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 № 3 з дисципліни
«Проектування алгоритмів»

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

Виконав(ла)
ІП-24 Ротань Олександр Євгенович

Перевірив
Ахаладзе І. Е.

Київ 2023

Мета лабораторної роботи

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

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.

Варіант завдання

19	Задача про кліку (обчислювальна задача) + Бджолиний алгоритм

Виконання

1. Покроковий алгоритм

```
Class GraphClickBeeColony extends GraphBeeColony
  Private:
    random: Random

  Constructor(graph: Graph, numberOfWorkers: Integer, numberOfScouts: Integer)
    Call superclass constructor with graph, numberOfWorkers, numberOfScouts

  Constructor(graph: Graph)
    Call superclass constructor with graph, 900, 100

  Method getBestSolution() returns Graph
    click = getInitialClick()
    If click is empty
      Return click
    Return enhanceClick(click)

  Private Method enhanceClick(click: Graph) returns Graph
    count = 0
    While count < 100
      scoutedGraphs = scout(click)
      For each g in scoutedGraphs
        Repeat numberOfWorkers / numberOfScouts times
          workOnScoutedGraph(g)
      bestOpt = max of scoutedGraphs based on evaluateClick
      best = click
      If bestOpt is present
        best = bestOpt
```

```
        If evaluateClick(best) > evaluateClick(click)
            click = best
            count = 0
        Else
            Increment count
    Return click

Private Method evaluateClick(click: Graph) returns Integer
    If click is not a click
        Return -1
    Return size of click nodes * 10 + sum of specific neighbour sizes

Private Method workOnScoutedGraph(scoutedGraph: Graph)
    For each node in graph not in scoutedGraph
        Optionally add node to scoutedGraph and update edges
        If scoutedGraph is not a click
            Remove node from scoutedGraph

Private Method scout(initialClick: Graph) returns List of Graph
    scoutedGraphs = new empty list of Graph
    Perform operations to populate scoutedGraphs based on initialClick
    Return scoutedGraphs

Private Method isClick(click: Graph) returns Boolean
    Check if all nodes in click are interconnected
    Return true if interconnected, else false

Private Method getTrimmedNode(node: Node, click: Graph) returns Node
    Create trimmedNode with node's value
    Add neighbours to trimmedNode based on click
```

Return trimmedNode

Private Method getInitialClick() returns Graph

Create and populate a click graph based on initial nodes and neighbours

Return the click graph

End Class

2.1. Програмна реалізація

```
package org.example.lab5.beecolony;

import lombok.ToString;
import org.example.lab5.graph.AbstractGraph.Node;
import org.example.lab5.graph.Graph;
import org.example.lab5.graph.GraphFactory;

import java.util.ArrayList;
import java.util.Collections;
import java.util.List;
import java.util.Random;
import java.util.stream.IntStream;

@ToString
public class GraphClickBeeColony<T> extends GraphBeeColony<T> {

    private final Random random = new Random();

    public GraphClickBeeColony(Graph<T> graph, int numberOfWorkers, int numberOfScouts) {
        super(graph, numberOfWorkers, numberOfScouts);
    }
```

```
public GraphClickBeeColony(Graph<T> graph) {
    super(graph, 900, 100);
}

@Override
public Graph<T> getBestSolution() {
    var click = getInitialClick();
    if (click.getNodes().isEmpty()) {
        return click;
    }
    return enhanceClick(click);
}



/***
 * Enhances click using bee colony algorithm
 * @param click click to enhance
 * @return enhanced click
 */
private Graph<T> enhanceClick(Graph<T> click) {
    var count = 0;
    while (count < 100) {
        var scoutedGraphs = scout(click);
        scoutedGraphs.forEach(g -> {
            for (var i = 0; i < numberOfWorkers / numberOfScouts; i++) {
                workOnScoutedGraph(g);
            }
        });
        var bestOpt = scoutedGraphs.stream()
```

```
                .max((o1, o2) -> evaluateClick(o1) - evaluateClick(o2));
            var best = click;
            if (bestOpt.isPresent()) {
                best = bestOpt.get();
            }
            if (evaluateClick(best) > evaluateClick(click)) {
                click = best;
                count = 0;
            } else {
                count++;
            }
        }
        return click;
    }

    private int evaluateClick(Graph<T> click) {
        if (!isClick(click)) {
            return -1;
        }
        return click.getNodes().size() * 10 + graph.getNodes().stream().filter(node ->
                    click.hasNode(node.getValue()))
                .mapToInt(node -> node.getNeighbours().size() -
click.getNodes().size()).sum();
    }

    private void workOnScoutedGraph(Graph<T> scoutedGraph) {
        graph.getNodes().stream().filter(node ->
                    !scoutedGraph.hasNode(node.getValue()))
                .skip(random.nextInt(0, graph.getNodes().size() -
scoutedGraph.getNodes().size() + 1))
```

```
            .findAny().ifPresent(node -> {
                scoutedGraph.addNode(getTrimmedNode(node, scoutedGraph));
                node.getNeighbours().stream().filter(neighbour ->

scoutedGraph.hasNode(neighbour.getValue())).forEach(neighbour ->
                        scoutedGraph.addEdge(node.getValue(),     neighbour.getValue(),
1));
                if (!isClick(scoutedGraph)) {
                    scoutedGraph.removeNode(node.getValue());
                }
            });
    }

    private List<Graph<T>> scout(Graph<T> initialClick) {
        var scoutedGraphs = new ArrayList<Graph<T>>();
        IntStream.range(0, numberOfScouts * 9 / 10).forEach(i ->
                graph.getNodes().stream().filter(node                                    ->
!initialClick.hasNode(node.getValue()))
                        .skip(random.nextInt(0,              graph.getNodes().size()          -
initialClick.getNodes().size() + 1))
                        .findAny().ifPresent(node -> {
                            if                  (graph.getNodes().stream().filter(n              ->
initialClick.hasNode(n.getValue()))
                                    .allMatch(n          ->          graph.hasEdge(node.getValue(),
n.getValue()))) {
                                var newClick = initialClick.clone();
                                newClick.addNode(getTrimmedNode(node, newClick));
                                node.getNeighbours().stream().filter(neighbour ->

newClick.hasNode(neighbour.getValue())).forEach(neighbour ->
```

```
                        newClick.addEdge(node.getValue(),
neighbour.getValue(), 1));
                    scoutedGraphs.add(newClick);
                }
            }));
    IntStream.range(numberOfScouts * 9 / 10, (int) (numberOfScouts * 9.8 /
10)).forEach(i -> {
        var newClick = initialClick.clone();

newClick.removeNode(newClick.getNodes().stream().skip(random.nextInt(newCli
ck.getNodes().size())).findAny().get().getValue());
        graph.getNodes().stream().filter(n -> !newClick.hasNode(n.getValue()))
            .skip(random.nextInt(graph.getNodes().size())).findAny().ifPresent(n -
>
                newClick.addNode(getTrimmedNode(n, newClick)));
        for (var node : newClick.getNodes()) {
            for (var neighbour : node.getNeighbours()) {
                if (newClick.hasNode(neighbour.getValue())) {
                    newClick.addEdge(node.getValue(), neighbour.getValue(), 1);
                }
            }
        }
        if (isClick(newClick)) {
            scoutedGraphs.add(newClick);
        }
    });
    IntStream.range((int) (numberOfScouts * 9.8 / 10),
numberOfScouts).forEach(i -> {
        var newClick = getInitialClick();
        scoutedGraphs.add(newClick);
```

```
        });
        return scoutedGraphs;
    }

    private boolean isClick(Graph<T> click) {
        return click.getNodes().stream().allMatch(node ->
                click.getNodes().stream().filter(n -> !n.equals(node))
                        .allMatch(n -> click.hasEdge(node.getValue(), n.getValue())));
    }

    private Node<T> getTrimmedNode(Node<T> node, Graph<T> click) {
        var trimmedNode = new Node<>(node.getValue());
        node.getNeighbours().stream().filter(neighbour ->
click.hasNode(neighbour.getValue()))
                .forEach(trimmedNode::addNeighbour);
        return trimmedNode;
    }

    private Graph<T> getInitialClick() {
        Graph<T> click = GraphFactory.createGraph(graph.getClass());
        var nodes = new ArrayList<>(graph.getNodes());
        Collections.shuffle(nodes);
        for (var node : nodes) {
            var neighbours = new ArrayList<>(node.getNeighbours());
            Collections.shuffle(neighbours);
            for (var neighbour : neighbours) {
                if (neighbour.hasNeighbour(node.getValue())) {
                    click.addNode(getTrimmedNode(node, click));
                    click.addNode(getTrimmedNode(neighbour, click));
                    click.addEdge(node.getValue(), neighbour.getValue(), 1);
```

```
                return click;
            }
        }
    }
    return click;
  }
}
```

2.2. Приклади роботи

2.2.1. Приклад 1

Value: 0 Neighbours: [Node{value=4, edges=[end: 6 weight: 1, end: 3 weight: 1, end: 2 weight: 1, end: 1 weight: 1, end: 0 weight: 1, end: 9 weight: 1]}, Node{value=3, edges=[end: 6 weight: 1, end: 4 weight: 1, end: 2 weight: 1, end: 1 weight: 1, end: 0 weight: 1, end: 9 weight: 1, end: 8 weight: 1, end: 7 weight: 1]}, Node{value=2, edges=[end: 6 weight: 1, end: 4 weight: 1, end: 3 weight: 1, end: 1 weight: 1, end: 0 weight: 1, end: 8 weight: 1, end: 7 weight: 1]}, Node{value=1, edges=[end: 4 weight: 1, end: 3 weight: 1, end: 2 weight: 1, end: 0 weight: 1]}, Node{value=9, edges=[end: 4 weight: 1, end: 3 weight: 1, end: 0 weight: 1, end: 7 weight: 1]}, Node{value=8, edges=[end: 3 weight: 1, end: 2 weight: 1, end: 0 weight: 1]}]

Value: 1 Neighbours: [Node{value=4, edges=[end: 6 weight: 1, end: 3 weight: 1, end: 2 weight: 1, end: 1 weight: 1, end: 0 weight: 1, end: 9 weight: 1]}, Node{value=3, edges=[end: 6 weight: 1, end: 4 weight: 1, end: 2 weight: 1, end: 1 weight: 1, end: 0 weight: 1, end: 9 weight: 1, end: 8 weight: 1, end: 7 weight: 1]}, Node{value=2, edges=[end: 6 weight: 1, end: 4 weight: 1, end: 3 weight: 1, end: 1 weight: 1, end: 0 weight: 1, end: 8 weight: 1, end: 7 weight: 1]}, Node{value=0, edges=[end: 4 weight: 1, end: 3 weight: 1, end: 2 weight: 1, end: 1 weight: 1, end: 9 weight: 1, end: 8 weight: 1]}]

Value: 2 Neighbours: [Node{value=6, edges=[end: 5 weight: 1, end: 4 weight: 1, end: 3 weight: 1, end: 2 weight: 1]}, Node{value=4, edges=[end: 6 weight: 1, end: 3 weight: 1, end: 2 weight: 1, end: 1 weight: 1, end: 0 weight: 1, end: 9 weight: 1]}, Node{value=3, edges=[end: 6 weight: 1, end: 4 weight: 1, end: 2 weight: 1, end: 1 weight: 1, end: 0 weight: 1, end: 9 weight: 1, end: 8 weight: 1, end: 7 weight: 1]}, Node{value=1, edges=[end: 4 weight: 1, end: 3 weight: 1, end: 2 weight: 1, end: 0 weight: 1]}, Node{value=0, edges=[end: 4 weight: 1, end: 3 weight: 1, end: 2 weight: 1, end: 1 weight: 1, end: 9 weight: 1, end: 8 weight: 1]}, Node{value=8, edges=[end: 3 weight: 1, end: 2 weight: 1, end: 0 weight: 1]}, Node{value=7, edges=[end: 5 weight: 1, end: 3 weight: 1, end: 2 weight: 1, end: 9 weight: 1]}]

Value: 3 Neighbours: [Node{value=6, edges=[end: 5 weight: 1, end: 4 weight: 1, end: 3 weight: 1, end: 2 weight: 1]}, Node{value=4, edges=[end: 6 weight: 1, end: 3 weight: 1, end: 2 weight: 1, end: 1 weight: 1, end: 0 weight: 1, end: 9 weight: 1]}, Node{value=2, edges=[end: 6 weight: 1, end: 4 weight: 1, end: 3 weight: 1, end: 1 weight: 1, end: 0 weight: 1, end: 8 weight: 1, end: 7 weight: 1]}, Node{value=1, edges=[end: 4 weight: 1, end: 3 weight: 1, end: 2 weight: 1, end: 0 weight: 1]}, Node{value=0, edges=[end: 4 weight: 1, end: 3 weight: 1, end: 2 weight: 1, end: 1 weight: 1, end: 9 weight: 1, end: 8 weight: 1]}, Node{value=9, edges=[end: 4 weight: 1, end: 3 weight: 1, end: 0 weight: 1, end: 7 weight: 1]}, Node{value=8, edges=[end: 3 weight: 1, end: 2 weight: 1, end: 0 weight: 1]}, Node{value=7, edges=[end: 5 weight: 1, end: 3 weight: 1, end: 2 weight: 1, end: 9 weight: 1]}]
Value: 4 Neighbours: [Node{value=6, edges=[end: 5 weight: 1, end: 4 weight: 1, end: 3 weight: 1, end: 2 weight: 1]}, Node{value=3, edges=[end: 6 weight: 1, end: 4 weight: 1, end: 2 weight: 1, end: 1 weight: 1, end: 0 weight: 1, end: 9 weight: 1, end: 8 weight: 1, end: 7 weight: 1]}, Node{value=2, edges=[end: 6 weight: 1, end: 4 weight: 1, end: 3 weight: 1, end: 1 weight: 1, end: 0 weight: 1, end: 8 weight: 1, end: 7 weight: 1]}, Node{value=1, edges=[end: 4 weight: 1, end: 3 weight: 1, end: 2 weight: 1, end: 0 weight: 1]}, Node{value=0, edges=[end: 4 weight: 1, end: 3 weight: 1, end: 2 weight: 1, end: 1 weight: 1, end: 9 weight: 1, end: 8 weight: 1]}, Node{value=9, edges=[end: 4 weight: 1, end: 3 weight: 1, end: 0 weight: 1, end: 7 weight: 1]}]
Value: 5 Neighbours: [Node{value=6, edges=[end: 5 weight: 1, end: 4 weight: 1, end: 3 weight: 1, end: 2 weight: 1]}, Node{value=7, edges=[end: 5 weight: 1, end: 3 weight: 1, end: 2 weight: 1, end: 9 weight: 1]}]
Value: 6 Neighbours: [Node{value=5, edges=[end: 6 weight: 1, end: 7 weight: 1]}, Node{value=4, edges=[end: 6 weight: 1, end: 3 weight: 1, end: 2 weight: 1, end: 1 weight: 1, end: 0 weight: 1, end: 9 weight: 1]}, Node{value=3, edges=[end: 6 weight: 1, end: 4 weight: 1, end: 2 weight: 1, end: 1 weight: 1, end: 0 weight: 1, end: 9 weight: 1, end: 8 weight: 1, end: 7 weight: 1]}, Node{value=2, edges=[end: 6 weight: 1, end: 4 weight: 1, end: 3 weight: 1, end: 1 weight: 1, end: 0 weight: 1, end: 8 weight: 1, end: 7 weight: 1]}]
Value: 7 Neighbours: [Node{value=5, edges=[end: 6 weight: 1, end: 7 weight: 1]}, Node{value=3, edges=[end: 6 weight: 1, end: 4 weight: 1, end: 2 weight: 1, end: 1 weight: 1, end: 0 weight: 1, end: 9 weight: 1, end: 8 weight: 1, end: 7 weight: 1]}, Node{value=2, edges=[end: 6 weight: 1, end: 4 weight: 1, end: 3 weight: 1, end: 1 weight: 1, end: 0 weight: 1, end: 8 weight: 1, end: 7 weight: 1]}, Node{value=9, edges=[end: 4 weight: 1, end: 3 weight: 1, end: 0 weight: 1, end: 7 weight: 1]}]
Value: 8 Neighbours: [Node{value=3, edges=[end: 6 weight: 1, end: 4 weight: 1, end: 2 weight: 1, end: 1 weight: 1, end: 0 weight: 1, end: 9 weight: 1, end: 8 weight: 1, end: 7 weight: 1]},

Node{value=2, edges=[end: 6 weight: 1, end: 4 weight: 1, end: 3 weight: 1, end: 1 weight: 1, end: 0 weight: 1, end: 8 weight: 1, end: 7 weight: 1]}, Node{value=0, edges=[end: 4 weight: 1, end: 3 weight: 1, end: 2 weight: 1, end: 1 weight: 1, end: 9 weight: 1, end: 8 weight: 1]}]
Value: 9 Neighbours: [Node{value=4, edges=[end: 6 weight: 1, end: 3 weight: 1, end: 2 weight: 1, end: 1 weight: 1, end: 0 weight: 1, end: 9 weight: 1]}, Node{value=3, edges=[end: 6 weight: 1, end: 4 weight: 1, end: 2 weight: 1, end: 1 weight: 1, end: 0 weight: 1, end: 9 weight: 1, end: 8 weight: 1, end: 7 weight: 1]}, Node{value=0, edges=[end: 4 weight: 1, end: 3 weight: 1, end: 2 weight: 1, end: 1 weight: 1, end: 9 weight: 1, end: 8 weight: 1]}, Node{value=7, edges=[end: 5 weight: 1, end: 3 weight: 1, end: 2 weight: 1, end: 9 weight: 1]}]
Biggest click: UndirectedGraph(super=Graph{nodes=[3, 1, 4, 2, 0]})

2.2.2. Приклад 2

Value: 0 Neighbours: [Node{value=3, edges=[end: 5 weight: 1, end: 2 weight: 1, end: 1 weight: 1, end: 0 weight: 1]}, Node{value=2, edges=[end: 3 weight: 1, end: 1 weight: 1, end: 0 weight: 1, end: 12 weight: 1, end: 10 weight: 1, end: 9 weight: 1]}, Node{value=1, edges=[end: 6 weight: 1, end: 3 weight: 1, end: 2 weight: 1, end: 0 weight: 1, end: 12 weight: 1, end: 10 weight: 1]}, Node{value=11, edges=[end: 6 weight: 1, end: 5 weight: 1, end: 4 weight: 1, end: 0 weight: 1, end: 12 weight: 1, end: 10 weight: 1, end: 9 weight: 1, end: 8 weight: 1, end: 7 weight: 1]}, Node{value=7, edges=[end: 6 weight: 1, end: 5 weight: 1, end: 4 weight: 1, end: 0 weight: 1, end: 12 weight: 1, end: 11 weight: 1, end: 10 weight: 1, end: 9 weight: 1, end: 8 weight: 1]}]
Value: 1 Neighbours: [Node{value=6, edges=[end: 5 weight: 1, end: 4 weight: 1, end: 1 weight: 1, end: 12 weight: 1, end: 11 weight: 1, end: 10 weight: 1, end: 9 weight: 1, end: 8 weight: 1, end: 7 weight: 1]}, Node{value=3, edges=[end: 5 weight: 1, end: 2 weight: 1, end: 1 weight: 1, end: 0 weight: 1]}, Node{value=2, edges=[end: 3 weight: 1, end: 1 weight: 1, end: 0 weight: 1, end: 12 weight: 1, end: 10 weight: 1, end: 9 weight: 1]}, Node{value=0, edges=[end: 3 weight: 1, end: 2 weight: 1, end: 1 weight: 1, end: 11 weight: 1, end: 7 weight: 1]}, Node{value=12, edges=[end: 6 weight: 1, end: 5 weight: 1, end: 4 weight: 1, end: 2 weight: 1, end: 1 weight: 1, end: 11 weight: 1, end: 10 weight: 1, end: 9 weight: 1, end: 8 weight: 1, end: 7 weight: 1]}, Node{value=10, edges=[end: 6 weight: 1, end: 5 weight: 1, end: 4 weight: 1, end: 2 weight: 1, end: 1 weight: 1, end: 12 weight: 1, end: 11 weight: 1, end: 9 weight: 1, end: 8 weight: 1, end: 7 weight: 1]}]
Value: 2 Neighbours: [Node{value=3, edges=[end: 5 weight: 1, end: 2 weight: 1, end: 1 weight: 1, end: 0 weight: 1]}, Node{value=1, edges=[end: 6 weight: 1, end: 3 weight: 1, end: 2 weight: 1, end: 0 weight: 1, end: 12 weight: 1, end: 10 weight: 1]}, Node{value=0, edges=[end: 3 weight: 1, end: 2 weight: 1, end: 1 weight: 1, end: 11 weight: 1, end: 7 weight: 1]}, Node{value=12, edges=[end: 6 weight: 1, end: 5 weight: 1, end: 4 weight: 1, end: 2 weight: 1, end: 1 weight: 1, end: 11 weight: 1, end: 10 weight: 1, end: 9 weight: 1, end: 8 weight: 1, end: 7 weight: 1]},

Node{value=10, edges=[end: 6 weight: 1, end: 5 weight: 1, end: 4 weight: 1, end: 2 weight: 1, end: 1 weight: 1, end: 12 weight: 1, end: 11 weight: 1, end: 9 weight: 1, end: 8 weight: 1, end: 7 weight: 1]}, Node{value=9, edges=[end: 6 weight: 1, end: 5 weight: 1, end: 4 weight: 1, end: 2 weight: 1, end: 12 weight: 1, end: 11 weight: 1, end: 10 weight: 1, end: 8 weight: 1, end: 7 weight: 1]}]
Value: 3 Neighbours: [Node{value=5, edges=[end: 6 weight: 1, end: 4 weight: 1, end: 3 weight: 1, end: 12 weight: 1, end: 11 weight: 1, end: 10 weight: 1, end: 9 weight: 1, end: 8 weight: 1, end: 7 weight: 1]}, Node{value=2, edges=[end: 3 weight: 1, end: 1 weight: 1, end: 0 weight: 1, end: 12 weight: 1, end: 10 weight: 1, end: 9 weight: 1]}, Node{value=1, edges=[end: 6 weight: 1, end: 3 weight: 1, end: 2 weight: 1, end: 0 weight: 1, end: 12 weight: 1, end: 10 weight: 1]}, Node{value=0, edges=[end: 3 weight: 1, end: 2 weight: 1, end: 1 weight: 1, end: 11 weight: 1, end: 7 weight: 1]}]
Value: 4 Neighbours: [Node{value=6, edges=[end: 5 weight: 1, end: 4 weight: 1, end: 1 weight: 1, end: 12 weight: 1, end: 11 weight: 1, end: 10 weight: 1, end: 9 weight: 1, end: 8 weight: 1, end: 7 weight: 1]}, Node{value=5, edges=[end: 6 weight: 1, end: 4 weight: 1, end: 3 weight: 1, end: 12 weight: 1, end: 11 weight: 1, end: 10 weight: 1, end: 9 weight: 1, end: 8 weight: 1, end: 7 weight: 1]}, Node{value=12, edges=[end: 6 weight: 1, end: 5 weight: 1, end: 4 weight: 1, end: 2 weight: 1, end: 1 weight: 1, end: 11 weight: 1, end: 10 weight: 1, end: 9 weight: 1, end: 8 weight: 1, end: 7 weight: 1]}, Node{value=11, edges=[end: 6 weight: 1, end: 5 weight: 1, end: 4 weight: 1, end: 0 weight: 1, end: 12 weight: 1, end: 10 weight: 1, end: 9 weight: 1, end: 8 weight: 1, end: 7 weight: 1]}, Node{value=10, edges=[end: 6 weight: 1, end: 5 weight: 1, end: 4 weight: 1, end: 2 weight: 1, end: 1 weight: 1, end: 12 weight: 1, end: 11 weight: 1, end: 9 weight: 1, end: 8 weight: 1, end: 7 weight: 1]}, Node{value=9, edges=[end: 6 weight: 1, end: 5 weight: 1, end: 4 weight: 1, end: 2 weight: 1, end: 12 weight: 1, end: 11 weight: 1, end: 10 weight: 1, end: 8 weight: 1, end: 7 weight: 1]}, Node{value=8, edges=[end: 6 weight: 1, end: 5 weight: 1, end: 4 weight: 1, end: 12 weight: 1, end: 11 weight: 1, end: 10 weight: 1, end: 9 weight: 1, end: 7 weight: 1]}, Node{value=7, edges=[end: 6 weight: 1, end: 5 weight: 1, end: 4 weight: 1, end: 0 weight: 1, end: 12 weight: 1, end: 11 weight: 1, end: 10 weight: 1, end: 9 weight: 1, end: 8 weight: 1]}]
Value: 5 Neighbours: [Node{value=6, edges=[end: 5 weight: 1, end: 4 weight: 1, end: 1 weight: 1, end: 12 weight: 1, end: 11 weight: 1, end: 10 weight: 1, end: 9 weight: 1, end: 8 weight: 1, end: 7 weight: 1]}, Node{value=4, edges=[end: 6 weight: 1, end: 5 weight: 1, end: 12 weight: 1, end: 11 weight: 1, end: 10 weight: 1, end: 9 weight: 1, end: 8 weight: 1, end: 7 weight: 1]}, Node{value=3, edges=[end: 5 weight: 1, end: 2 weight: 1, end: 1 weight: 1, end: 0 weight: 1]}, Node{value=12, edges=[end: 6 weight: 1, end: 5 weight: 1, end: 4 weight: 1, end: 2 weight: 1, end: 1 weight: 1, end: 11 weight: 1, end: 10 weight: 1, end: 9 weight: 1, end: 8 weight: 1, end: 7 weight: 1]}, Node{value=11, edges=[end: 6 weight: 1, end: 5 weight: 1, end: 4 weight: 1, end: 0 weight: 1, end: 12 weight: 1, end: 10 weight: 1, end: 9 weight: 1, end: 8 weight: 1, end: 7 weight: 1]},

Node{value=10, edges=[end: 6 weight: 1, end: 5 weight: 1, end: 4 weight: 1, end: 2 weight: 1, end: 1 weight: 1, end: 12 weight: 1, end: 11 weight: 1, end: 9 weight: 1, end: 8 weight: 1, end: 7 weight: 1]}, Node{value=9, edges=[end: 6 weight: 1, end: 5 weight: 1, end: 4 weight: 1, end: 2 weight: 1, end: 12 weight: 1, end: 11 weight: 1, end: 10 weight: 1, end: 8 weight: 1, end: 7 weight: 1]}, Node{value=8, edges=[end: 6 weight: 1, end: 5 weight: 1, end: 4 weight: 1, end: 12 weight: 1, end: 11 weight: 1, end: 10 weight: 1, end: 9 weight: 1, end: 7 weight: 1]}, Node{value=7, edges=[end: 6 weight: 1, end: 5 weight: 1, end: 4 weight: 1, end: 0 weight: 1, end: 12 weight: 1, end: 11 weight: 1, end: 10 weight: 1, end: 9 weight: 1, end: 8 weight: 1]}]
Value: 6 Neighbours: [Node{value=5, edges=[end: 6 weight: 1, end: 4 weight: 1, end: 3 weight: 1, end: 12 weight: 1, end: 11 weight: 1, end: 10 weight: 1, end: 9 weight: 1, end: 8 weight: 1, end: 7 weight: 1]}, Node{value=4, edges=[end: 6 weight: 1, end: 5 weight: 1, end: 12 weight: 1, end: 11 weight: 1, end: 10 weight: 1, end: 9 weight: 1, end: 8 weight: 1, end: 7 weight: 1]}, Node{value=1, edges=[end: 6 weight: 1, end: 3 weight: 1, end: 2 weight: 1, end: 0 weight: 1, end: 12 weight: 1, end: 10 weight: 1]}, Node{value=12, edges=[end: 6 weight: 1, end: 5 weight: 1, end: 4 weight: 1, end: 2 weight: 1, end: 1 weight: 1, end: 11 weight: 1, end: 10 weight: 1, end: 9 weight: 1, end: 8 weight: 1, end: 7 weight: 1]}, Node{value=11, edges=[end: 6 weight: 1, end: 5 weight: 1, end: 4 weight: 1, end: 0 weight: 1, end: 12 weight: 1, end: 10 weight: 1, end: 9 weight: 1, end: 8 weight: 1, end: 7 weight: 1]}, Node{value=10, edges=[end: 6 weight: 1, end: 5 weight: 1, end: 4 weight: 1, end: 2 weight: 1, end: 1 weight: 1, end: 12 weight: 1, end: 11 weight: 1, end: 9 weight: 1, end: 8 weight: 1, end: 7 weight: 1]}, Node{value=9, edges=[end: 6 weight: 1, end: 5 weight: 1, end: 4 weight: 1, end: 2 weight: 1, end: 12 weight: 1, end: 11 weight: 1, end: 10 weight: 1, end: 8 weight: 1, end: 7 weight: 1]}, Node{value=8, edges=[end: 6 weight: 1, end: 5 weight: 1, end: 4 weight: 1, end: 12 weight: 1, end: 11 weight: 1, end: 10 weight: 1, end: 9 weight: 1, end: 7 weight: 1]}, Node{value=7, edges=[end: 6 weight: 1, end: 5 weight: 1, end: 4 weight: 1, end: 0 weight: 1, end: 12 weight: 1, end: 11 weight: 1, end: 10 weight: 1, end: 9 weight: 1, end: 8 weight: 1]}]
Value: 7 Neighbours: [Node{value=6, edges=[end: 5 weight: 1, end: 4 weight: 1, end: 1 weight: 1, end: 12 weight: 1, end: 11 weight: 1, end: 10 weight: 1, end: 9 weight: 1, end: 8 weight: 1, end: 7 weight: 1]}, Node{value=5, edges=[end: 6 weight: 1, end: 4 weight: 1, end: 3 weight: 1, end: 12 weight: 1, end: 11 weight: 1, end: 10 weight: 1, end: 9 weight: 1, end: 8 weight: 1, end: 7 weight: 1]}, Node{value=4, edges=[end: 6 weight: 1, end: 5 weight: 1, end: 12 weight: 1, end: 11 weight: 1, end: 10 weight: 1, end: 9 weight: 1, end: 8 weight: 1, end: 7 weight: 1]}, Node{value=0, edges=[end: 3 weight: 1, end: 2 weight: 1, end: 1 weight: 1, end: 11 weight: 1, end: 7 weight: 1]}, Node{value=12, edges=[end: 6 weight: 1, end: 5 weight: 1, end: 4 weight: 1, end: 2 weight: 1, end: 1 weight: 1, end: 11 weight: 1, end: 10 weight: 1, end: 9 weight: 1, end: 8 weight: 1, end: 7 weight: 1]}, Node{value=11, edges=[end: 6 weight: 1, end: 5 weight: 1, end: 4 weight: 1, end: 0 weight:

1, end: 12 weight: 1, end: 10 weight: 1, end: 9 weight: 1, end: 8 weight: 1, end: 7 weight: 1]}, Node{value=10, edges=[end: 6 weight: 1, end: 5 weight: 1, end: 4 weight: 1, end: 2 weight: 1, end: 1 weight: 1, end: 12 weight: 1, end: 11 weight: 1, end: 9 weight: 1, end: 8 weight: 1, end: 7 weight: 1]}, Node{value=9, edges=[end: 6 weight: 1, end: 5 weight: 1, end: 4 weight: 1, end: 2 weight: 1, end: 12 weight: 1, end: 11 weight: 1, end: 10 weight: 1, end: 8 weight: 1, end: 7 weight: 1]}, Node{value=8, edges=[end: 6 weight: 1, end: 5 weight: 1, end: 4 weight: 1, end: 12 weight: 1, end: 11 weight: 1, end: 10 weight: 1, end: 9 weight: 1, end: 7 weight: 1]}]
Value: 8 Neighbours: [Node{value=6, edges=[end: 5 weight: 1, end: 4 weight: 1, end: 1 weight: 1, end: 12 weight: 1, end: 11 weight: 1, end: 10 weight: 1, end: 9 weight: 1, end: 8 weight: 1, end: 7 weight: 1]}, Node{value=5, edges=[end: 6 weight: 1, end: 4 weight: 1, end: 3 weight: 1, end: 12 weight: 1, end: 11 weight: 1, end: 10 weight: 1, end: 9 weight: 1, end: 8 weight: 1, end: 7 weight: 1]}, Node{value=4, edges=[end: 6 weight: 1, end: 5 weight: 1, end: 12 weight: 1, end: 11 weight: 1, end: 10 weight: 1, end: 9 weight: 1, end: 8 weight: 1, end: 7 weight: 1]}, Node{value=12, edges=[end: 6 weight: 1, end: 5 weight: 1, end: 4 weight: 1, end: 2 weight: 1, end: 1 weight: 1, end: 11 weight: 1, end: 10 weight: 1, end: 9 weight: 1, end: 8 weight: 1, end: 7 weight: 1]}, Node{value=11, edges=[end: 6 weight: 1, end: 5 weight: 1, end: 4 weight: 1, end: 0 weight: 1, end: 12 weight: 1, end: 10 weight: 1, end: 9 weight: 1, end: 8 weight: 1, end: 7 weight: 1]}, Node{value=10, edges=[end: 6 weight: 1, end: 5 weight: 1, end: 4 weight: 1, end: 2 weight: 1, end: 1 weight: 1, end: 12 weight: 1, end: 11 weight: 1, end: 9 weight: 1, end: 8 weight: 1, end: 7 weight: 1]}, Node{value=9, edges=[end: 6 weight: 1, end: 5 weight: 1, end: 4 weight: 1, end: 2 weight: 1, end: 12 weight: 1, end: 11 weight: 1, end: 10 weight: 1, end: 8 weight: 1, end: 7 weight: 1]}, Node{value=7, edges=[end: 6 weight: 1, end: 5 weight: 1, end: 4 weight: 1, end: 0 weight: 1, end: 12 weight: 1, end: 11 weight: 1, end: 10 weight: 1, end: 9 weight: 1, end: 8 weight: 1]}]
Value: 9 Neighbours: [Node{value=6, edges=[end: 5 weight: 1, end: 4 weight: 1, end: 1 weight: 1, end: 12 weight: 1, end: 11 weight: 1, end: 10 weight: 1, end: 9 weight: 1, end: 8 weight: 1, end: 7 weight: 1]}, Node{value=5, edges=[end: 6 weight: 1, end: 4 weight: 1, end: 3 weight: 1, end: 12 weight: 1, end: 11 weight: 1, end: 10 weight: 1, end: 9 weight: 1, end: 8 weight: 1, end: 7 weight: 1]}, Node{value=4, edges=[end: 6 weight: 1, end: 5 weight: 1, end: 12 weight: 1, end: 11 weight: 1, end: 10 weight: 1, end: 9 weight: 1, end: 8 weight: 1, end: 7 weight: 1]}, Node{value=2, edges=[end: 3 weight: 1, end: 1 weight: 1, end: 0 weight: 1, end: 12 weight: 1, end: 10 weight: 1, end: 9 weight: 1]}, Node{value=12, edges=[end: 6 weight: 1, end: 5 weight: 1, end: 4 weight: 1, end: 2 weight: 1, end: 1 weight: 1, end: 11 weight: 1, end: 10 weight: 1, end: 9 weight: 1, end: 8 weight: 1, end: 7 weight: 1]}, Node{value=11, edges=[end: 6 weight: 1, end: 5 weight: 1, end: 4 weight: 1, end: 0 weight: 1, end: 12 weight: 1, end: 10 weight: 1, end: 9 weight: 1, end: 8 weight: 1, end: 7 weight: 1]}, Node{value=10, edges=[end: 6 weight: 1, end: 5 weight: 1, end: 4 weight:

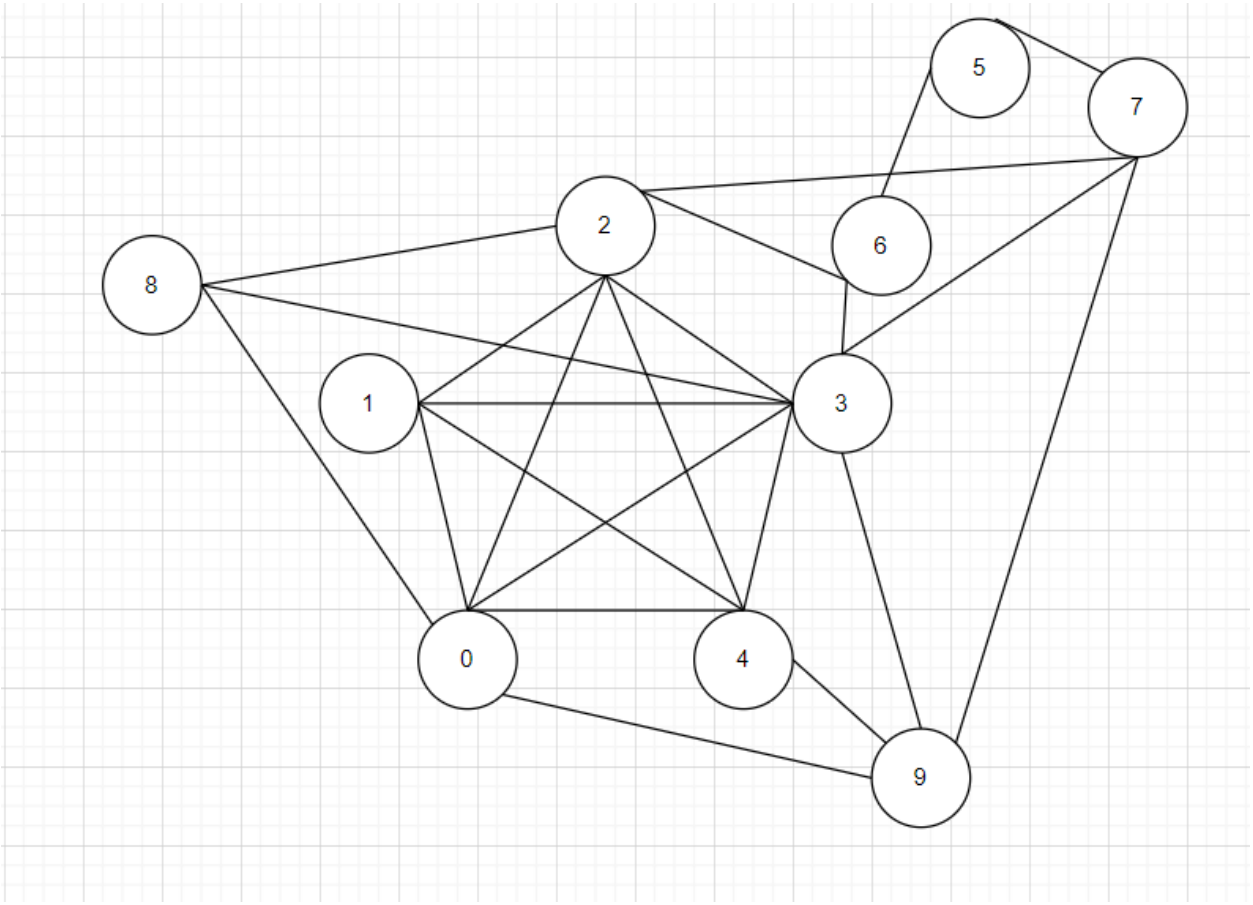
1, end: 2 weight: 1, end: 1 weight: 1, end: 12 weight: 1, end: 11 weight: 1, end: 9 weight: 1, end: 8 weight: 1, end: 7 weight: 1]}, Node{value=8, edges=[end: 6 weight: 1, end: 5 weight: 1, end: 4 weight: 1, end: 12 weight: 1, end: 11 weight: 1, end: 10 weight: 1, end: 9 weight: 1, end: 7 weight: 1]}, Node{value=7, edges=[end: 6 weight: 1, end: 5 weight: 1, end: 4 weight: 1, end: 0 weight: 1, end: 12 weight: 1, end: 11 weight: 1, end: 10 weight: 1, end: 9 weight: 1, end: 8 weight: 1]}]
Value: 10 Neighbours: [Node{value=6, edges=[end: 5 weight: 1, end: 4 weight: 1, end: 1 weight: 1, end: 12 weight: 1, end: 11 weight: 1, end: 10 weight: 1, end: 9 weight: 1, end: 8 weight: 1, end: 7 weight: 1]}, Node{value=5, edges=[end: 6 weight: 1, end: 4 weight: 1, end: 3 weight: 1, end: 12 weight: 1, end: 11 weight: 1, end: 10 weight: 1, end: 9 weight: 1, end: 8 weight: 1, end: 7 weight: 1]}, Node{value=4, edges=[end: 6 weight: 1, end: 5 weight: 1, end: 12 weight: 1, end: 11 weight: 1, end: 10 weight: 1, end: 9 weight: 1, end: 8 weight: 1, end: 7 weight: 1]}, Node{value=2, edges=[end: 3 weight: 1, end: 1 weight: 1, end: 0 weight: 1, end: 12 weight: 1, end: 10 weight: 1, end: 9 weight: 1]}, Node{value=1, edges=[end: 6 weight: 1, end: 3 weight: 1, end: 2 weight: 1, end: 0 weight: 1, end: 12 weight: 1, end: 10 weight: 1]}, Node{value=12, edges=[end: 6 weight: 1, end: 5 weight: 1, end: 4 weight: 1, end: 2 weight: 1, end: 1 weight: 1, end: 11 weight: 1, end: 10 weight: 1, end: 9 weight: 1, end: 8 weight: 1, end: 7 weight: 1]}, Node{value=11, edges=[end: 6 weight: 1, end: 5 weight: 1, end: 4 weight: 1, end: 0 weight: 1, end: 12 weight: 1, end: 10 weight: 1, end: 9 weight: 1, end: 8 weight: 1, end: 7 weight: 1]}, Node{value=9, edges=[end: 6 weight: 1, end: 5 weight: 1, end: 4 weight: 1, end: 2 weight: 1, end: 12 weight: 1, end: 11 weight: 1, end: 10 weight: 1, end: 8 weight: 1, end: 7 weight: 1]}, Node{value=8, edges=[end: 6 weight: 1, end: 5 weight: 1, end: 4 weight: 1, end: 12 weight: 1, end: 11 weight: 1, end: 10 weight: 1, end: 9 weight: 1, end: 7 weight: 1]}, Node{value=7, edges=[end: 6 weight: 1, end: 5 weight: 1, end: 4 weight: 1, end: 0 weight: 1, end: 12 weight: 1, end: 11 weight: 1, end: 10 weight: 1, end: 9 weight: 1, end: 8 weight: 1]}]
Value: 11 Neighbours: [Node{value=6, edges=[end: 5 weight: 1, end: 4 weight: 1, end: 1 weight: 1, end: 12 weight: 1, end: 11 weight: 1, end: 10 weight: 1, end: 9 weight: 1, end: 8 weight: 1, end: 7 weight: 1]}, Node{value=5, edges=[end: 6 weight: 1, end: 4 weight: 1, end: 3 weight: 1, end: 12 weight: 1, end: 11 weight: 1, end: 10 weight: 1, end: 9 weight: 1, end: 8 weight: 1, end: 7 weight: 1]}, Node{value=4, edges=[end: 6 weight: 1, end: 5 weight: 1, end: 12 weight: 1, end: 11 weight: 1, end: 10 weight: 1, end: 9 weight: 1, end: 8 weight: 1, end: 7 weight: 1]}, Node{value=0, edges=[end: 3 weight: 1, end: 2 weight: 1, end: 1 weight: 1, end: 11 weight: 1, end: 7 weight: 1]}, Node{value=12, edges=[end: 6 weight: 1, end: 5 weight: 1, end: 4 weight: 1, end: 2 weight: 1, end: 1 weight: 1, end: 11 weight: 1, end: 10 weight: 1, end: 9 weight: 1, end: 8 weight: 1, end: 7 weight: 1]}, Node{value=10, edges=[end: 6 weight: 1, end: 5 weight: 1, end: 4 weight: 1, end: 2 weight: 1, end: 1 weight: 1, end: 12 weight: 1, end: 11 weight: 1, end: 9 weight: 1, end: 8 weight: 1, end:

7 weight: 1]}, Node{value=9, edges=[end: 6 weight: 1, end: 5 weight: 1, end: 4 weight: 1, end: 2 weight: 1, end: 12 weight: 1, end: 11 weight: 1, end: 10 weight: 1, end: 8 weight: 1, end: 7 weight: 1]}, Node{value=8, edges=[end: 6 weight: 1, end: 5 weight: 1, end: 4 weight: 1, end: 12 weight: 1, end: 11 weight: 1, end: 10 weight: 1, end: 9 weight: 1, end: 7 weight: 1]}, Node{value=7, edges=[end: 6 weight: 1, end: 5 weight: 1, end: 4 weight: 1, end: 0 weight: 1, end: 12 weight: 1, end: 11 weight: 1, end: 10 weight: 1, end: 9 weight: 1, end: 8 weight: 1]}]
Value: 12 Neighbours: [Node{value=6, edges=[end: 5 weight: 1, end: 4 weight: 1, end: 1 weight: 1, end: 12 weight: 1, end: 11 weight: 1, end: 10 weight: 1, end: 9 weight: 1, end: 8 weight: 1, end: 7 weight: 1]}, Node{value=5, edges=[end: 6 weight: 1, end: 4 weight: 1, end: 3 weight: 1, end: 12 weight: 1, end: 11 weight: 1, end: 10 weight: 1, end: 9 weight: 1, end: 8 weight: 1, end: 7 weight: 1]}, Node{value=4, edges=[end: 6 weight: 1, end: 5 weight: 1, end: 12 weight: 1, end: 11 weight: 1, end: 10 weight: 1, end: 9 weight: 1, end: 8 weight: 1, end: 7 weight: 1]}, Node{value=2, edges=[end: 3 weight: 1, end: 1 weight: 1, end: 0 weight: 1, end: 12 weight: 1, end: 10 weight: 1, end: 9 weight: 1]}, Node{value=1, edges=[end: 6 weight: 1, end: 3 weight: 1, end: 2 weight: 1, end: 0 weight: 1, end: 12 weight: 1, end: 10 weight: 1]}, Node{value=11, edges=[end: 6 weight: 1, end: 5 weight: 1, end: 4 weight: 1, end: 0 weight: 1, end: 12 weight: 1, end: 10 weight: 1, end: 9 weight: 1, end: 8 weight: 1, end: 7 weight: 1]}, Node{value=10, edges=[end: 6 weight: 1, end: 5 weight: 1, end: 4 weight: 1, end: 2 weight: 1, end: 1 weight: 1, end: 12 weight: 1, end: 11 weight: 1, end: 9 weight: 1, end: 8 weight: 1, end: 7 weight: 1]}, Node{value=9, edges=[end: 6 weight: 1, end: 5 weight: 1, end: 4 weight: 1, end: 2 weight: 1, end: 12 weight: 1, end: 11 weight: 1, end: 10 weight: 1, end: 8 weight: 1, end: 7 weight: 1]}, Node{value=8, edges=[end: 6 weight: 1, end: 5 weight: 1, end: 4 weight: 1, end: 12 weight: 1, end: 11 weight: 1, end: 10 weight: 1, end: 9 weight: 1, end: 7 weight: 1]}, Node{value=7, edges=[end: 6 weight: 1, end: 5 weight: 1, end: 4 weight: 1, end: 0 weight: 1, end: 12 weight: 1, end: 11 weight: 1, end: 10 weight: 1, end: 9 weight: 1, end: 8 weight: 1]}]
Biggest click: UndirectedGraph(super=Graph{nodes=[8, 4, 10, 6, 5, 11, 9, 12, 7]})

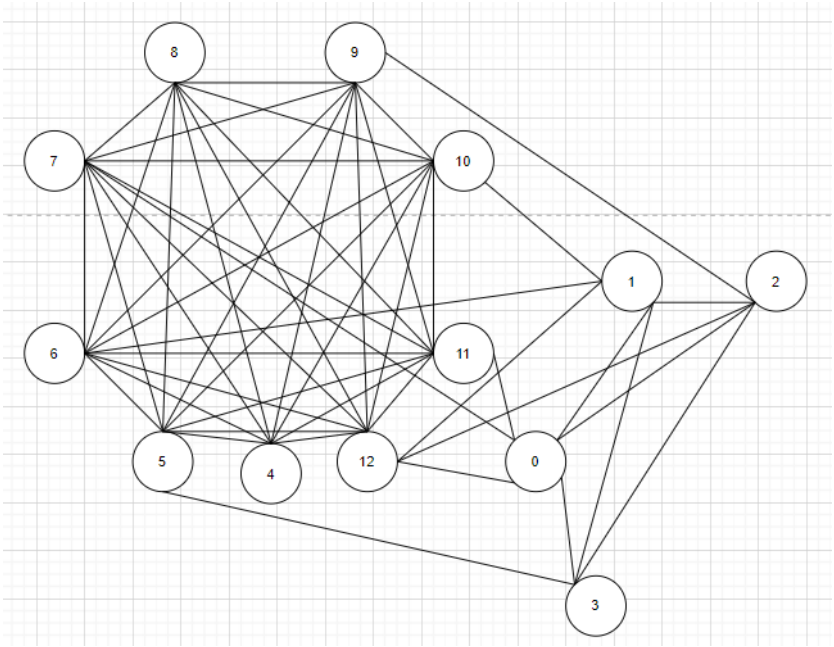
3. Тестування алгоритму

Розглянемо приклади роботи програми.

Для першого прикладу граф виглядає наступним чином:



З малюнку явно видно, що найбільшою клікою є підграф з вершинами 0,1,2,3,4
Тест номер 2 перевіряє, чи падає програма в локальний мінімум. Розглянемо граф на рисунку:



У даному графі присутні 2 кліки: значення 4-12 та 0-3. Тест перевіряв, чи не падає алгоритм в локальний максимум. Як видно з результату, цього не відбувається.

Розглянемо оптимальні параметри алгоритму

Для дослідження використано випадково згенерований граф на 1000 вершин та 10000 ребер

Дослідження було проведене на кількості розвідників від 1 до 1000 і робочих від 1 до 10000. Зафіксувавши кількість розвідників, оцінимо роботу колонії з різною кількістю робочих з кроком 250. Після перевірки всіх значень робочих на алгоритмі, шляхом повтору пошуку 10 разів для мінімізації впливу випадкових факторів, повторимо цей цикл для всіх значень розвідників з кроком 25.

Для оцінки результату роботи алгоритму використана наступна цільова функція: розмір_кліки * 1000000d / час_роботи. Вибір цільової функції зумовлений тим, щоб розмір знайденої кліки однозначно визначав, наскільки алгоритм був ефективним, а час був другорядним параметром для вибору найкращих за часом параметрів, що знайшли найбільшу кліку.

За результатом роботи програми маємо, що найефективнішою була конфігурація: 51 розвідник, 250 робочих. Повний лог роботи програми можна знайти у папці docs репозиторію.

Загальна закономірність така: надто малі колонії не можуть впоратись з пошуком оптимального розв’язку, у той час як найбільші виконують забагато зайвих операцій, що сповільнює пошук рішення. Крім того, цікаво, що зі зростанням кількості робочих значення цільової функції змінюються хвилеподібно, тому має сенс обирати значення першого з локальних максимумів.

Окремо варто зазначити, що оптимальні параметри колонії значною мірою залежать від розміру графу та кількості ребер. При підборі параметрів для графів з різними параметрами, чисельінсть колонії буде зростати і змінюватись пропорційно до зміни кількості елементів графу.

Висновок

У ході виконання лабораторної роботи було досягнуто основної мети – вивчення основних підходів розробки метаевристичних алгоритмів для типових прикладних задач. Було успішно формалізовано алгоритм вирішення задачі, відповідно до загальної методології, та розроблено його покрокове виконання з достатнім степенем деталізації. Програмна реалізація алгоритму була виконана на обраній мові програмування, що дозволило перевірити його практичну застосовність.

Особливу увагу було приділено методології підбору прийнятних параметрів алгоритму. Через систематичне змінення параметрів алгоритму та використання критерію зупинки, було визначено оптимальні вхідні параметри для конкретної задачі. Цей процес дозволив глибше зрозуміти взаємозв'язок між різними параметрами алгоритму та їх вплив на якість розв’язку задачі.

Важливим висновком є те, що якість розв’язку значною мірою залежить від правильності вибору вхідних параметрів. Була виявлена загальна залежність якості та часу розв’язку від чисельності колонії: надто малі колонії не можуть впоратись з пошуком оптимального розв’язку, у той час як найбільші виконують забагато зайвих операцій, що сповільнює пошук рішення. Також було виявлено, що для різних розмірів графу та кількості ребер оптимальні параметри можуть відрізнятися, що підкреслює необхідність індивідуального підходу до кожної задачі.

Загалом, робота над лабораторною дозволила не тільки розробити та впровадити ефективний алгоритм, але й зрозуміти важливість гнучкого підходу до вибору параметрів для досягнення оптимальних результатів.