифр, прізвище, ім'я, по батькові)

Перевірив Головченко М.М. _____
(прізвище, ім'я, по батькові)

Київ 2024

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.
- **F3** – кількість ферзів, які стоять не на свої місцях, в порівнянні з одним з еталонних розв’язків.
- **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	COLOR			HILL	MRV
2	COLOR			ANNEAL	MRV
3	COLOR			BEAM	MRV
4	COLOR			HILL	DGR
5	COLOR			ANNEAL	DGR
6	COLOR			BEAM	DGR
7	8-ферзів	LDFS	A*		F1
8	8-ферзів	LDFS	A*		F2
9	8-ферзів	LDFS	RBFS		F3
10	8-ферзів	LDFS	RBFS		F1

№	Задача	АНП	АІП	АЛП	Func
11	8-ферзів	BFS	A*		F2
12	8-ферзів	BFS	A*		F3
13	8-ферзів	BFS	RBFS		F1
14	8-ферзів	BFS	RBFS		F2
15	8-ферзів	IDS	A*		F3
16	8-ферзів	IDS	A*		F1
17	8-ферзів	IDS	RBFS		F2
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

№	Задача	АНП	АІП	АЛП	Func
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	Лабіринт	LDFS	A*		H2
32	Лабіринт	LDFS	RBFS		H3
33	Лабіринт	BFS	A*		H2
34	Лабіринт	BFS	RBFS		H3
35	Лабіринт	IDS	A*		H2
36	Лабіринт	IDS	RBFS		H3

В цій лабораторній роботі треба реалізувати наступні алгоритми пошуку в графі:

1	COLOR			HILL	MRV

3 ВИКОНАННЯ

1.1 Псевдокод алгоритмів

3.1.1 Псевдокод Backtracking з використанням евристики MRV

```
func backtracking_impl(G, C, c)
    if no nil colors left
        return true

    bV, bR = mrv(G, C, c)

    for i, v in bv
        if bR[i] == 0
            continue

        for c_ in c
            if not correct color
                continue

                C[v] = c_

                if backtracking_impl(G, C, c)
                    return true

                C[v] = nil


    return false

func backtracking(G, c)
    C = [nil * c]

    if backtracking(G, C, c)
        return C
    else
        return nil
```

3.1.2 Псевдокод Hill Climbing з евристикою по кількості конфліктів кольорів з можливістю руху вбік на обмежену кількість кроків та з можливістю перезапустити алгоритм:

```
def hill_climb(G, c, r, w)
    bC = None
    mc = float('inf')

    for _ in range(r)
        C = rgc(G, c)

        blC = C.copy()
        mlc = gc(G, blC)

        wc = 0
        loop forever
            impr = false

            bV = mrvs(G, C, c)

            for bv in bV
                bc = C[bv]
```

```
                for c_ = 0, c - 1
                    if not (c_ != bc)
                        continue

                    dcC = copy(C)
                    dcC[bv] = c_
                    dcc = gc(G, dcC)

                    if dcc < mlc:
                        blC = dcC
                        mlc = dcc
                        impr = true

                    if mlc == 0
                        break

            if not (wc < w)
                break

            if not impr
                C[rndv(G)] = rndc(c)
                wc++

        if mlc < mc
            bC = blC
            mc = mlc

        if mc == 0
            break

    return bC, mc
```

1.2 Програмна реалізація

1.2.1 Вихідний код

Алгоритм Hill Climbing розфарбування графу з евристикою по кількості конфліктів кольорів з можливістю руху вбік на обмежену кількість кроків та з можливістю перезапустити алгоритм:

```
def hill_climb(G, c, r, w):
    bC = None
    mc = float('inf')

    for _ in range(r):
        blC = rgc(G, c)
        mlc = gc(G, blC)

        wc = 0
        impr = True

        while impr:
            impr = False

            bV = mxcnfvs(G, blC)

            for bv in bV:
                if not (wc < w):
                    break
```

```
                bc = blC[bv]

                for c_ in range(c):
                    if not (c_ != bc):
                        continue

                    dcC = blC.copy()
                    dcC[bv] = c_
                    dcc = gc(G, dcC)

                    if dcc < mlc:
                        blC = dcC
                        mlc = dcc

                        impr = True

                    if mlc == 0:
                        break

                wc += 1

        if mlc < mc:
            bC  = blC
            mc = mlc

        if mc == 0:
            break

    return bC, mc
```

Алгоритм Backtracking з евристикою MRV:

```
def rv(G, C, c, v):
    c = set(range(c))

    for nb in G[v]:
        c -= {C[nb]}

    return len(c)

def mrvsci(G, C, c):
    R = []
    V = []
    m = float('inf')

    for v in G:
        if not (C[v] == None):
            continue

        crv = rv(G, C, c, v)

        if crv < m:
            m = crv

            R = []
            V = []

        if crv <= m:
            R += [crv]
            V += [v]

    return V, R
```

```
def vs(G, C, v, c):
    for neighbor in G[v]:
        if C[neighbor] == c:
            return False

    return True

def backtracking_impl(G, C, c):
    if not (len([None for v in C if C[v] == None]) >= 1):
        return True

    bV, bR = mrvsci(G, C, c)

    for i, bv in enumerate(bV):
        if not (bR[i] >= 1):
            continue

        for c_ in range(c):
            if not vs(G, C, bv, c_):
                continue

            C[bv] = c_

            if backtracking_impl(G, C, c):
                return True

            C[bv] = None

    return False

def backtracking(G, c):
    C = {v: None for v in G}

    if backtracking_impl(G, C, c):
        return C
    else:
        return None
```

1.2.2 Приклади роботи

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

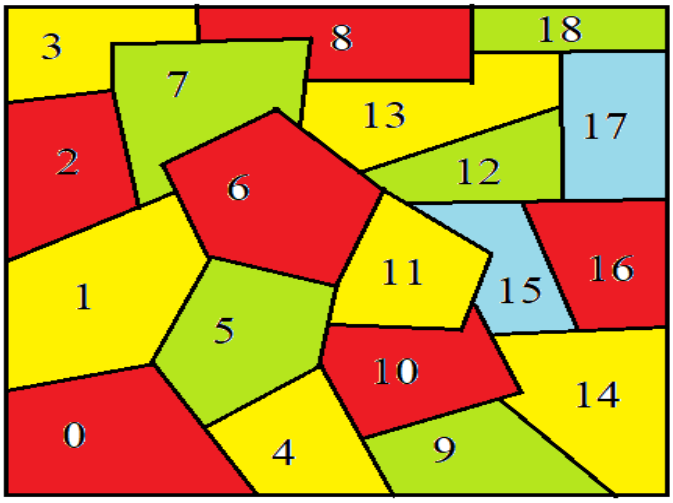

Рисунок 3.1 – Алгоритм Backtracking

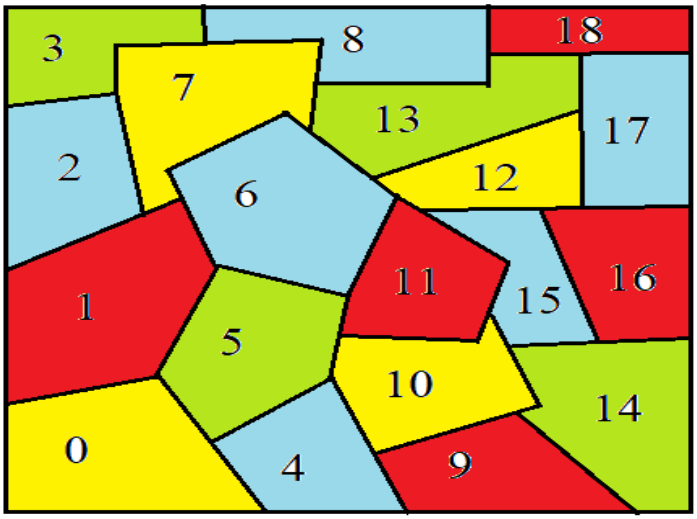

Рисунок 3.2 – Алгоритм Hill Climbing при 4 п. та 20 р/в

1.3 Дослідження алгоритмів

В таблиці 3.1 наведені характеристики оцінювання алгоритму задачі розфарбування графу для 20 початкових станів.

Таблиця 3.1 – Характеристики оцінювання Backtracking

Початкові стани	Ітерації	К-сть гл. кутів	Всього вузлів	Всього вузлів у пам’яті
0	44	0	1	1
1	46	0	1	1
2	44	0	1	1
3	44	0	1	1
4	45	0	1	1
5	48	0	1	1
6	50	0	1	1
7	48	0	1	1
8	40	0	1	1
9	45	0	1	1
10	50	0	1	1
11	52	0	1	1
12	56	0	1	1
13	56	0	1	1
14	46	0	1	1
15	65	0	1	1
16	40	0	1	1
17	214	99	1	1
18	49	0	1	1

В таблиці 3.2 наведені характеристики оцінювання алгоритму Backtracking, задачі розфарбування графу для 20 початкових станів.

Таблиця 3.3 – Характеристики оцінювання Hill Climbing

Початкові стани	Ітерації	К-сть гл. кутів	Всього вузлів	Всього вузлів у пам’яті
1 3 0 2 0 3 3 2 0 2 3 0 3 3 0 2 2 3 3	484	69	1	1
2 1 2 2 1 2 2 0 3 1 0 3 1 2 0 3 2 3 3	116	9	1	1
3 1 2 1 1 2 0 3 3 1 3 2 1 2 1 3 2 2 2	228	40	1	1
0 0 1 3 3 1 0 2 1 1 3 0 1 3 3 1 3 0 1	57	22	1	1
2 0 3 3 0 3 1 2 3 3 3 0 3 3 0 1 2 1 3	48	19	1	1
1 3 3 0 3 3 0 1 1 3 3 3 0 2 1 3 2 1 1	298	39	1	1
0 0 3 3 0 3 2 0 3 1 0 2 0 1 1 0 0 0 1	130	11	1	1
3 0 0 1 0 1 3 0 0 1 3 2 0 0 3 0 1 2 3	644	92	1	1
1 2 3 2 0 0 3 3 1 0 0 1 0 1 0 3 0	52	20	1	1

Початкові стани	Ітерації	К-сть гл. кутів	Всього вузлів	Всього вузлів у пам’яті
3 2				
3 0 2 2 0 0 2 2 1 3 0 1 1 2 0 2 0 2 0	76	21	1	1
0 2 0 3 2 0 1 1 2 1 3 2 0 1 0 1 2 2 1	358	50	1	1
2 1 0 3 3 2 3 0 0 1 1 2 3 1 3 0 1 2 1	90	12	1	1
0 2 1 2 3 2 3 0 1 2 0 3 2 1 2 1 2 0 0	198	33	1	1
3 0 1 3 0 3 1 1 1 0 1 0 2 0 1 3 2 1 1	117	26	1	1
3 2 0 2 2 1 1 1 0 1 1 2 0 1 0 1 0 0 2	111	29	1	1
1 1 0 1 2 1 0 0 0 3 2 2 0 3 2 3 3 0 3	39	3	1	1
1 1 0 1 2 2 2 1 3 1 2 2 3 0 0 1 0 1 0	279	50	1	1
3 2 3 2 1 1 3 2 1 3 0 0 3 1 2 2 2	145	20	1	1

Початкові стани	Ітерації	К-сть гл. кутів	Всього вузлів	Всього вузлів у пам’яті
0 3				
2 0 3 3 3 3 1 1 2 3 1 0 0 2 3 0 1 0 2	134	24	1	1
1 3 1 0 1 2 1 1 2 0 1 3 2 1 3 1 3 0 1	57	20	1	1

Час виконання алгоритмів на даних різного розміру зображено нижче в таблиці 3.4.

Таблиця 3.4 — практична часова складність виконання алгоритмів.

	Backtracking	Hill Climbing		
Розмір графу	Час виконання, с.	Час виконання, с.	Налаштування	
			Перезапуски, к-сть	Рухи вбік, к-сть
3	0.00006	0.00004	10	100
9	0.00021	0.00013	10	100
20	0.00056	0.00495	10	100

ВИСНОВОК

При виконанні даної лабораторної роботи було розглянуто такі алгоритми, як Backtracking з використанням евристики MRV (“Найменше можливе число кольорів для розмалювання”) та Hill Climbing з використанням евристики кількості конфліктів. Максимальна кількість рестартів для Hill Climbing встановлено на рівні 10, а максимальна кількість рухів убік — 100 ітерацій.

Під час дослідів було встановлено, що алгоритм Backtracking працює набагато швидше аніж Hill Climbing на даних середнього розміру (від 20) у десятки разів швидше. На даних меншого розміру алгоритм Hill Climbing є достатньо ефективним. Алгоритми споживають схожу кількість оперативної пам’яті комп’ютера.

Алгоритм Backtracking є більш ефективними ніж Hill Climbing під час виконання цієї лабораторної роботи.

КРИТЕРІЇ ОЦІНЮВАННЯ

Максимальний бал за лабораторну роботу №2 дорівнює – 5.

Критерії оцінювання у відсотках від максимального балу:

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