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 № 4 з дисципліни
«Проектування алгоритмів»

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

Виконав(ла) *ІП-01 Пашковський Євгеній Сергійович*
(шифр, прізвище, ім'я, по батькові)

Перевірив *Вєчерковська А. С.*
(прізвище, ім'я, по батькові)

Київ 2021

ЗМІСТ

1 МЕТА ЛАБОРАТОРНОЇ РОБОТИ 3

2 ЗАВДАННЯ 4

3 ВИКОНАННЯ 6

3.1 Покроковий алгоритм 6

3.2 Програмна реалізація алгоритму 6

3.2.1 Вихідний код 6

3.2.2 Приклади роботи 15

3.3 Тестування алгоритму 16

ВИСНОВОК 18

КРИТЕРІЇ ОЦІНЮВАННЯ 19

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**
5	**Задача про кліку** (300 вершин, степінь вершини не більше 30, але не менше 2)**.** Клікою в неорієнтованому графі називається підмножина вершин, кожні дві з яких з'єднані ребром графа. Іншими словами, це повний підграф первісного графа. Розмір кліки визначається як число вершин в ній.

	Задача про кліку існує у двох варіантах: у **задачі розпізнавання** потрібно визначити, чи існує в заданому графі G кліка розміру k, тоді як в **обчислювальному варіанті** потрібно знайти в заданому графі G кліку максимального розміру або всі максимальні кліки (такі, що не можна збільшити). Застосування: – біоінформатика; – електротехніка;

Таблиця 2.2 – Варіанти алгоритмів і досліджувані параметри

№	**Алгоритми і досліджувані параметри**
3	**Бджолиний алгоритм:** – кількість ділянок; – кількість бджіл (фуражирів і розвідників).

Таблиця 2.3 – Варіанти задач і алгоритмів

№	**Задачі і алгоритми**
19	Задача про кліку (обчислювальна задача) + Бджолиний алгоритм

3 ВИКОНАННЯ

3.1 Покроковий алгоритм

1) Генерація ділянок для пошуку нектару.

2) Оцінка корисності ділянок.

3) Вибір ділянок для пошуку в їх околиці.

4) Відправка фуражирів.

5) Пошук в околицях джерел нектару.

6) Оновити оцінки корисності ділянок.

7) Відправка бджіл-розвідників.

8) Випадковий пошук (опціонально).

9) Якщо умова зупинки не виконується, то п. 2.

10) Кінець роботи алгоритму.

3.2 Програмна реалізація алгоритму

3.2.1 Вихідний код

```
'use strict';

const Field = require('./Field.js');

class Bee {
  constructor(field) {
    this.field = field;
  }

  moveTo(field) {
    this.field?.bees.splice(this.field?.bees.indexOf(this), 1);
    this.field = field;
    this.field?.bees.push(this);
  }

  moveToNeighbour() {
    const randomIndex = Math.round(
      Math.random() * (this.field.location.length - 1)
    );
    const neighbourLocation = new Array(...this.field.location);
    neighbourLocation[randomIndex] =
      neighbourLocation[randomIndex] === 0 ? 1 : 0;
    const neighbour = new Field(neighbourLocation);
    this.moveTo(neighbour);
  }

  static isBee(obj) {
    return obj instanceof Bee;
  }
```

```
}

module.exports = Bee;
```

```
'use strict';

class Field {
  constructor(location = null, size = 1) {
    this.location = location;
    this.size = size;
    this.nectar = 0;
    this.bees = [];
    this.nearFields = [];
  }
}

module.exports = Field;
```

```
'use strict';

const Bee = require('./Bee.js');

class Scout extends Bee {
  constructor(field) {
    super(field);
  }

  static isScout(obj) {
    return obj instanceof Scout;
  }
}

module.exports = Scout;
```

```
'use strict';

const Bee = require('./Bee.js');
const Scout = require('./Scout.js');
const Field = require('./Field.js');

const percentageWorkers = 0.9;

class Hive {
  bees = {
    workers: [],
    scouts: [],
  };
  constructor(graph, beesNum = 1000, location = null) {
    this.graph = graph;
    for (let i = 0; i < beesNum; i++) {
      if (i <= Math.round(percentageWorkers * beesNum) - 1) {
        this.bees.workers.push(new Bee(location));
      } else {
        this.bees.scouts.push(new Scout(location));
      }
    }
    this.location = location;
  }

  sendScouts(...fields) {
    const newFields = [];
    for (const field of fields) {
      const scout = this.freeScouts[0];
      if (this.freeScouts.length < this.bees.scouts.length * 0.05) {
        const randomLocation = [];
        for (let i = 0; i < field.location.length; i++) {
          randomLocation.push(Math.random() < 0.4 ? 1 : 0);
        }
        scout?.moveTo(new Field(randomLocation));
        continue;
      }
```

```
      scout?.moveTo(field);
    }

    for (const scout of this.bees.scouts) {
      if (!scout.field) break;
      const verteces = this.#getVertecesFromLocation(scout.field.location);
      if (this.graph.subgraph(verteces).isComplete) {
        scout.field.nectar = verteces.length;
        newFields.push(scout.field);
      }
      scout.moveTo(this.location);
    }
    return newFields;
  }

  sendWorkers(...fields) {
    const newFields = [];
    const sumNectar = fields.reduce((acc, el) =>
      typeof acc === 'number' ? acc + el.nectar : acc.nectar + el.nectar
    );
    const possibilities = fields.map(el => el.nectar / sumNectar);

    for (const worker of this.bees.workers) {
      const possibility = Math.random();

      let i = 0;
      let sum = 0;
      while (sum < possibility && i < possibilities.length - 1) {
        sum += possibilities[i];
        i++;
      }

      worker.moveTo(fields[i]);

      for (let i = 0; i < 10; i++) {
        worker.moveToNeighbour();
        const verteces = this.#getVertecesFromLocation(worker.field.location);
        if (
          this.graph.subgraph(verteces).isComplete &&
          verteces.length > fields[i].nectar
        ) {
          newFields.push(worker.field);
        }
      }
    }

    this.bees.workers.forEach(el => el.moveTo(this.location));
    return newFields.concat(fields);
  }

  #getVertecesFromLocation(location) {
    return location
      .map((el, i) => (el !== 0 ? this.graph.verteces[i] : 0))
      .filter(el => el !== 0);
  }

  get freeScouts() {
    return this.bees.scouts.filter(scout => scout.field === this.location);
  }

  get freeWorkers() {
    return this.bees.workers.filter(scout => scout.field === this.location);
  }
}

module.exports = Hive;
```

```
'use strict';

module.exports = (min = 0, max = 1) => Math.random() * (max - min) + min;
```

```
'use strict';
const generateRandom = require('./generateRandom.js');
const Graph = require('./Graph.js');
const Vertex = require('./Vertex.js');
const fs = require('fs');

module.exports = (vertecesNum, minPower, maxPower) => {
  if (minPower > maxPower || maxPower > vertecesNum) {
    throw new Error('Invalid values');
  }

  const graph = new Graph();

  for (let i = 0; i < vertecesNum; i++) {
    const vertex = new Vertex({});
    graph.insert(vertex);
  }

  for (let i = 0; i < vertecesNum; i++) {
    const vertex1 = graph.verteces[i];
    const expectedPower = Math.round(generateRandom(minPower, maxPower));
    while (vertex1.power < expectedPower) {
      const j = Math.round(Math.random() * (vertecesNum - 1));
      const vertex2 = graph.verteces[j];

      if (vertex2.power < maxPower) {
        graph.connect(i, j);
        graph.connect(j, i);
      }
    }
  }

  fs.writeFileSync('./graph.log', JSON.stringify(graph.matrix.arr));
  return graph;
};
```

```
'use strict';

class Matrix {
  constructor(arr = []) {
    this.arr = this.#alignRows(arr);
  }

  static isMatrix(obj) {
    return obj instanceof Matrix;
  }

  #alignRows(arr) {
    const res = [];
    const maxRowLength = Math.max(...arr.map(el => el.length));

    for (const row of arr) {
      while (row.length !== maxRowLength) {
        row.push(0);
      }
      res.push(row);
    }
    return res;
  }

  getElement(y, x) {
    return this.arr[y][x];
  }

  setElement(y, x, data) {
    this.arr[y][x] = data;
    return this;
  }

  pushRow(row = []) {
    this.arr.push(row || 0);
```

```
    this.arr = this.#alignRows(this.arr);
    return this;
  }

  removeRow(index) {
    this.arr.splice(index, 1);
    return this;
  }

  removeCol(index) {
    for (const row of this.arr) {
      row.splice(index, 1);
    }
    return this;
  }

  pushCol(col = []) {
    for (const i in this.arr) {
      this.arr[i].push(col[i] || 0);
    }
    return this;
  }

  find(item) {
    for (const i in this.arr) {
      for (const j in this.arr[i]) {
        if (this.arr[i][j] === item) return { y: i, x: j };
      }
    }
    return false;
  }

  copy() {
    const res = [];
    for (const row of this.arr) {
      res.push(new Array(...row));
    }
    return new Matrix(res);
  }

  isEqual(matrix) {
    let res = true;
    for (const i in this.arr) {
      for (const j in matrix.arr) {
        if (this.arr[i][j] !== matrix.arr[i][j]) {
          res = false;
          break;
        }
      }
    }
    return res;
  }

  add(matrix) {
    if (!Matrix.isMatrix(matrix)) {
      throw new TypeError('Must be an instance of Matrix');
    }

    if (this.xLength !== matrix.xLength || this.yLength !== matrix.yLength) {
      throw new Error('Impossible to add matrixes with different sizes');
    }

    for (const i in this.arr) {
      for (const j in this.arr[i]) {
        this.arr[i][j] += matrix.arr[i][j];
      }
    }
    return this;
  }

  mult(matrix) {
```

```
    if (!Matrix.isMatrix(matrix)) {
      throw new TypeError('Must be an instance of Matrix');
    }

    if (this.xLength !== matrix.yLength || this.yLength !== matrix.xLength) {
      throw new Error(
        'Impossible to multiply matrixes with unappropriate sizes'
      );
    }

    for (const i in this.arr) {
      for (const j in this.arr[i]) {
        this.arr[i][j] = this.arr[i][0] * matrix.arr[0][j];
        for (const k in this.xLength) {
          this.arr[i][j] += this.arr[i][k] * matrix.arr[k][j];
        }
      }
    }

    return this;
  }

  transpone() {
    const res = [];
    for (const j in this.arr) {
      res.push([]);
      for (const i in this.arr[j]) {
        res[j][i] = this.arr[i][j];
      }
    }
    return new Matrix(res);
  }

  toString() {
    let res = '';
    for (const i in this.arr) {
      for (const j in this.arr[i]) {
        res += this.arr[i][j] === null ? '0' : String(this.arr[i][j]);
      }
    }
    return res;
  }

  get xLength() {
    return this.arr[0].length;
  }

  get yLength() {
    return this.arr.length;
  }

  get rows() {
    return this.arr;
  }

  get cols() {
    return this.transpone().arr;
  }

  set cols(arr) {
    if (!Array.isArray(arr)) {
      throw new TypeError('Should be an array');
    }
    this.arr = new Matrix(arr).transpone().arr;
  }

  set rows(arr) {
    if (!Array.isArray(arr)) {
      throw new TypeError('Should be an array');
    }
    this.arr = arr;
```

```
  }
}

module.exports = Matrix;
```

```
'use strict';

class Vertex {
  constructor(data = null) {
    this.data = data;
    this.links = new Set();
  }

  linkTo(...verteces) {
    verteces.forEach(this.links.add, this.links);
  }

  unlinkFrom(...verteces) {
    verteces.forEach(this.links.delete, this.links);
  }

  static isVertex(obj) {
    return obj instanceof Vertex;
  }

  get power() {
    return this.links.size;
  }
}

module.exports = Vertex;
```

```
'use strict';

const Vertex = require('./Vertex.js');
const Matrix = require('./Matrix.js');

class Graph {
  constructor(verteces = [], matrix = new Matrix()) {
    if (verteces.some(el => !Vertex.isVertex(el))) {
      throw new TypeError('Must be instance of Vertex');
    }
    if (!Matrix.isMatrix(matrix)) {
      throw new TypeError('Must be instance of Matrix');
    }
    this.verteces = verteces;
    this.matrix = matrix;
    this.#linkVerteces(verteces, matrix);
  }

  static isGraph(obj) {
    return obj instanceof Graph;
  }

  #linkVerteces(verteces, matrix) {
    for (const i in verteces) {
      const vertex = verteces[i];
      for (const j in matrix.cols) {
        if (matrix.getElement(i, j) !== 0) {
          vertex.linkTo(verteces[j]);
        } else {
          vertex.unlinkFrom(verteces[j]);
        }
      }
    }
  }

  isConnected(v1, v2) {
    return +this.matrix.getElement(v1, v2) !== 0;
  }
```

```
  connect(v1, v2, weight = 1) {
    this.matrix.setElement(v1, v2, weight);
    this.verteces[v1].linkTo(this.verteces[v2]);
  }

  disconnect(v1, v2) {
    this.matrix.setElement(v1, v2, 0);
    this.verteces[v1].unlinkFrom(this.verteces[v2]);
  }

  insert(...vertexes) {
    for (const vertex of vertexes) {
      this.#add(vertex);
    }
  }

  remove(obj) {
    if (this.verteces.includes(obj)) {
      this.#delete(obj);
    }
  }

  subgraph(verteces = this.verteces) {
    const newVerteces = verteces.map(el => new Vertex(el.data));
    return new Graph(newVerteces, this.#matrixFromVerteces(verteces));
  }

  #matrixFromVerteces(verteces) {
    const newMatrix = new Matrix();
    for (const vertex of verteces) {
      const i = this.verteces.indexOf(vertex);
      const row = [];
      for (let j = 0; j < verteces.length; j++) {
        row.push(0);
      }
      const linkedVerteces = vertex.links.values();
      for (const linkedVertex of linkedVerteces) {
        const index = verteces.indexOf(linkedVertex);
        if (index > -1) {
          const j = this.verteces.indexOf(linkedVertex);
          row[index] = this.matrix.getElement(i, j);
        }
      }
      newMatrix.pushRow(row.map(el => el || 0));
    }
    return newMatrix;
  }

  #add(obj) {
    this.verteces.push(Vertex.isVertex(obj) ? obj : new Vertex(obj));
    this.matrix.pushRow().pushCol();
  }

  #delete(obj) {
    const index = this.verteces.indexOf(obj);
    this.verteces.splice(index, 1);
    this.matrix.removeRow(index).removeCol(index);
  }

  get isComplete() {
    for (let i = 0; i < this.verteces.length; i++) {
      for (let j = 0; j < this.verteces.length; j++) {
        if (i !== j && !this.matrix.rows[i][j]) {
          return false;
        }
      }
    }
    return true;
  }

  get connections() {
```

```
    let res = 0;
    for (let i = 0; i < this.verteces.length; i++) {
      for (let j = 0; j < this.verteces.length; j++) {
        if (this.matrix.rows[i][j]) {
          res++;
        }
      }
    }
    return res;
  }
}

module.exports = Graph;
```

```
'use strict';

const Field = require('./Field.js');
const generateGraph = require('./generateGraph.js');
const Hive = require('./Hive.js');

const vertecesNum = 300;
const minPower = 2;
const maxPower = 30;
const iterations = 40;

class Lab4 {
  start() {
    const graph = generateGraph(vertecesNum, minPower, maxPower);
    const complete2subgraphs = [];
    for (let i = 0; i < vertecesNum; i++) {
      for (let j = 0; j < vertecesNum; j++) {
        const newLocation = [];
        for (let k = 0; k < vertecesNum; k++) {
          newLocation.push(0);
        }
        if (
          j > i &&
          graph.subgraph([graph.verteces[i], graph.verteces[j]]).isComplete
        ) {
          newLocation[i] = 1;
          newLocation[j] = 1;
          complete2subgraphs.push(newLocation);
        }
      }
    }

    let fields = complete2subgraphs.map(el => new Field(el));

    const hive = new Hive(graph);
    fields = hive.sendScouts(...fields);

    let maxField = fields[0];
    for (let i = 0; i < iterations; i++) {
      if (i % 20 === 0) {
        console.log(`===${i}===`);
        for (const field of fields) {
          if (field.nectar > maxField.nectar) {
            maxField = field;
          }
        }
        console.log(`Max "clicka": ${maxField.nectar}`);
      }
      fields = hive.sendWorkers(...fields);
      fields = hive.sendScouts(...fields);
    }
    console.log(`Max "clicka": ${maxField.nectar}`);
    console.log(
      graph.subgraph(
        maxField.location
          .map((el, i) => (el === 0 ? 0 : graph.verteces[i]))
          .filter(el => el !== 0)
```

```
            )
          );
        }
      }

      module.exports = Lab4;
```

3.2.2 Приклади роботи

На рисунках 3.1 і 3.2 показані приклади роботи програми.

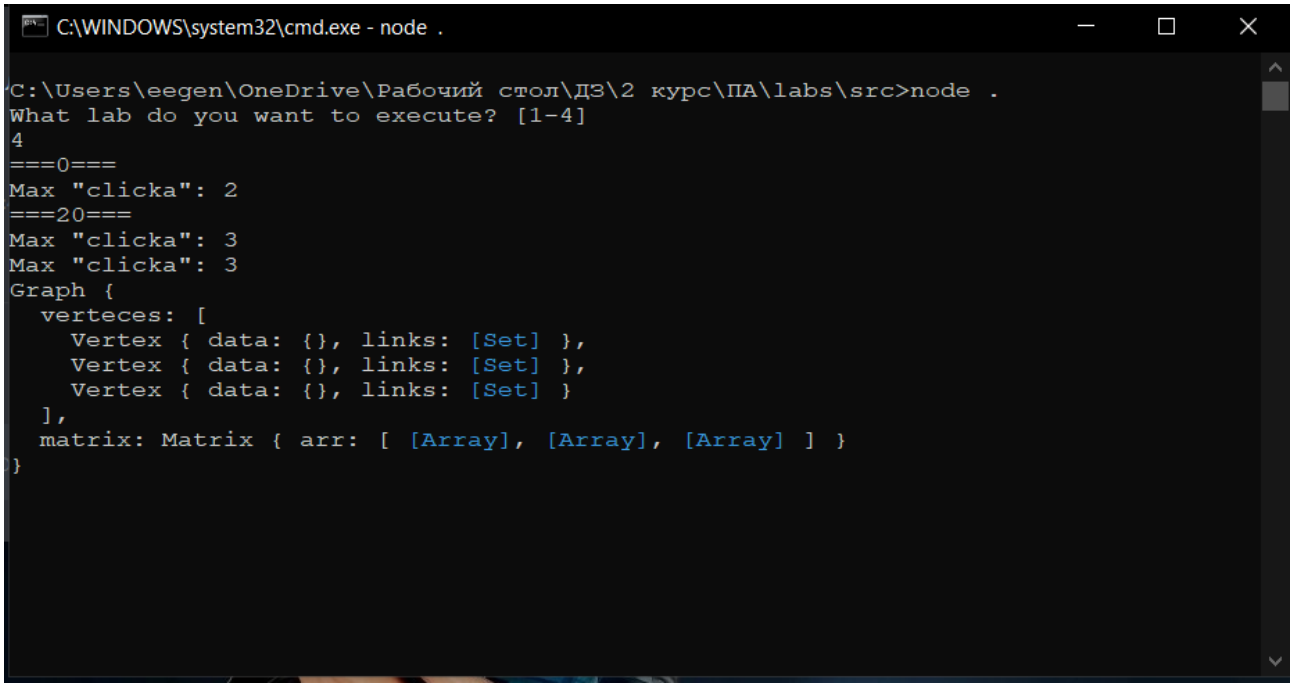

Рисунок 3.1 Виконання програми

```
C:\Users\eegen\OneDrive\Рабочий стол\ДЗ\2 курс\ПА\labs\src>node .
What lab do you want to execute? [1-4]
4
Test 1: 10 bees
===0===
===20===
===40===
===60===
===80===
Max "clicka": 4
Graph {
  verteces: [
    Vertex { data: {}, links: [Set] },
    Vertex { data: {}, links: [Set] },
    Vertex { data: {}, links: [Set] },
    Vertex { data: {}, links: [Set] }
  ],
  matrix: Matrix { arr: [ [Array], [Array], [Array], [Array] ] }
}
Test 2: 20 bees
===0===
===20===
===40===
===60===
```

Рисунок 3.2 Тестування для багатьох значень бджіл

3.3 Тестування алгоритму

Фіксуємо один випадковий граф (а відповідно – всі його вершини і зв’язки), програмно змінюємо кількість бджіл.

Кількість бджіл	Максимальна кліка
100	4
200	4
300	4
400	4
500	4
600	4
700	4
800	4
900	4
1000	4

Рисунок 3.3 Графік залежності розв’язку від кількості бджіл

Під час тестування виявилося, що при значній кількості ітерацій (100+) та відносно невеликій кількості вершин (300), значення кількості ділянок не впливає на розв’язок, оскільки всі інші ділянки знаходяться внаслідок випадкового пошуку.

ВИСНОВОК

В рамках даної лабораторної роботи я розробив програму для пошуку максимальної кліки (такого підграфа, всі вершини якого з’єднані між собою) за допомогою бджолиного алгоритму. Дослідив залежність якості розв’язку від кількості бджіл та кількості ділянок. При відносно великій кількості ітерацій та відносно невеликій кількості вершин графа кількість бджіл та ділянок майже не впливає на якість отриманого розв’язку, проте при недостатній кількості ітерацій кількість бджіл та доступних ділянок є визначальними факторами для якості розв’язку.

КРИТЕРІЇ ОЦІНЮВАННЯ

При здачі лабораторної роботи до 26.11.2021 включно максимальний бал дорівнює – 5. Після 26.11.2021 максимальний бал дорівнює – 1.

Критерії оцінювання у відсотках від максимального балу:

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