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 № 5 з дисципліни
«Проектування алгоритмів»

„Проектування і аналіз алгоритмів для вирішення NP-складних задач ч.2”

Виконав(ла) *ІП-13 Вдовиченко С.Ю.* _____
(шифр, прізвище, ім'я, по батькові)

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

Київ 2022

ЗМІСТ

1 МЕТА ЛАБОРАТОРНОЇ РОБОТИ

Мета роботи – вивчити основні підходи розробки метаевристичних алгоритмів для типових прикладних задач. Опрацювати методологію підбору прийнятних параметрів алгоритму.

2 ЗАВДАННЯ

Згідно варіанту, формалізувати алгоритм вирішення задачі відповідно загальної методології.

Записати розроблений алгоритм у покроковому вигляді. З достатнім степенем деталізації.

Виконати його програмну реалізацію на будь-якій мові програмування.

Перелік задач наведено у таблиці 2.1.

Перелік алгоритмів і досліджуваних параметрів у таблиці 2.2.

Задача і алгоритм наведені в таблиці 2.3.

Змінюючи параметри алгоритму, визначити кращі вхідні параметри алгоритму. Для цього необхідно:

– обрати критерій зупинки алгоритму (кількість ітерацій або значення ЦФ);

– зафіксувати усі параметри крім одного і змінювати цей параметр, поки не буде досягнуто пікової ефективності;

– після цього параметр фіксується і змінюються інші параметри;

– далі повторюємо процедуру спочатку, з першого зафіксованого параметру;

– зупиняємось коли будуть знайдені оптимальні параметри для даної задачі або встановлена залежність одних параметрів від інших.

Зробити узагальнений висновок в якому обов’язково описати залежність якості розв’язку від вхідних параметрів.

Таблиця 2.1 – Прикладні задачі

№	**Задача**
2	**Задача комівояжера** (300 вершин, відстань між вершинами випадкова від 5 до 150) полягає у знаходженні найвигіднішого маршруту, що проходить через вказані міста хоча б по одному разу. В умовах завдання вказуються критерій вигідності маршруту (найкоротший, найдешевший, сукупний критерій тощо) і відповідні матриці

	відстаней, вартості тощо. Зазвичай задано, що маршрут повинен проходити через кожне місто тільки один раз, в такому випадку розв'язок знаходиться серед гамільтонових циклів. **Розглядається симетричний, асиметричний та змішаний варіанти.** В загальному випадку, асиметрична задача комівояжера відрізняється тим, що ребра між вершинами можуть мати різну вагу в залежності від напряму, тобто, задача моделюється орієнтованим графом. Таким чином, окрім ваги ребер графа, слід також зважати і на те, в якому напрямку знаходяться ребра. У випадку симетричної задачі всі пари ребер між одними й тими самими вершинами мають однакову вагу. У випадку реальних міст може бути як симетричною, так і асиметричною в залежності від тривалості або довжини маршрутів і напряму руху. Застосування: – доставка товарів (в цьому випадку може бути більш доречна постановка транспортної задачі - доставка в кілька магазинів з декількох складів); – доставка води; – моніторинг об'єктів; – поповнення банкоматів готівкою; – збір співробітників для доставки вахтовим методом.

Таблиця 2.2 – Варіанти алгоритмів і досліджувані параметри

№	**Алгоритми і досліджувані параметри**
1	**Генетичний алгоритм:** - оператор схрещування (мінімум 3); - мутація (мінімум 2); - оператор локального покращення (мінімум 2).

Таблиця 2.3 – Варіанти задач і алгоритмів

№	Задачі і алгоритми
1	Задача про рюкзак + Генетичний алгоритм
2	Задача про рюкзак + Бджолиний алгоритм
3	Задача комівояжера (асиметрична мережа) + Генетичний алгоритм
4	Задача комівояжера (симетрична мережа) + Генетичний алгоритм
5	Задача комівояжера (змішана мережа) + Генетичний алгоритм
6	Задача комівояжера (асиметрична мережа) + Мурашиний алгоритм
7	Задача комівояжера (симетрична мережа) + Мурашиний алгоритм
8	Задача комівояжера (змішана мережа) + Мурашиний алгоритм
9	Задача вершинного покриття + Генетичний алгоритм
10	Задача вершинного покриття + Бджолиний алгоритм
11	Задача комівояжера (асиметрична мережа) + Бджолиний алгоритм
12	Задача комівояжера (симетрична мережа) + Бджолиний алгоритм
13	Задача комівояжера (змішана мережа) + Бджолиний алгоритм
14	Розфарбовування графа + Генетичний алгоритм
15	Розфарбовування графа + Бджолиний алгоритм
16	Задача про кліку (задача розпізнавання) + Генетичний алгоритм
17	Задача про кліку (задача розпізнавання) + Бджолиний алгоритм
18	Задача про кліку (обчислювальна задача) + Генетичний алгоритм
19	Задача про кліку (обчислювальна задача) + Бджолиний алгоритм
20	Задача про найкоротший шлях + Генетичний алгоритм
21	Задача про найкоротший шлях + Мурашиний алгоритм
22	Задача про найкоротший шлях + Бджолиний алгоритм
23	Задача про рюкзак + Генетичний алгоритм
24	Задача про рюкзак + Бджолиний алгоритм
25	Задача комівояжера (асиметрична мережа) + Генетичний алгоритм

26	Задача комівояжера (симетрична мережа) + Генетичний алгоритм
27	Задача комівояжера (змішана мережа) + Генетичний алгоритм
28	Задача комівояжера (асиметрична мережа) + Мурашиний алгоритм
29	Задача комівояжера (симетрична мережа) + Мурашиний алгоритм
30	Задача комівояжера (змішана мережа) + Мурашиний алгоритм

3 ВИКОНАННЯ

3.1 Покроковий алгоритм

- Створюємо клас City для міст та розрахунку відстані.
- Створюємо клас Fitness.
- Створюємо початкову популяцію.
- Використовуємо Fitness для сортування кожної особи, створюємо список з ідентифікаторами маршрутів та вартістю.
- Обираємо батьків методом **Fitness proportionate selection.**
- Використовуємо оператор схрещування.
- Застововуємо мутацію.
- Застосовуємо генетичний алгоритм для визначеної кількості ітерацій: створюємо нову популяцію, обираємо найкращу.

3.2 Програмна реалізація алгоритму

3.2.1 Вихідний код

```
import numpy as np, random, operator, pandas as pd

class City:
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def distance(self, city):
        x_distance = abs(self.x - city.x)
        y_distance = abs(self.y - city.y)
        distance = np.sqrt((x_distance ** 2) + (y_distance ** 2))
        return distance

    def __repr__(self):
        return "(" + str(self.x) + "," + str(self.y) + ")"

class Fitness:
    def __init__(self, route):
        self.route = route
        self.distance = 0
        self.fitness = 0.0

    def route_distance(self):
        if self.distance == 0:
            path_distance = 0
            for i in range(0, len(self.route)):
                from_city = self.route[i]
                to_city = None
                if i + 1 < len(self.route):
```

```
                    to_city = self.route[i + 1]
                else:
                    to_city = self.route[0]
                path_distance += from_city.distance(to_city)
            self.distance = path_distance
        return self.distance

    def route_fitness(self):
        if self.fitness == 0:
            self.fitness = 1 / float(self.route_distance())
        return self.fitness


def create_route(city_list):
    route = random.sample(city_list, len(city_list))
    return route


def initial_population(pop_size, city_list):
    population = []

    for i in range(0, pop_size):
        population.append(create_route(city_list))
    return population


def rank_routes(population):
    fitness_results = {}
    for i in range(0,len(population)):
        fitness_results[i] = Fitness(population[i]).route_fitness()
    return sorted(fitness_results.items(), key = operator.itemgetter(1), reverse
= True)


def selection(pop_ranked, elite_size):
    selection_results = []
    df = pd.DataFrame(np.array(pop_ranked), columns=["Index", "Fitness"])
    df['cum_sum'] = df.Fitness.cumsum()
    df['cum_perc'] = 100 * df.cum_sum / df.Fitness.sum()

    for i in range(0, elite_size):
        selection_results.append(pop_ranked[i][0])
    for i in range(0, len(pop_ranked) - elite_size):
        pick = 100 * random.random()
        for i in range(0, len(pop_ranked)):
            if pick <= df.iat[i, 3]:
                selection_results.append(pop_ranked[i][0])
                break
    return selection_results


def mating_pool(population, selection_results):
    matingpool = []
    for i in range(0, len(selection_results)):
        index = selection_results[i]
        matingpool.append(population[index])
    return matingpool


def breed(parent1, parent2):
    child = []
    child_p1 = []
    child_p2 = []
```

```
    gene_a = int(random.random() * len(parent1))
    gene_b = int(random.random() * len(parent1))

    start_gene = min(gene_a, gene_b)
    end_gene = max(gene_a, gene_b)

    for i in range(start_gene, end_gene):
        child_p1.append(parent1[i])

    child_p2 = [item for item in parent2 if item not in child_p1]

    child = child_p1 + child_p2
    return child


def breed_population(mating_pool, elite_size):
    children = []
    length = len(mating_pool) - elite_size
    pool = random.sample(mating_pool, len(mating_pool))

    for i in range(0, elite_size):
        children.append(mating_pool[i])

    for i in range(0, length):
        child = breed(pool[i], pool[len(mating_pool) - i - 1])
        children.append(child)
    return children


def mutate(individual, mutation_rate):
    for swapped in range(len(individual)):
        if (random.random() < mutation_rate):
            swap_with = int(random.random() * len(individual))

            city1 = individual[swapped]
            city2 = individual[swap_with]

            individual[swapped] = city2
            individual[swap_with] = city1
    return individual


def mutatePopulation(population, mutation_rate):
    mutated_pop = []

    for ind in range(0, len(population)):
        mutated_ind = mutate(population[ind], mutation_rate)
        mutated_pop.append(mutated_ind)
    return mutated_pop


def next_generation(current_gen, elite_size, mutation_rate):
    pop_ranked = rank_routes(current_gen)
    selection_results = selection(pop_ranked, elite_size)
    matingpool = mating_pool(current_gen, selection_results)
    children = breed_population(matingpool, elite_size)
    next_generation = mutatePopulation(children, mutation_rate)
    return next_generation


def genetic_algorithm(population, pop_size, elite_size, mutation_rate,
generations):
    pop = initial_population(pop_size, population)
    print(f"Initial distance: {str(int(1 / rank_routes(pop)[0][1]))}")
```

```
    for i in range(0, generations):
        pop = next_generation(pop, elite_size, mutation_rate)

    print(f"Final distance: {str(int(1 / rank_routes(pop)[0][1]))}")
    best_route_index = rank_routes(pop)[0][0]
    best_route = pop[best_route_index]
    print(f"Number of iterations: {generations}")
    # print(best_route)
    return best_route


if __name__ == "__main__":
    city_list = []

    for i in range(0, 300):
        city_list.append(City(x=int(random.random() * 5 + random.random() * 150
- 1),
                              y=int(random.random() * 5 + random.random() * 150
- 1)))
    genetic_algorithm(population=city_list, pop_size=100, elite_size=60,
mutation_rate=0.3, generations=100)
```

3.2.2 Приклади роботи

На рисунках 3.1 і 3.2 показані приклади роботи програми.

```
E:\ads_lab5\venv\Scripts\python.exe E:/ads_lab5/main.py
Initial distance: 22745
Final distance: 19637
Number of iterations: 100
[(148,24), (80,26), (138,97), (98,135), (114,126), (114,77), (46,10), (57,12), (52,5), (80,20), (67,149), (58,13), (12,4), (81,150), (86,138), (34,149), (37,109), (86,94), (14,83), (4,65),
 (69,39), (59,92), (41,132), (19,123), (104,135), (31,47), (88,142), (90,143), (102,148), (45,25), (89,132), (57,135), (123,148), (84,142), (62,13), (116,38), (151,70), (135,24), (131,15),
 (33,56), (43,33), (42,59), (27,67), (41,79), (10,108), (30,93), (6,149), (16,75), (2,108), (17,129), (7,142), (15,148), (21,42), (26,5), (60,52), (82,80), (50,91), (74,45), (76,64), (52,17),
 (39,38), (125,28), (76,69), (4,47), (143,44), (81,27), (96,87), (57,12), (103,103), (94,109), (42,132), (61,112), (146,143), (132,107), (148,108), (121,113), (102,82), (21,148), (110,11),
 (129,36), (136,137), (75,146), (136,74), (122,94), (123,94), (136,3), (20,18), (96,35), (103,43), (73,43), (66,92), (36,54), (105,65), (107,122), (133,18), (58,23), (83,11), (105,89), (146,124),
 (146,31), (102,18), (119,46), (114,88), (59,129), (41,42), (86,120), (113,107), (129,129), (66,2), (60,80), (126,50), (51,148), (110,128), (117,105), (74,111), (30,32), (44,62), (31,83), (10,67),
 (54,128), (7,49), (129,30), (145,120), (123,66), (136,145), (66,116), (33,146), (37,80), (7,100), (131,108), (66,137), (10,103), (10,71), (120,18), (19,24), (101,34), (18,18), (62,141),
 (26,152), (31,52), (146,134), (149,59), (89,63), (38,132), (127,89), (129,32), (85,70), (75,86), (129,81), (147,56), (108,59), (84,20), (119,32), (30,51), (56,125), (142,87), (52,28), (23,35),
 (11,129), (90,77), (83,94), (109,97), (150,26), (100,93), (44,38), (31,53), (21,34), (9,26), (42,140), (82,149), (105,117), (118,74), (114,19), (62,144), (139,93), (56,127), (52,115), (15,117),
 (47,65), (3,53), (97,38), (92,43), (8,18), (72,14), (131,56), (70,72), (76,148), (29,70), (33,76), (24,7), (124,21), (44,113), (65,143), (76,134), (1,85), (21,119), (76,119), (132,147), (99,135),
 (113,46), (90,98), (74,26), (7,64), (93,42), (70,83), (28,53), (21,17), (3,115), (103,113), (135,78), (119,120), (139,124), (133,35), (74,30), (142,48), (96,137), (100,102), (148,125), (56,128),
 (149,3), (121,49), (102,53), (129,131), (75,116), (16,32), (84,42), (138,125), (81,84), (98,3), (4,107), (145,100), (46,142), (86,112), (108,67), (37,55), (141,32), (141,74), (116,142), (92,80),
 (57,18), (18,80), (47,28), (128,12), (115,80), (112,89), (132,120), (138,28), (111,30), (4,91), (101,89), (152,137), (102,115), (109,110), (38,101), (96,109), (61,18), (12,94), (27,122),
(134,35), (14,141), (15,148), (19,149), (93,112), (141,140), (94,91), (56,41), (59,97), (109,30), (113,14), (12,73), (36,128), (78,20), (97,116), (139,70), (10,89), (135,148), (122,145), (83,78),
 (59,117), (35,76), (36,146), (144,12), (68,95), (129,126), (134,122), (81,87), (19,110), (125,32), (34,57), (57,3), (73,88), (85,22), (84,52), (71,61), (55,52), (138,28), (141,115), (55,73),
 (79,11), (99,21)]

Process finished with exit code 0
```

Рисунок 3.1 – Робота алгоритму при 100 ітераціях

```
E:\ads_lab5\venv\Scripts\python.exe E:/ads_lab5/main.py
Initial distance: 22166
Final distance: 19058
Number of iterations: 1000
[(87,83), (81,114), (31,8), (35,7), (51,40), (98,33), (46,9), (38,82), (62,95), (81,44), (61,92), (16,79), (89,18), (57,70), (127,108), (116,149), (17,148), (11,134), (7,36), (129,70), (121,141),
 (84,112), (31,44), (71,66), (110,77), (117,124), (5,116), (19,112), (23,149), (101,140), (52,132), (122,0), (124,9), (34,32), (100,99), (51,130), (85,144), (97,53), (4,129), (27,101), (40,77),
 (14,76), (43,61), (70,59), (62,20), (1,151), (12,140), (131,33), (119,58), (81,19), (85,40), (55,51), (49,119), (38,70), (65,9), (99,3), (134,103), (24,133), (2,41), (88,27), (106,0), (148,126),
 (121,146), (93,113), (75,109), (64,51), (113,44), (44,114), (85,43), (94,118), (130,36), (40,28), (13,144), (31,53), (141,18), (10,59), (80,63), (6,25), (7,32), (67,50), (10,103), (62,144),
 (45,129), (80,89), (51,3), (21,54), (57,39), (28,142), (100,133), (88,32), (99,25), (139,130), (130,148), (102,115), (86,84), (105,43), (143,101), (96,131), (132,101), (151,100), (150,99),
 (126,45), (48,91), (27,104), (55,80), (63,76), (88,129), (108,135), (127,129), (109,145), (30,142), (93,34), (97,86), (59,92), (3,57), (40,38), (63,4), (93,88), (85,145), (134,95), (129,61),
 (105,152), (98,137), (65,66), (85,24), (148,24), (123,86), (71,97), (140,125), (5,144), (113,116), (114,143), (13,29), (7,96), (21,150), (115,17), (89,96), (111,35), (72,18), (67,3), (56,150),
 (79,144), (43,95), (59,37), (67,85), (45,96), (9,86), (13,3), (103,93), (104,92), (13,89), (21,111), (74,47), (23,74), (134,69), (73,27), (61,12), (98,23), (111,13), (101,46), (114,31),
 (149,128), (95,53), (41,62), (91,70), (28,139), (16,33), (108,6), (122,21), (141,110), (101,73), (93,110), (35,139), (105,144), (44,6), (25,27), (65,55), (22,12), (22,36), (51,64), (13,73),
 (147,7), (7,137), (67,133), (115,102), (26,65), (31,29), (40,30), (46,14), (143,98), (128,77), (131,98), (119,105), (137,147), (32,141), (47,77), (47,75), (147,13), (130,66), (129,45), (28,129),
 (149,1), (141,76), (90,91), (119,109), (115,84), (116,94), (76,21), (136,31), (93,59), (18,113), (87,118), (36,51), (40,9), (71,102), (137,46), (64,81), (108,119), (38,65), (123,8), (107,40),
 (100,3), (98,86), (94,131), (104,110), (79,80), (76,51), (80,94), (47,76), (83,43), (107,9), (112,29), (152,28), (124,97), (59,88), (76,70), (140,94), (10,140), (125,101), (101,91), (83,81),
 (145,44), (139,63), (136,96), (147,101), (149,83), (53,73), (72,131), (32,96), (151,97), (33,12), (29,74), (22,24), (127,125), (18,6), (26,95), (61,40), (27,28), (55,52), (83,42), (109,142),
 (106,91), (58,122), (60,115), (13,58), (90,120), (47,114), (33,73), (147,45), (48,27), (40,62), (42,80), (42,123), (46,131), (63,60), (150,70), (68,8), (136,27), (115,24), (22,45), (76,47),
 (141,128), (86,132), (123,66), (148,48), (111,118), (83,1), (16,54), (36,22), (6,14), (44,62), (38,97), (33,43), (48,90), (20,110), (35,30), (123,127), (70,128), (36,123), (9,35)]

Process finished with exit code 0
```

Рисунок 3.2 – Робота алгоритму при 1000 ітераціях

3.3 Тестування алгоритму

Протестуємо алгоритм на різних вхідних даних. Тестування буде проходити на 100 ітераціях, популяція – 100 осіб.

Вхідні дані: 20 найкращих осіб, 0.1 ймовірність мутації.

Результат:

```
Initial distance: 22012
Final distance: 21813
Number of iterations: 100
```

Вхідні дані: 20 найкращих осіб, 0.3 ймовірність мутації.

Результат:

```
Initial distance: 22159
Final distance: 21926
Number of iterations: 100
```

Вхідні дані: 20 найкращих осіб, 0.6 ймовірність мутації.

Результат:

```
E:\ads_lab5\venv\Scripts\python.exe E:/ads_lab5/main.py
Initial distance: 22108
Final distance: 21876
Number of iterations: 100
```

Вхідні дані: 40 найкращих осіб, 0.1 ймовірність мутації.

Результат:

```
E:\ads_lab5\venv\Scripts\python.exe E:/ads_lab5/main.py
Initial distance: 22338
Final distance: 22208
Number of iterations: 100
```

Вхідні дані: 40 найкращих осіб, 0.3 ймовірність мутації.

Результат:

```
Initial distance: 21885
Final distance: 21443
Number of iterations: 100
```

Вхідні дані: 40 найкращих осіб, 0.6 ймовірність мутації.

Результат:

```
E:\ads_lab5\venv\Scripts\python.exe E:/ads_lab5/main.py
Initial distance: 22146
Final distance: 22003
Number of iterations: 100
```

Вхідні дані: 60 найкращих осіб, 0.1 ймовірність мутації.

Результат:

```
E:\ads_lab5\venv\Scripts\python.exe E:/ads_lab5/main.py
Initial distance: 21618
Final distance: 21083
Number of iterations: 100
```

Вхідні дані: 60 найкращих осіб, 0.3 ймовірність мутації.

Результат:

```
E:\ads_lab5\venv\Scripts\python.exe E:/ads_lab5/main.py
Initial distance: 22374
Final distance: 21541
Number of iterations: 100
```

Вхідні дані: 60 найкращих осіб, 0.6 ймовірність мутації.

Результат:

```
E:\ads_lab5\venv\Scripts\python.exe E:/ads_lab5/main.py
Initial distance: 21439
Final distance: 21537
Number of iterations: 100
```

Отже, найкращі вхідні дані – 60 найкращих особин (60% від популяції) та 0.3 ймовірність мутації.

Проаналізуємо результати виконання програми, використовуючи дані вище.

Побудуємо графік залежності вартості від ітерацій.

Для цього побудуємо таблицю.

Ітерації	Вартість
100	22898
200	22550
300	22516
400	22439
500	22423

600	22477
700	22362
800	22301
900	22265
1000	22178

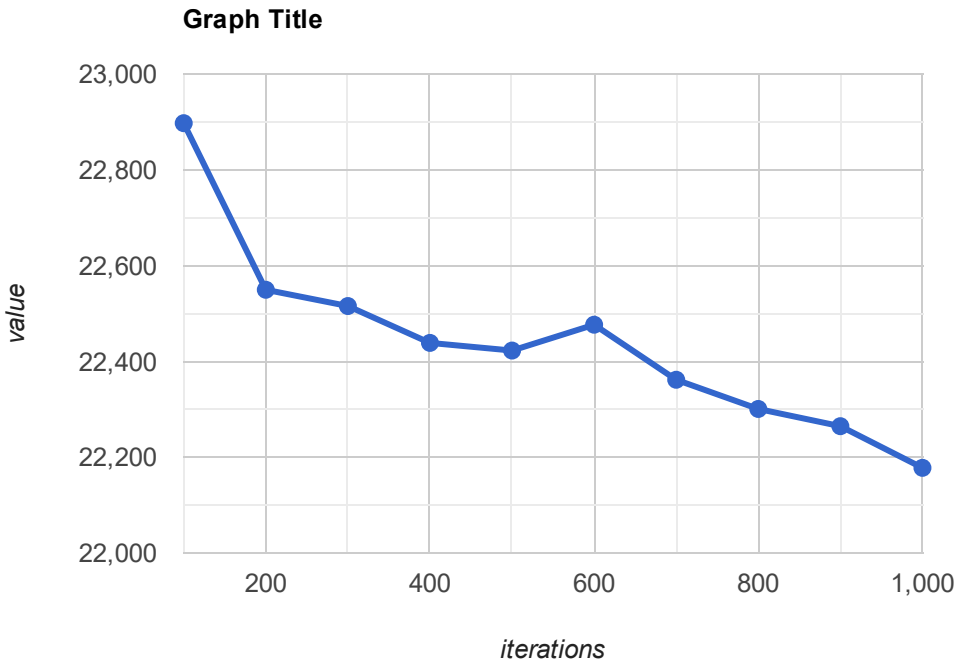

Рисунок 3.3 – Графік залежності результату від ітерацій

ВИСНОВОК

В рамках даної лабораторної роботи лабораторної роботи я застосував на практиці метаевристичний алгоритм на прикладі задачі комівояжера. Алгоритм було програмно реалізовано та проведено повний аналіз його роботи з тестуванням виконання на найкращих вхідних параметрах.

КРИТЕРІЇ ОЦІНЮВАННЯ

При здачі лабораторної роботи до 11.12.2022 включно максимальний бал дорівнює – 5. Після 11.12.2022 максимальний бал дорівнює – 1.

Критерії оцінювання у відсотках від максимального балу:

- покроковий алгоритм – 15%;
- програмна реалізація алгоритму – 50%;
- тестування алгоритму– 30%;
- висновок – 5%.