

**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 № 1 з дисципліни  
«Проектування алгоритмів»

**«Неінформативний, інформативний та локальний пошук»**

**Виконав(ла)**

ІІ-01 Пашковський Євгеній Сергійович  
(шифр, прізвище, ім'я, по батькові)

**Перевірив**

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

## ЗМІСТ

1. Мета лабораторної роботи .....	3
2. Завдання .....	4
3. Виконання .....	7
3.1. Псевдокод алгоритмів.....	7
3.2. Програмна реалізація .....	8
3.2.1. Вихідний код .....	8
3.2.2. Приклади роботи.....	18
3.3. Дослідження алгоритмів.....	19
4. Висновок.....	26

## 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 **АПІ**, 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. 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 (512 Мб)

### **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 впорядкування їх по номерах, бажано зробивши якомога менше переміщень.

– **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 А може стояти на одній лінії з ферзем В, проте між ними стоїть ферзь С; тому А не б'є В).

– **F2** – кількість пар ферзів, які б'ють один одного без урахування видимості.

– **H1** – кількість фішок, які не стоять на своїх місцях.

– **H2** – Манхетенська 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 – Варіант алгоритмів

№	Задача	АНП	АП	АЛП	Func
1	8-puzzle	BFS	RBFS		H1
9					

## 3 ВИКОНАННЯ

### 3.1 Псевдокод алгоритмів

#### BFS:

```
function solveBFS():
    rootNode = initNode()
    tree = initTree(rootNode)
    open = queue(rootNode)
    repeat:
        parent = open.next
        if (parent.state === goalState) return parent
        newVerteces <- tree.expand(parent, ...verteces)
        open.push(newVerteces)
        if (open.length === 0) return false
```

#### RBFS:

```
function solveRBFS:
    rootNode = initNode()
    rootNode.g = 0
    rootNode.f = rootNode.state.h
    tree = initTree(rootNode)
    open = priorityQueue(rootNode, 'inverse')
    function RBFS(parent, fLimit):
        newVerteces <- tree.expand(parent, ...verteces)
        newVerteces.each.g = parent.g + 1,
        f = max(parent.g + vertex.h, parent.f)
        if (open.length === 0) return [false, false]
        repeat:
            best = open.lowest
            if (best.f > fLimit) {
                return [false, best.f];
            }
            alternative = open.nextLowest
            [result, best.f] = RBFS(best, min(fLimint, alternative))
            if(result) return [result, false]
```

## 3.2 Програмна реалізація

### 3.2.1 Вихідний код

```
'use strict';

const State = require('./State.js');
const Matrix = require('./Matrix.js');
const Tree = require('./Tree.js');
const shuffle = require('./shuffle.js');
const Vertex = require('./Vertex.js');
const PriorityQueue = require('./PriorityQueue.js');

class Puzzle {
  constructor(initialState = false) {
    this.state = initialState
    ? this.#generateState()
    : new State(new Matrix(initialState));
  }

  #generateState() {
    const ids = [1, 2, 3, 4, 5, 6, 7, 8, null];
    const shuffledIds = shuffle(ids);
    const arr = new Matrix([
      shuffledIds.slice(0, 3),
      shuffledIds.slice(3, 6),
      shuffledIds.slice(6, 9),
    ]);
    return new State(arr);
  }

  findSolutionBFS(logger) {
    setTimeout(() => {
      logger.send({ name: 'error', data: 'Time limit exeeded' });
      process.exit();
    }, 1000 * 60 * 30);
    return new Promise((resolve, reject) => {
      const rootNode = new Vertex({
        state: this.state,
        chosenChange: null,
        parent: null,
        depth: 0,
      });
      const tree = new Tree(rootNode);
```



```

const open = [rootNode];
const closed = [];
while (true) {
  const parent = open.shift();
  const parentState = parent.data.state;

  if (parentState.matrix.isEqual(new Matrix(Puzzle.solutionTemplate))) {
    console.log(parentState.printState());
    resolve(parent);
    break;
  }

  const newVerteces = [];
  for (const possibleChange of parentState.possibleChanges) {
    const newState = parentState.changeState(possibleChange);
    if (!closed.includes(newState.matrix.toString())) {
      newVerteces.push(
        new Vertex({
          state: newState,
          chosenChange: possibleChange,
          parent,
          depth: parent.data.depth + 1,
        })
      );
    }
  }

  if (newVerteces.length !== 0) {
    tree.expand(parent, ...newVerteces);
    open.push(...newVerteces);
  }
  closed.push(parentState.matrix.toString());

  if (open.length === 0) {
    reject('Expandable is empty');
  }

  if (logger) {
    if (process.resourceUsage().maxRSS / 1000 > 512) {
      logger.send({ name: 'error', data: 'Maximum heap size exceeded' });
      process.exit();
    }
    logger.send({
      name: 'data',
      data: {

```

```

        memory: process.resourceUsage().maxRSS,
        depth: parent.data.depth,
    },
    });
}
}
});
}

findSolutionRBFS(logger) {
    setTimeout(() => {
        logger.send({ name: 'error', data: 'Time limit exceeded' });
        process.exit();
    }, 1000 * 60 * 30);
    return new Promise((resolve, reject) => {
        const rootNode = new Vertex({
            state: this.state,
            chosenChange: null,
            parent: null,
            depth: 0,
            f: this.state.h,
        });
        const tree = new Tree(rootNode);
        const open = new PriorityQueue(rootNode, Infinity, true);

        const rbfs = (parent, fLimit) => {
            if (!tree.verteces.includes(parent)) {
                return [false, false];
            }

            const parentState = parent.data.state;
            if (parentState.matrix.isEqual(new Matrix(Puzzle.solutionTemplate))) {
                return [parent, false];
            }

            for (const possibleChange of parentState.possibleChanges) {
                const newState = parentState.changeState(possibleChange);
                const newVertex = new Vertex({
                    state: newState,
                    chosenChange: possibleChange,
                    parent,
                    depth: parent.data.depth + 1,
                    f: Math.max(parent.data.depth + 1 + newState.h, parent.data.f),
                });

                tree.expand(parent, newVertex);
            }
        };
    });
}

```

```

    open.push(newVertex, newVertex.data.f);
  }

  if (open.length === 0) reject('No open verteces left');

  while (true) {
    const best = open.pop();
    if (best.data.f > fLimit) {
      return [false, best.data.f];
    }

    if (logger) {
      if (process.resourceUsage().maxRSS / 1000 > 512) {
        logger.send({ name: 'error', data: 'Maximum heap size exeeded' });
        process.exit();
      }

      logger.send({
        name: 'data',
        data: {
          memory: process.resourceUsage().maxRSS,
          depth: parent.data.depth,
        },
      });
    }

    let pointer = open.head;
    while (pointer && pointer.priority <= best.data.f) {
      pointer = pointer.nextNode;
    }
    const alternative = pointer?.priority || open.head.data.data.f;

    let result = false;
    [result, best.data.f] = rbfs(best, Math.min(fLimit, alternative));
    if (best.data.f) {
      open.push(best, best.data.f);
      for (const child of best.links.values()) {
        tree.cut(child);
      }
    }
    if (result) {
      return [result, false];
    }
  }
};

```

```

        resolve(rbfs(rootNode, Infinity));
    });
}

static solutionTemplate = State.GoalState;
}

module.exports = Puzzle;

```

```

'use strict';

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

class State {
  constructor(matrix) {
    this.#checkMatrix(matrix);
    this.matrix = matrix;
  }

  #checkMatrix(matrix) {
    if (
      Matrix.isMatrix(matrix) &&
      matrix.yLength !== 3 &&
      matrix.xLength !== 3 &&
      matrix.arr.some(el => !Array.isArray(el))
    ) {
      throw new Error('Unappropriate state structure');
    }
  }

  printState() {
    let res = '';
    for (const row of this.matrix.rows) {
      const newRow = row.slice(0);

      if (newRow.includes(null)) {
        newRow[newRow.indexOf(null)] = ' ';
      }
      res += newRow.join(' ') + '\n';
    }
    return res;
  }

  changeState(dir) {
    let { x, y } = this.matrix.find(null);
    x = Number(x);
    y = Number(y);

    const arr = this.matrix.copy().arr;

    let buf;
    if (dir === 't') {
      buf = arr[y + 1][x];
      arr[y + 1][x] = null;
    } else if (dir === 'b') {
      buf = arr[y - 1][x];
      arr[y - 1][x] = null;
    } else if (dir === 'l') {
      buf = arr[y][x + 1];
      arr[y][x + 1] = null;
    } else if (dir === 'r') {
      buf = arr[y][x - 1];
      arr[y][x - 1] = null;
    }
    arr[y][x] = buf;
  }
}

```

```

    const newMatrix = new Matrix(arr);
    return new State(newMatrix);
  }

  get possibleChanges() {
    let { x, y } = this.matrix.find(null);
    x = Number(x);
    y = Number(y);
    return State.possible[y][x];
  }

  get h() {
    let res = 0;

    for (const i in this.matrix.arr) {
      for (const j in this.matrix.arr[i]) {
        if (State.GoalState[i][j] !== this.matrix.arr[i][j]) res++;
      }
    }

    return res;
  }

  static possible = [
    [
      ['t', 'l'],
      ['t', 'l', 'r'],
      ['t', 'r'],
    ],
    [
      ['t', 'b', 'l'],
      ['t', 'b', 'l', 'r'],
      ['t', 'b', 'r'],
    ],
    [
      ['b', 'l'],
      ['b', 'l', 'r'],
      ['b', 'r'],
    ],
  ],
];

  static GoalState = [
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, null],
  ];
}

module.exports = State;

```

```

'use strict';

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

class Tree extends Graph {
  constructor(obj) {
    const vertex = Vertex.isVertex(obj) ? obj : new Vertex(obj);
    super([vertex], new Matrix([[0]]));
  }

  expand(vertex, ...verteces) {
    this.insert(verteces);
    const v1 = this.verteces.indexOf(vertex);
    for (const v of verteces) {
      const v2 = this.verteces.indexOf(v);
      this.connect(v1, v2);
    }
  }
}

```

```

    }

    cut(vertex) {
      const parent = vertex.data.parent;
      if (Vertex.isVertex(parent)) {
        const v1 = this.verteces.indexOf(parent);
        const v2 = this.verteces.indexOf(vertex);
        if (vertex.links.size !== 0) {
          for (const child of vertex.links.values()) {
            this.cut(child);
          }
        }
        this.disconnect(v1, v2);
        this.remove(vertex);
      } else {
        throw new Error('Vertex must have parent in data object');
      }

      return vertex;
    }
  }
}

module.exports = Tree;

```

```

'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(null);
      }
      res.push(row);
    }
    return res;
  }

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

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

  pushRow(row = []) {
    this.arr.push(row);
    this.arr = this.#alignRows(this.arr);
    return this;
  }

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

```

```

}

removeCol(index) {
  for (const i in this.arr) {
    this.arr[i].splice(index, 1);
  }
  return this;
}

pushCol(col = []) {
  for (const i in this.arr) {
    this.arr[i].push(col[i] || null);
  }
  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 i in this.arr) {
    res.push(new Array(...this.arr[i]));
  }
  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) {
    this.arr = new Matrix(arr).transpone().arr;
}

set rows(arr) {
    this.arr = arr;
}
}

module.exports = Matrix;

```

```

'use strict';
const Node = require('./Node.js');
const Queue = require('./Queue.js');

```



```

class PriorityQueue extends Queue {
  constructor(data, priority = 0, inverseDirection = false) {
    super(data);
    if (this.head) this.head.priority = priority;
    this.direction = inverseDirection ? -1 : 1;
  }

  push(data, priority) {
    const node = new Node(data, priority);

    if (this.head.priority * this.direction < priority * this.direction) {
      node.nextNode = this.head;
      this.head = node;
    } else if (
      this.lastNode.priority * this.direction >
      priority * this.direction
    ) {
      this.lastNode.nextNode = node;
      this.lastNode = node;
    } else {
      let pointer = this.head;
      while (
        pointer.nextNode &&
        pointer.nextNode.priority * this.direction >= priority * this.direction
      ) {
        pointer = pointer.nextNode;
      }

      const buf = pointer.nextNode;
      pointer.nextNode = node;
      node.nextNode = buf;
    }
  }
}

module.exports = PriorityQueue;

```

```

'use strict';

const shuffle = array => array.sort(() => Math.random() - 0.5);

module.exports = shuffle;

```

```

'use strict';

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

  linkTo(...verteces) {
    for (const item of verteces) {
      this.links.add(item);
    }
  }

  unlinkFrom(...verteces) {
    for (const item of verteces) {
      this.links.delete(item);
    }
  }

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

module.exports = Vertex;

```

### 3.2.2 Приклади роботи

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

```
Initial state:  
  
1  2  
4 5 3  
7 8 6  
  
Memory: 35.548Mb  
Depth: 3  
  
Solution found!  
Solution: 0.998ms  
  
l->t->t  
█
```

Рисунок 3.1 – Алгоритм BFS


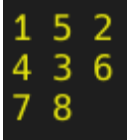

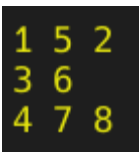


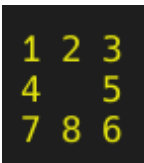
```
Initial state:  
  
1  2  
4 5 3  
7 8 6  
  
Memory: 35.592Mb  
Depth: 2  
  
Solution found!  
Solution: 1.124ms  
  
l->t->t  
█
```









Рисунок 3.2 – Алгоритм RBFS

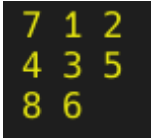

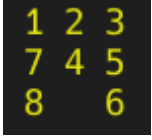
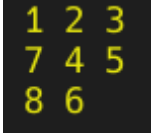
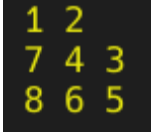
### 3.3 Дослідження алгоритмів

В таблиці 3.1 наведені характеристики оцінювання алгоритму BFS, задачі 8-puzzle для 20 початкових станів.

Таблиця 3.1 – Характеристики оцінювання алгоритму BFS


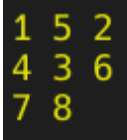

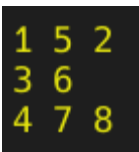


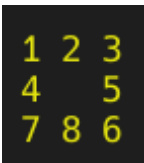
Початкові стани	Ітерації	Всього станів	Всього станів у пом'яті
Стан 1 	15	25	25
Стан 2 	63	108	108
Стан 3 	277	433	433
Стан 4 	1380	2191	2191
Стан 5 	5497	8633	8633
Стан 6 	3	7	7
Стан 7 	9	17	17

Стан 8 	17	31	31
Стан 9 	21	39	39
Стан 10 	89	155	155
Стан 11 	122	218	218
Стан 12 	426	714	714
Стан 13 	724	1172	1172
Стан 14 	1096	1814	1814
Стан 15 	2596	4132	4132

Стан 16 	4378	7081	7081
Стан 17 	1	4	4
Стан 18 	59	103	103
Стан 19 	89	155	155
Стан 20 	174	314	314

В таблиці 3.2 наведені характеристики оцінювання алгоритму RBFS, задачі 8-puzzle для 20 початкових станів.

Таблиця 3.2 – Характеристики оцінювання алгоритму RBFS

Початкові стани	Ітерації	Всього станів	Всього станів у пом'яті
Стан 1 	3	9	9
Стан 2 	13	36	36
Стан 3 	64	174	134
Стан 4 	593	1576	1228
Стан 5 	4367	11515	8626
Стан 6 	2	6	6
Стан 7 	2	8	8

Стан 8 <div>1 2 3 4 5 7 8 6</div>	3	11	11
Стан 9 <div>1 2 3 7 4 5 8 6</div>	4	13	13
Стан 10 <div>1 2 3 7 4 5 8 6</div>	9	25	25
Стан 11 <div>1 2 3 7 4 8 6 5</div>	20	56	50
Стан 12 <div>1 2 7 4 3 8 6 5</div>	44	119	104
Стан 13 <div>1 2 7 4 3 8 6 5</div>	70	186	161
Стан 14 <div>7 1 2 4 3 8 6 5</div>	204	521	481
Стан 15 <div>7 1 2 4 3 8 6 5</div>	1122	2911	2342



Стан 16 <div>7 1 2 4 3 5 8 6</div>	1	4	4
Стан 17 <div>1 2 3 4 5 7 8 6</div>	6	18	18
Стан 18 <div>1 2 3 7 4 5 8 6</div>	9	25	25
Стан 19 <div>1 2 3 7 4 5 8 6</div>	21	58	52
Стан 20 <div>1 2 7 4 3 8 6 5</div>	24	58	51

## ВИСНОВОК

При виконанні даної лабораторної роботи було розглянуто два алгоритми пошуку – неінформативний Breadth First Search (пошук вшир) та інформативний Recursive Best First Search (рекурсивний пошук по першому найкращому співпадінню) на прикладі задачі 8-puzzle. Було проведено дослідження швидкодії та затрат оперативної пам'яті кожного з вказаних алгоритмів для 20 початкових станів. Це дослідження вказує на те, що інформативний алгоритм RBFS є більш ефективним та швидшим ніж неінформативний BFS, проте варто пам'ятати, що успіх та ефективність інформативних алгоритмів на пряму пов'язана з доцільністю використання евристичної функції  $h$  в умовах конкретної задачі. У даному варіанті евристична  $H1$  (кількість фішок, що не стоять на своїх місцях) допустима, проте в деяких випадках може сповільнювати роботу алгоритма, що «перескакує» з однієї вітки дерева на іншу з високою частотою, адже не в змозі вибрати кращу одразу.

## КРИТЕРІЇ ОЦІНЮВАННЯ

За умови здачі протоколу лабораторної роботи до 02.10.2021 включно максимальний бал дорівнює – 5. Після 02.10.2021 максимальний бал дорівнює – 1.

Критерії оцінювання у відсотках від максимального балу:

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