

НУЛП, ІКНІ, САПР		Тема	оцінка	підпис
КН-414	19	АЛГОРИТМ ПОБУДОВИ ДЕРЕВ		
Михайлівський Я. О.				
№ залікової: 1708315				
Дискретні моделі в САПР			Викладач: к.т.н., асистент Кривий Р.З.	

Мета:

Вивчення алгоритмів рішення задач побудови остових дерев.

Завдання: Написати програму для побудови мінімального та максимального покриваючого дерева.

Алгоритм Борувки.

Теоретичні відомості:

Максимальне остове дерево.

Даний зважений неорієнтований граф з вершинами і ребрами. Потрібно знайти таке піддерево цього графа, яке б з'єднувало всі його вершини, і при цьому мало найбільшу можливу вагою (тобто сумою ваг ребер). Таке піддерево називається максимальним остовим деревом.

У природному постановці ця задача звучить наступним чином: є міст, і для кожної пари відома вартість з'єднання їх дорогою (або відомо, що з'єднати їх не можна). Потрібно з'єднати всі міста так, щоб можна було доїхати з будь-якого міста в інший, а при цьому вартість прокладання доріг була б максимальною. Сам алгоритм має дуже простий вигляд. Шуканий максимальний кістяк будується поступово, додаванням до нього ребер по одному. Спочатку остов покладається складається з єдиної вершини (її можна вибрати довільно). Потім вибирається ребро максимальної ваги, що виходить з цієї вершини, і додається в максимальне остове дерево. Після цього остов містить уже дві вершини, і тепер шукається і додається ребро максимальної ваги, що має один кінець в одній з двох обраних вершин, а інший - навпаки, у всіх інших, крім цих двох. І так далі, тобто щоразу шукається максимальне по вазі ребро, один кінець якого - вже взята в остов вершина, а інший кінець - ще не взята, і це ребро додається в остов (якщо таких ребер кілька, можна взяти будь-яке). Цей процес повторюється до тих пір, поки остов не стане містити всі вершини (або, що те ж саме, ребро). У результаті буде побудований остов, що є максимальним. Якщо граф був спочатку не зв'язний, то остов знайдений не буде (кількість вибраних ребер залишиться менше).

Алгоритм Борувки.

Це алгоритм знаходження мінімального остового дерева в графі. Вперше був опублікований в 1926 році Отакаром Борувкой, як метод знаходження оптимальної електричної мережі в Моравії. Робота алгоритму складається з декількох ітерацій, кожна з яких полягає в послідовному додаванні ребер до остового лісу графа, до тих пір, поки ліс не перетвориться на дерево, тобто, ліс, що складається з однієї компоненти зв'язності. У псевдокоді, алгоритм можна описати так: Спочатку, нехай T - порожня множина ребер (представляє собою остовий ліс, до якого кожна вершина входить в якості окремого дерева). Поки T не є деревом (поки число ребер у T менше, ніж $V-1$, де V - кількість вершин у графі): Для кожної компоненти зв'язності (тобто, дерева в остовому лісі) в підпункті з ребрами T , знайдемо ребро найменшої ваги, що зв'язує цю компоненту з деякої іншої компонентою зв'язності. (Передбачається, що ваги ребер різні, або як-то додатково впорядковані так, щоб завжди можна було знайти єдине ребро з мінімальною вагою). Додамо всі знайдені ребра в множину T . Отримана множина ребер T є мінімальним остовим деревом вхідного графа.

Програмна реалізація:

```
public class Boruvka {
    final static int N = 7;

    public static void main(String[] args) {

        Graph global = new Graph();
        Graph forest = new Graph();

        char ii=' ', jj=' ';
        for (int i=0;i<N;i++){
            global.L.add(new Leaf(Transformer.numToUpperCase(i)));
        }

        try(FileReader reader = new
FileReader("E:\\Laby\\Yaroslav\\src\\com.lab\\one\\lab1"))
        {
            int c;
            int num=0;
            while((c=reader.read())!=-1){

                if ((char)c=='\n') {
                    global.addEdge(new Edge(num,new Leaf(ii),new Leaf(jj)));
                    num=0;
                    continue;
                }
            }
        }
    }
}
```

```

        }
        if (c >= 65) {
            ii=(char)c;
            while((c = reader.read())<65);
            jj=(char)c;
        }
        else if
(Character.getNumericValue(c)>=0&&Character.getNumericValue(c)<=9){
            num *= 10;
            num += Character.getNumericValue(c);
        }
    }
}
catch(IOException ex){

    System.out.println(ex.getMessage());
}

forest.becomeForest(global);
for (Edge e:forest.T) {
    System.out.println(e.u.name+"-"+e.v.name+" "+e.weight);
}
}

```

```

private static class Graph {
    TreeSet<Edge> T = new TreeSet<>();
    TreeSet<Leaf> L = new TreeSet<>();

    public void addEdge(Edge e){
        T.add(e);
        L.add(e.u);
        L.add(e.v);
    }

    public void addTree(LocalTree b){
        T.addAll(b.T);
        L.addAll(b.L);
    }

    public void becomeForest(Graph G) {
        this.T.add(G.T.first());
        this.L.add(G.T.first().u);
    }
}

```

```

this.L.add(G.T.first().v);
LinkedList<LocalTree> S = new LinkedList<>();

while (T.size() < N - 1) {
    for (Edge e:G.T) {
        if(S.size()>0){
            for (LocalTree b:S) {
                if (b.L.contains(e.u)&&this.L.contains(e.v)||
                    this.L.contains(e.u)&&b.L.contains(e.v)){
                    this.addTree(b);
                    this.addEdge(e);
                    S.remove(b);
                }
            }
        }
        if (L.contains(e.u)!=L.contains(e.v)){
            this.addEdge(e);
        }
        else if (L.contains(e.u)&&L.contains(e.v)) {
            continue;
        }
        else if(S.size()>0){
            for (LocalTree b:S) {
                if (b.L.contains(e.u)!=b.L.contains(e.v)){
                    b.addEdge(e);
                    break;
                }
                else if (L.contains(e.u)&&L.contains(e.v)) {
                    continue;
                }
            }
        }
        else {
            S.add(new LocalTree(e));
        }
    }
}
}

```

```

private static class LocalTree {
    TreeSet<Edge> T = new TreeSet<>();
    TreeSet<Leaf> L = new TreeSet<>();
}

```

```

LocalTree(Edge e){
    T.add(e);
    L.add(e.u);
    L.add(e.v);
}

public void addEdge(Edge e){
    T.add(e);
    L.add(e.u);
    L.add(e.v);
}
}

private static class Edge implements Comparable<Edge> {
    int weight;
    Leaf v;
    Leaf u;
    Edge(int weight, Leaf v, Leaf u){
        this.weight=weight;
        this.v=v;
        this.u=u;
    }

    @Override
    public int compareTo(Edge o) {
        return this.weight-o.weight;
    }
}

public static class Leaf implements Comparable<Leaf> {
    private char name;

    Leaf(char da){
        name=da;
    }

    @Override
    public int compareTo(Leaf o) {
        return this.name-o.name;
    }
}

private static class Transformer {
    public static char numToUpperLetter(int num) {
        return (char)(num+65);
    }
}

```

```
}  
}
```

Запускаємо в режимі дебагу, в папці з файлом запуску має знаходитися файл з параметрами.

Результати роботи програми:

Вхідні данні:

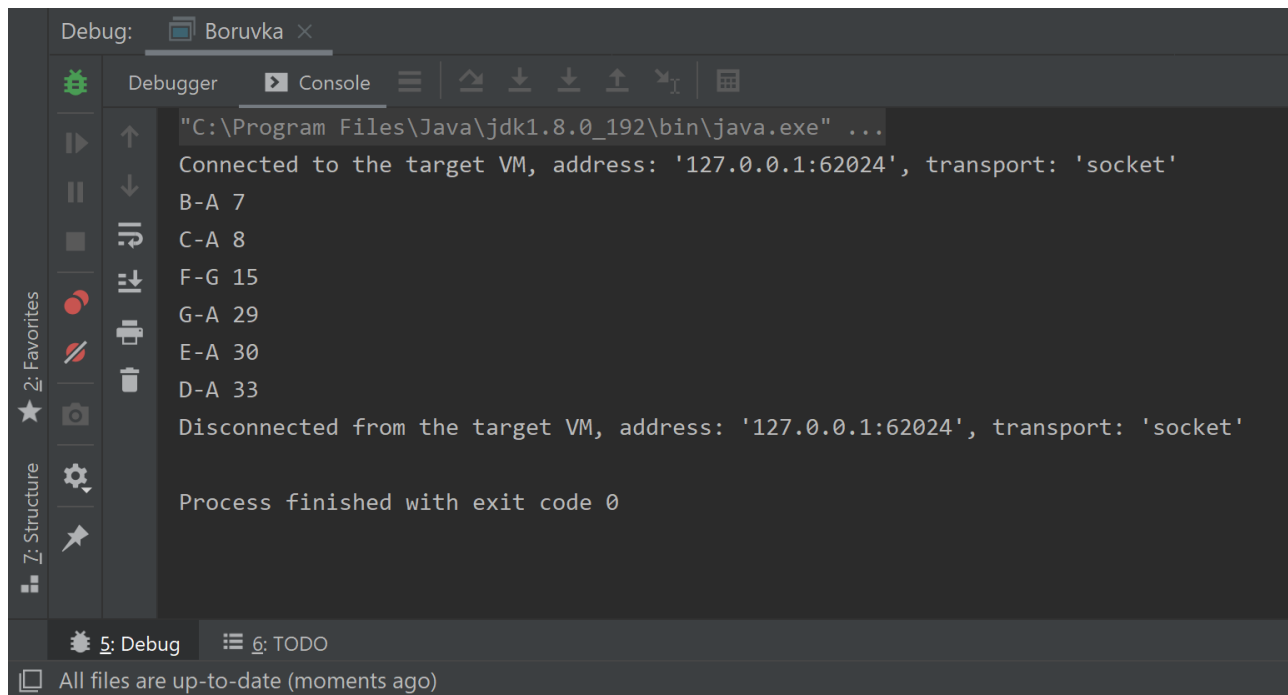
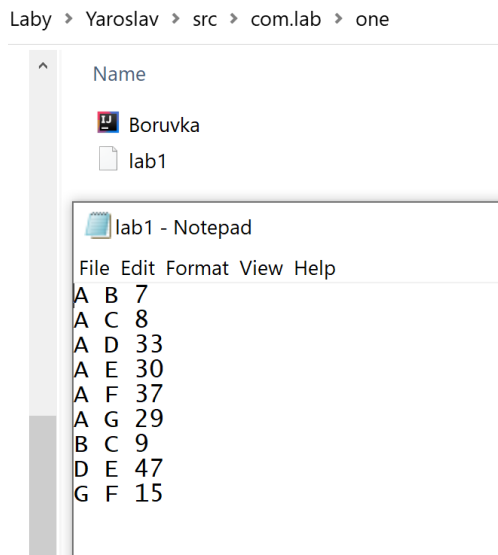


Рис.1. Результат роботи програми

Висновок: На цій лабораторній роботі я ознайомився з алгоритмами побудови остових дерев а також програмно реалізував роботу алгоритму Борувки.

НУЛП, ІКНІ, САПР		Тема	оцінка	підпис
КН-414	19	Алгоритм рішення задачі листоноші		
Михайлівський Я.О.				
№ залікової: 1708315				
Дискретні моделі в САПР			Викладач: к.т.н., асистент Кривий Р.З.	

Мета:

Метою даної лабораторної роботи є вивчення і дослідження алгоритмів рішення задачі листоноші.

Завдання:

Написати програму для демонстрації роботи алгоритму задачі листоноші.

Теоретичні відомості:

Задача листоноші. Основні поняття. Властивості.

Будь-який листоноша перед тим, як відправитись в дорогу повинен підібрати на пошті листи, що відносяться до його дільниці, потім він повинен рознести їх адресатам, що розмістились вздовж маршрута його проходження, і повернутись на пошту. Кожен листоноша, бажаючи втратити якомога менше сил, хотів би подолати свій маршрут найкоротшим шляхом. Загалом, задача листоноші полягає в тому, щоб пройти всі вулиці маршрута і повернутися в його початкову точку, мінімізуючи при цьому довжину пройденого шляху.

Перша публікація, присвячена рішенню подібної задачі, появилась в одному з китайських журналів, де вона й була названа задачею листоноші. Очевидно, що така задача стоїть не тільки перед листоношею. Наприклад, міліціонер хотів би знати найбільш ефективний шлях патрулювання вулиць свого району, ремонтна бригада зацікавлена у виборі найкоротшого шляху переміщення по всіх дорогах.

Задача листоноші може бути сформульована в термінах теорії графів. Для цього побудуємо граф $G = (X, E)$, в якому кожна дуга відповідає вулиці в маршруті руху листоноші, а кожна вершина - стик двох вулиць. Ця задача являє собою задачу пошуку найкоротшого маршруту, який включає кожне ребро хоча б один раз і закінчується у початковій вершині руху.

Нехай S -початкова вершина маршруту і $a(i,j) > 0$ - довжина ребра (i, j) . В графі на рис. 1 існує декілька шляхів, по яким листоноша може обійти всі ребра і повернутись у вершину S .

Наприклад :

Шлях 1: $(S,a), (a,b), (b,c), (c,d), (d,b), (b,S)$

Шлях 2: $(S,a), (a,b), (b,d), (d,c), (c,b), (b,S)$

Шлях 3: $(S,b), (b,c), (c,d), (d,b), (d,a), (a,S)$

Шлях 4: $(S,b), (b,d), (d,c), (c,b), (b,a), (a,S)$

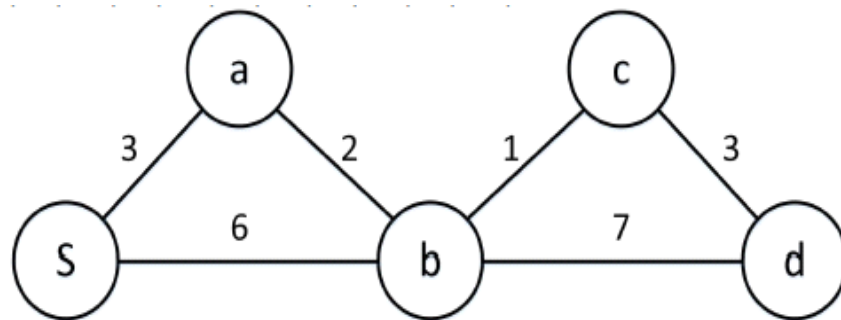


Рис. 1.

В будь-який з чотирьох шляхів кожне ребро входить тільки один раз.

Таким чином, загальна довжина кожного маршруту дорівнює $3+2+1+3+7+6=22$.

Кращих маршрутів у листоноші не існує.

Ейлеровий цикл

Ейлеревим циклом в графі називається шлях, який починається і закінчується в тій самій вершині, при чому всі ребра графа проходяться тільки один раз.

Ейлеревим шляхом називається шлях, який починається в вершині А, а закінчується в вершині Б, і всі ребра проходяться лише по одному разу.

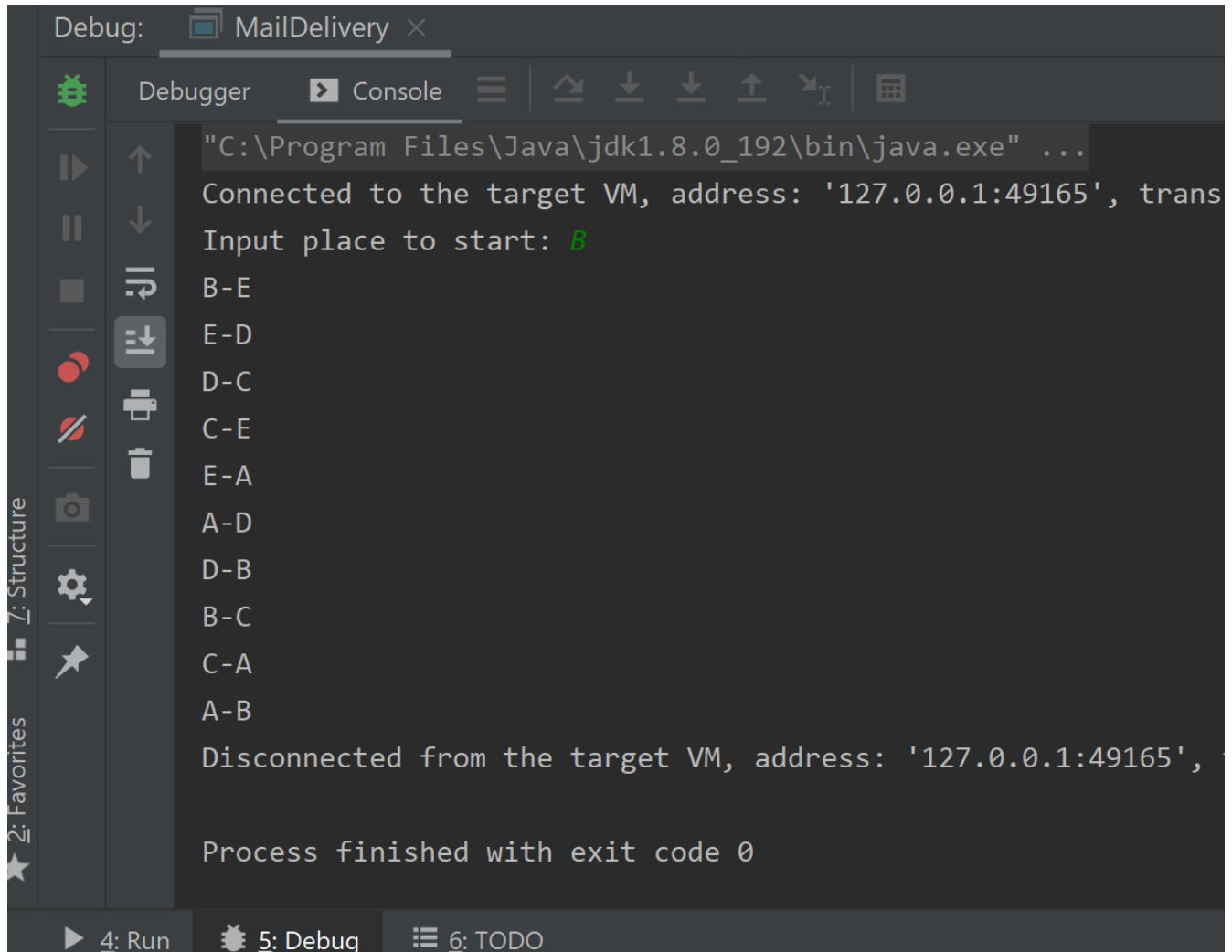
Граф, який включає в себе ейлерів цикл називається ейлеревим.

Індивідуальне завдання

Реалізувати програму для вирішення задачі листоноші.

Робота з програмою:

Після запуску програми з файлу який міститься у папці із вихідним кодом зчитується матриця суміжності графу. Це простий текстовий файл тому параметри легко замінити. Запускаємо програму в режимі дебагу і вводимо вершину з якої почнемо рух. На рисунку бачимо пройдений шлях ейлерівським графом.



Параметри:

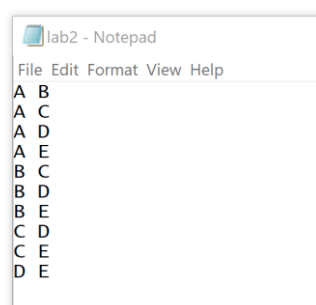


Рис.1 Результат програми

Фрагмент програми:

```
public class MailDelivery {
```

```

final static int M = 10; //number of edges

public static void main(String[] args) {

    boolean[] gasse = new boolean[M];
    Eji[] wvGraf = new Eji[M];

    try(FileReader reader = new
FileReader("E:\\Laby\\Yaroslav\\src\\com.lab\\two\\lab2"))
    {
        int c;
        int numbOut=0, numbIn=0, i=0;
        while((c=reader.read())!=-1){

            if ((char)c=='\n') {
                wvGraf[i]=new Eji(numbOut,numbIn);
                i++;
            }
            else if (c >= 65) {
                numbOut= Transformer.upperLetterToNum((char) c);
                while((c = reader.read())<65);
                numbIn= Transformer.upperLetterToNum((char) c);
            }
        }
    } catch (FileNotFoundException e) {
        e.printStackTrace();
    } catch (IOException e) {
        e.printStackTrace();
    }

    Scanner in = new Scanner(System.in);
    System.out.print("Input place to start: ");
    String firstDot = in.next();

    if (!Walk(gasse,wvGraf, Transformer.upperLetterToNum(firstDot.charAt(0))))
        System.out.println("no way");
}

public static boolean fullWay(boolean[] gasse){
    for (int i = 0; i<gasse.length;i++){
        if(!gasse[i]){
            return false;

```

```

    }
}
return true;
}

public static boolean Walk(boolean[] gasse, Eji[] graph, int gps)
{
    if(fullWay(gasse))
        return true;
    else
    {
        for(int i=0;i<M;i++)
            if(!gasse[i] && (graph[i].v==gps||graph[i].u==gps))
            {
                gasse[i]=true;
                if(graph[i].v==gps)
                {
                    Walk(gasse,graph,graph[i].u);
                    if(fullWay(gasse))
                    {
                        System.out.println(Transformer.numToUpperCaseLetter(graph[i].u)+"-"+
                        Transformer.numToUpperCaseLetter(graph[i].v));
                        return true;
                    }
                }
                else
                    gasse[i]=false;
            }
        else
        {
            Walk(gasse,graph,graph[i].v);
            if(fullWay(gasse))
            {
                System.out.println(Transformer.numToUpperCaseLetter(graph[i].v)+"-"+
                Transformer.numToUpperCaseLetter(graph[i].u));
                return true;
            }
            else
                gasse[i]=false;
        }
    }
}
return false;

```

```

    }
}

private static class Eji {
    int v;
    int u;
    public Eji(int v, int u){
        this.v=v;
        this.u=u;
    }
}

private static class Transformer {
    public static char numToUpperLetter(int num) {
        return (char)(num+65);
    }
    public static int upperLetterToNum(char s) {
        return (int)s-65;
    }
}
}

```

Висновок: На цій лабораторній роботі було здійснено ознайомлення з алгоритмом рішення задачі листоноші.

НУЛП, ІКНІ, САПР		Тема	оцінка	підпис
КН-414	19	Потокові алгоритми		
Михайлівський Я.О.				
№ залікової: 1708315				
Дискретні моделі в САПР			Викладач: к.т.н., асистент Кривий Р.З.	

Мета:

Вивчення поточкових алгоритмів.

Завдання: Написати програму для демонстрації роботи обраного поточкового алгоритму.

Теоретичні відомості:

Алгоритм Форда-Фалкерсона є одним з способів рішення задачі побудови максимального потоку в мережі.

Опис:

Знайти будь-який шлях, що збільшується. Збільшити потік по всіх його ребрах на мінімальну з їх залишкових пропускних здатностей. Повторювати, поки є шлях, що збільшується. Алгоритм працює тільки для цілих пропускних здатностей. В іншому випадку, він може працювати нескінченно довго, не сходячись до правильної відповіді.

Складність:

Залежить від алгоритму пошуку шляху, що збільшується. Потребує $O(\max |f|)$ таких пошуків.

Для пошуку шляху, що збільшується я використав алгоритм пошуку в ширину.

Алгоритм має таке практичне значення: рішення транспортної задачі, наприклад, потрібно перевезти з початкової вершини мережі в кінцеву вантаж по дугах мережі за мінімальний час, при цьому по кожній дузі не можна перевозити вантажу більше фіксованого об'єму.

Код програми:

```
public class Stonks {
    final static int T=7;
    final static int N=6;
    public static void main(String[] args) {

        Potik[] tred = new Potik[T];
        Kpp[] Sr = new Kpp[N];
        int maxTer;
```

```

try(FileReader reader = new FileReader("E:\\Laby\\Yaroslav\\src\\com.lab\\three\\lab3"))
{
    int i=0;
    int numb=0;
    int c;
    int numbOut=0, numbIn=0;
    while((c=reader.read())!=-1){

        if ((char)c=='\n') {
            tred[i]=new Potik(numbOut,numbIn,numb);
            numb=0;
            i++;
        }
        else if (c >= 65) {
            numbOut= Transformer.upperLetterToNum((char) c);
            if(numbOut==19){
                numbOut=N-2;
            }
            else if(numbOut==18){
                numbOut=N-1;
            }
            while((c = reader.read())<65);
            numbIn= Transformer.upperLetterToNum((char) c);
            if(numbIn==19){
                numbIn=N-2;
            }
            else if(numbIn==18){
                numbIn=N-1;
            }
        }
        else if (Character.getNumericValue(c)>=0&&Character.getNumericValue(c)<=9){
            numb *= 10;
            numb += Character.getNumericValue(c);
        }
    }
}

```

```

    } catch (FileNotFoundException e) {
        e.printStackTrace();
    } catch (IOException e) {
        e.printStackTrace();
    }
    for(int i=0;i<N;i++)
    {
        Sr[i]=new Kpp();
        Sr[i].name=i;
        Sr[i].input=0;
        Sr[i].output=0;
    }
    for(int i=0;i<N;i++)
        for(int j=0;j<T;j++)
        {
            if(Sr[i].name==tred[j].a)
                Sr[i].output+=tred[j].c;
            if(Sr[i].name==tred[j].b)
                Sr[i].input+=tred[j].c;
        }
    Sr[N-2].input=Sr[N-2].output;
    Sr[N-1].output=Sr[N-1].input;
    for(int i=0;i<N;i++)
        Sr[i].prop=Math.max(Sr[i].input,Sr[i].output);
    maxTer=Math.max(Sr[N-1].prop,Sr[N-2].prop);

    for(int j=0;j<T;j++)
        if(tred[j].b==N-1)
            Sr[N-1].prop-=Math.min(tred[j].c,Sr[tred[j].a].prop);

    for(int i=N-3;i>=0;i--)
        for(int j=0;j<T;j++)
            if(tred[j].b==i)
                Sr[i].prop-=Math.min(tred[j].c,Sr[tred[j].a].prop);

    System.out.println("Thread: " + maxTer);

```

```
}
```

```
private static class Potik {
```

```
    int a;
```

```
    int b;
```

```
    int c;
```

```
    Potik(int a, int b, int c){
```

```
        this.a=a;
```

```
        this.b=b;
```

```
        this.c=c;
```

```
    }
```

```
}
```

```
private static class Kpp {
```

```
    int name;
```

```
    int input;
```

```
    int output;
```

```
    int prop;
```

```
}
```

```
private static class Transformer {
```

```
    public static char numToUpperCaseLetter(int num) {
```

```
        return (char)(num+65);
```

```
    }
```

```
    public static int upperLetterToNum(char s) {
```

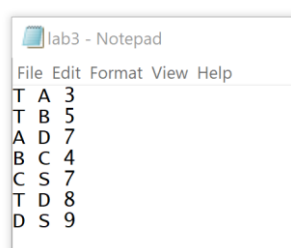
```
        return (int)s-65;
```

```
    }
```

```
}
```

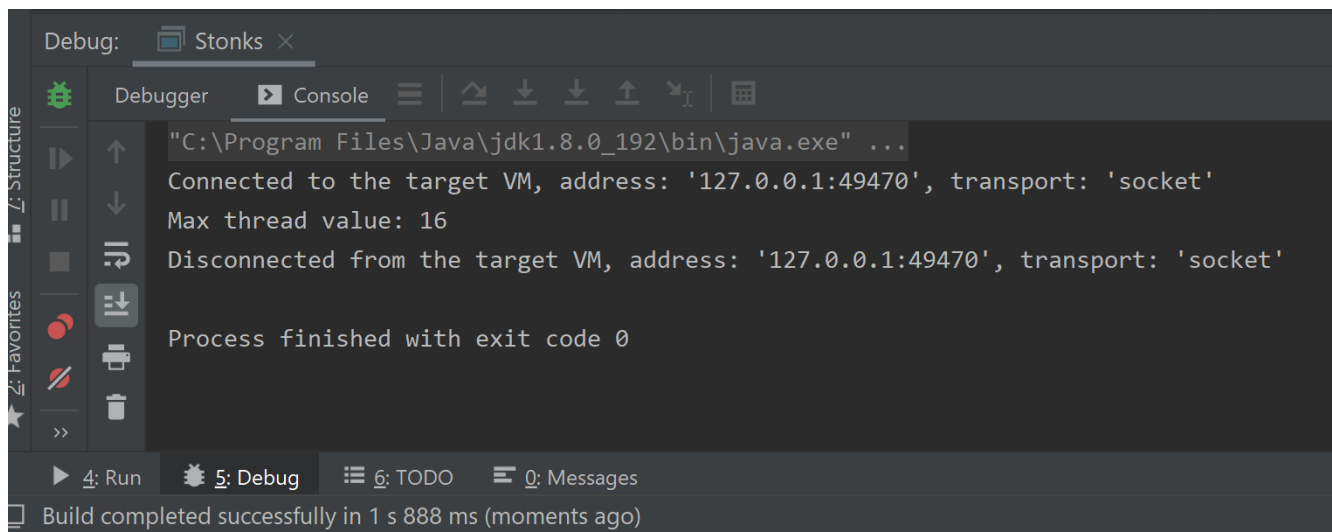
```
}
```

Параметри програми:



Результат роботи програми:

Після запуску із присутнім файлом параметрів програма видає результат у консоль:



```
Debug: Stonks x
Debugger Console
"C:\Program Files\Java\jdk1.8.0_192\bin\java.exe" ...
Connected to the target VM, address: '127.0.0.1:49470', transport: 'socket'
Max thread value: 16
Disconnected from the target VM, address: '127.0.0.1:49470', transport: 'socket'
Process finished with exit code 0
4: Run 5: Debug 6: TODO 0: Messages
Build completed successfully in 1 s 888 ms (moments ago)
```

Висновок: На цій лабораторній роботі я ознайомився з потоковим алгоритмом Форда-Фалкерсона та програмно реалізував його.

НУЛП, ІКНІ, САПР		Тема	оцінка	підпис
КН-414	19	АЛГОРИТМ РІШЕННЯ ЗАДАЧІ КОМІВОЯЖЕРА		
Михайлівський Я.О.				
№ залікової: 1708315				
Дискретні моделі в САПР			Викладач: к.т.н., асистент Кривий Р.З.	

Мета:

Метою даної лабораторної роботи є вивчення і дослідження алгоритмів рішення задачі комівояжера.

Завдання:

Написати програму для демонстрації роботи алгоритму задачі комівояжера.

Теоретичні відомості:

Умови існування гамільтонового контуру. Нижні границі.

Рішенням задачі комівояжера є оптимальний гамільтоновий контур. Нажаль, не всі графи містять гамільтоновий контур. Отже перед тим, ніж перейти до пошуку оптимального гамільтонового контура потрібно довести факт його існування в даному графі.

Можна знайти точний розв'язок задачі комівояжера, тобто, обчислити довжини всіх можливих маршрутів та обрати маршрут з найменшою довжиною. Однак, навіть для невеликої кількості міст в такий спосіб задача практично нерозв'язна. Для простого варіанта, симетричної задачі з n містами, існує $(n - 1)! / 2$ можливих маршрутів, тобто, для 15 міст існує 43 мільярдів маршрутів та для 18 міст вже 177 більйонів. Те, як стрімко зростає тривалість обчислень можна показати в наступному прикладі. Якщо існував би пристрій, що знаходив би розв'язок для 30 міст за годину, то для двох додаткових міст в тисячу раз більше часу; тобто, більш ніж 40 діб.

Відомо багато різних методів рішення задачі комівояжера. Серед них можна виділити методи розроблені Белмором і Немхаузером, Гарфинкелем і Немхаузером, Хелдом і Карном, Стекханом. Всі ці методи відносяться до одного з двох класів: а) методи рішення, які завжди приводять до знаходження оптимального рішення, але потребують для цього, в найгіршому випадку, недопустимо великої кількості операцій(метод гілок та границь); б) методи, які не завжди приводять до знаходження оптимального результату, але потребують для цього допустимої великої кількості операцій (метод послідовного покращення рішення).

Код програми:

```

public class VoyageQuest {
    final static int CVD=10;
    final static int CVV=5;

    public static void main(String[] args) {

        boolean[] city= new boolean[CVV];
        GraphStream[] startGraph = new GraphStream[CVD];

        try(FileReader reader = new FileReader("E:\\Laby\\Yaroslav\\src\\com.lab\\four\\lab4"))
        {
            int i=0;
            int numb=0;
            int c;
            int numbOut=0, numbIn=0;
            while((c=reader.read())!=-1){

                if ((char)c=='\n') {
                    startGraph[i]=new GraphStream(numbOut,numbIn,numb);
                    numb=0;
                    i++;
                }
                else if (c >= 65) {
                    numbOut= Transformer.upperLetterToNum((char) c);
                    while((c = reader.read())<65);
                    numbIn= Transformer.upperLetterToNum((char) c);
                }
                else if (Character.getNumericValue(c)>=0&&Character.getNumericValue(c)<=9){
                    numb *= 10;
                    numb += Character.getNumericValue(c);
                }
            }
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}

```

```
}
```

```
Scanner in = new Scanner(System.in);  
System.out.print("Input place to start: ");  
String firstDot = in.next();
```

```
city[Transformer.upperLetterToNum(firstDot.charAt(0))]=true;  
CVWalk(city,startGraph, Transformer.upperLetterToNum(firstDot.charAt(0)),  
Transformer.upperLetterToNum(firstDot.charAt(0)));  
}
```

```
public static void CVWalk(boolean[] city,GraphStream[] startGraph,int gps,int impulse)  
{  
    while(!isFullVectorTrue(city))  
    {  
        int min=6428;  
        int nD=0;  
        for(int i=0;i<CVD;i++)  
  
        if((startGraph[i].a==gps||startGraph[i].b==gps)&&(!city[startGraph[i].b]||!city[startGraph[i].a]))  
            if(startGraph[i].c<min)  
            {  
                min=startGraph[i].c;  
                if(startGraph[i].a==gps)  
                    nD=startGraph[i].b;  
                else  
                    nD=startGraph[i].a;  
            }  
  
        city[nD]=true;  
        CVWalk(city,startGraph,nD,impulse);  
        if(isFullVectorTrue(city))  
        {  
            System.out.println(Transformer.numToUpperLetter(nD)+"-"+  
Transformer.numToUpperLetter(gps));
```

```

        return;
    }
    else
    {
        city[nD]=false;
        startGraph[nD].c=Integer.MAX_VALUE;
        return;
    }
}

System.out.println(Transformer.numToUpperCase(impulse)+"-"+
Transformer.numToUpperCase(gps));
}

```

```

private static class GraphStream {
    int a;
    int b;
    int c;
    GraphStream(int a, int b, int c){
        this.a=a;
        this.b=b;
        this.c=c;
    }
}

```

```

public static boolean isFullVectorTrue(boolean[] r) {
    for (boolean b : r)
        if (!b)
            return false;
    return true;
}

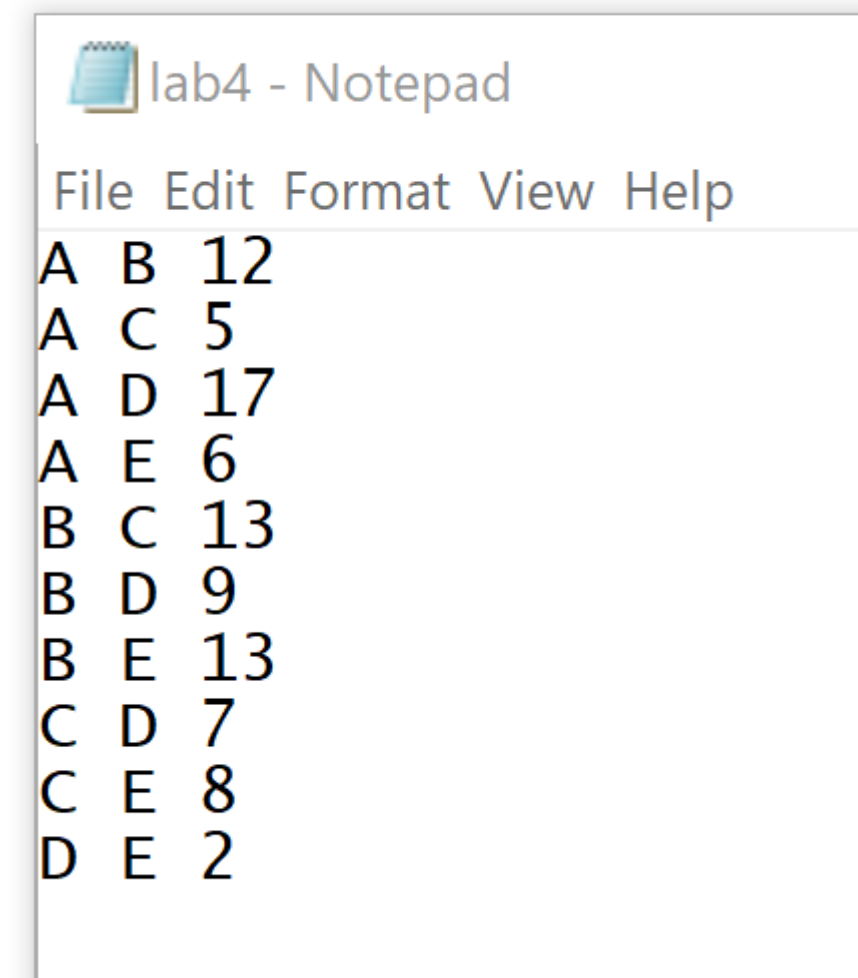
```

```

private static class Transformer {
    public static char numToUpperCase(int num) {
        return (char)(num+65);
    }
}

```

```
public static int upperLetterToNum(char s) {  
    return (int)s-65;  
}  
}  
}
```



The image shows a Notepad window with the title 'lab4 - Notepad'. The window contains a table with three columns: the first column lists letters A, B, C, and D; the second column lists letters B, C, D, and E; and the third column lists numerical values. The table data is as follows:

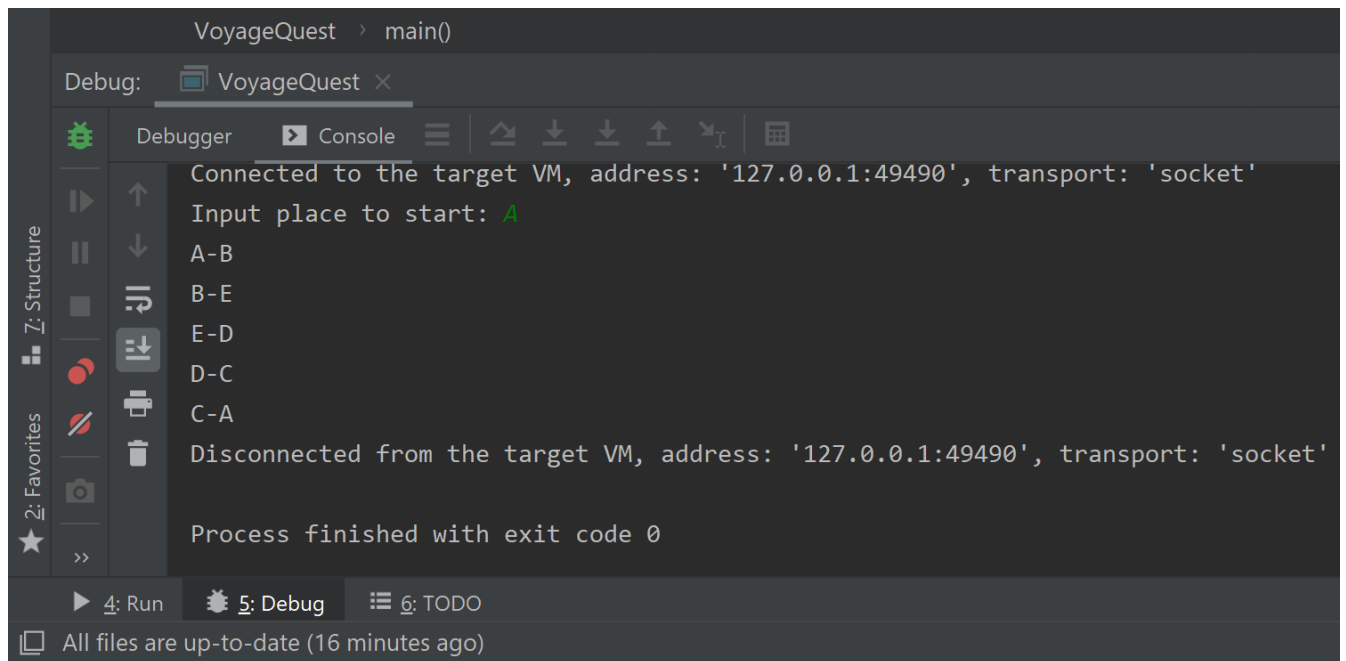
A	B	12
A	C	5
A	D	17
A	E	6
B	C	13
B	D	9
B	E	13
C	D	7
C	E	8
D	E	2

Рис.1 Заданий граф

Робота з програмою:

Запускаємо програму в режимі дебагу і вводимо літеру вершини з якої почнемо шлях.

Результат виводиться в консоль:



Висновок: На цій лабораторній роботі я ознайомився з алгоритмом рішення задачі комівояжера.

НУЛП, ІКНІ, САПР		Тема	оцінка	підпис
КН-414	19	Ізоморфізм графів		
Михайлівський Я.О.				
№ залікової: 1708315				
Дискретні моделі в САПР			Викладач: к.т.н., асистент Кривий Р.З.	

Мета:

Вивчення і дослідження основних підходів до встановлення ізоморфізму графів.

Завдання:

Реалізувати метод повного перебору для встановлення ізоморфізму графів

Теоретичні відомості:

Два графа $G=(X,U,P)$ і $G'=(X',U',P')$ називаються ізоморфними, якщо між їх вершинами, а також між їхніми ребрами можна встановити взаємно однозначне співвідношення $X \leftrightarrow X'$, $U \leftrightarrow U'$, що зберігає інцидентність, тобто таке, що для всякої пари $(x,y) \in X$ ребра $u \in U$, що з'єднує їх, обов'язково існує пара $(x',y') \in X'$ і ребро $u' \in U'$, що з'єднує їх, і навпаки. Тут P - предикат, інцидентор графа G . Зауважимо, що відношення ізоморфізму графів рефлексивне, симетричне і транзитивне, тобто представляє собою еквівалентність.

Одним з найпростіших з точки зору програмної реалізації, є алгоритм перевірки ізоморфізму графів повним перебором(можливої перенумерації вершин), але складність цього алгоритму є факторіальною.

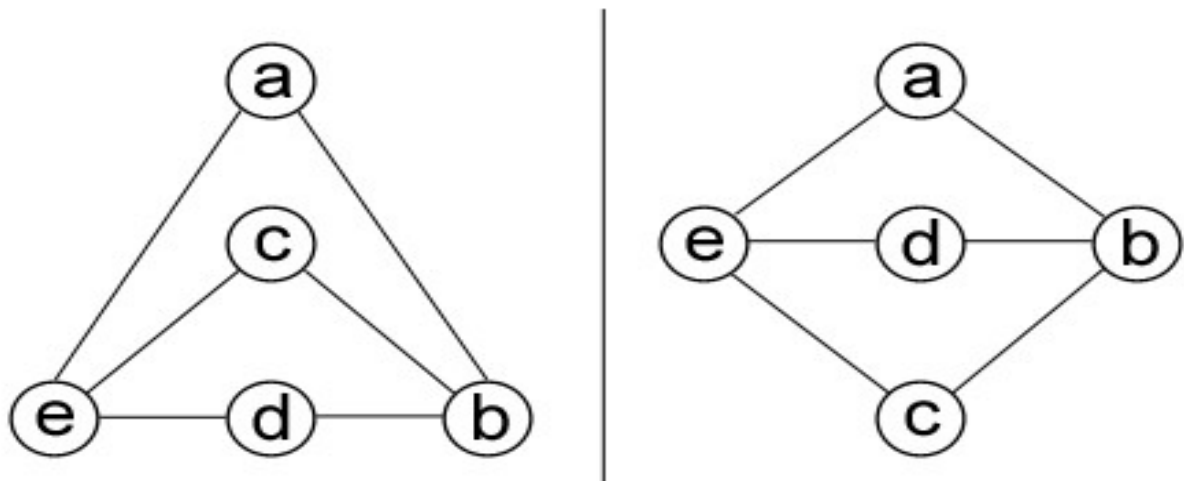


Рис.1 Графи для перевірки ізоморфізму (5 вершин)

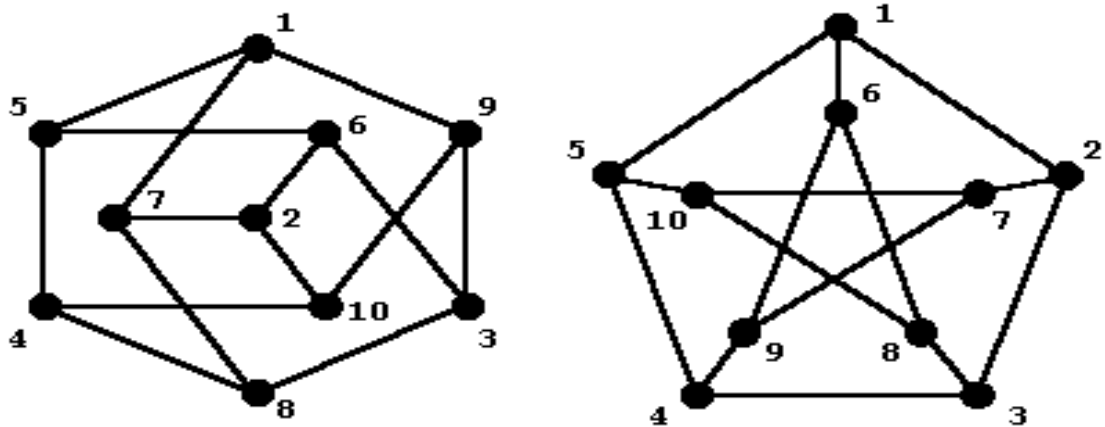
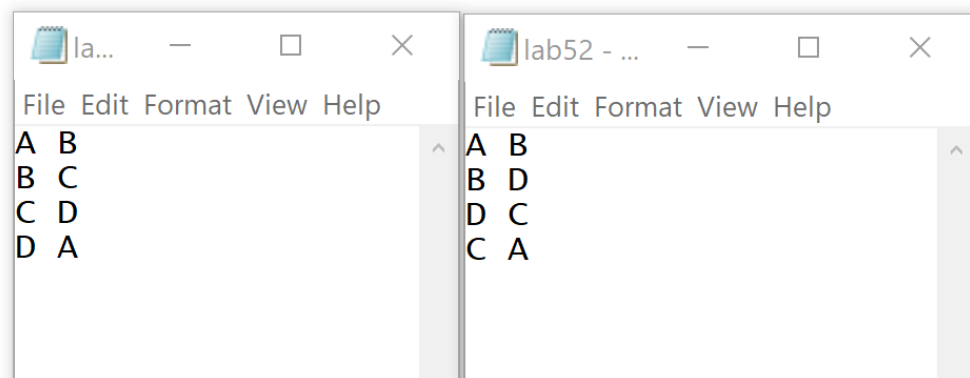


Рис.2 Графи для перевірки ізоморфізму (10 вершин)

Робота з програмою:

Запускаємо в режимі дебагу.

Вводимо вхідні дані у текстові файли, або залишаємо за замовчуванням:



Фрагмент програми:

```
public class Izomorphizm {
```

```
    final static int ZN = 4;
```

```
    final static int ZE = 4;
```

```
    final static int N = 4;
```

```
    static boolean general = false;
```

```
    public static void main(String[] args) {
```

```
        izomorfizm();
```

```
        if(general)
```

```
            System.out.println("its work");
```

```
        else System.out.println("its bad");
```

```
}
```

```
public static void izomorfizm()
```

```
{
```

```
    MatGraph oneInc= new MatGraph(ZN);
```

```
    MatGraph twoInc= new MatGraph(ZN);
```

```
    Eji[] oneGraph = new Eji[ZE];
```

```
    Eji[] twoGraph = new Eji[ZE];
```

```
    Eji.readGraphFromFile(oneGraph,"E:\\Laby\\Yaroslav\\src\\com.lab\\five\\lab51");
```

```
    Eji.readGraphFromFile(twoGraph,"E:\\Laby\\Yaroslav\\src\\com.lab\\five\\lab52");
```

```
    for(int i=0;i<ZE;i++)
```

```
    {
```

```
        oneInc.mat[oneGraph[i].A][oneGraph[i].B]=true;
```

```
        oneInc.mat[oneGraph[i].B][oneGraph[i].A]=true;
```

```
        twoInc.mat[twoGraph[i].A][twoGraph[i].B]=true;
```

```
        twoInc.mat[twoGraph[i].B][twoGraph[i].A]=true;
```

```
    }
```

```
    oneInc.print();
```

```
    twoInc.print();
```

```
    oneInc.antiphlex(twoInc, N-1);
```

```
}
```

```
private static class MatGraph {
```

```
    public boolean[][] mat;
```

```
    int size;
```

```
    private MatGraph(int size){
```

```
        this.size=size;
```

```
        mat=new boolean[size][size];
```

```
    }
```

```
    public boolean sameAs(MatGraph da){
```

```
        if(da.size!=this.size) return false;
```

```
        for (int i=0;i<size;i++) {
```

```

        for (int j = 0; j < size; j++) {
            if (da.mat[i][j]!=this.mat[i][j]) return false;
        }
    }
    return true;
}

```

```

static void reverse(MatGraph P, int m) {
    int i = 0, j = m;
    while (i < j) {
        P.swapLeafs(i,j);
        ++i;
        --j;
    }
}

```

```

void antiphlex(MatGraph P, int m) {
    int i;
    if (m == 0) {
        if(this.sameAs(P)) {
            general=true;
        }
    } else {
        for (i = 0; i <= m; ++i) {
            this.antiphlex(P, m-1);
            if (i < m) {
                P.swapLeafs(i,m);
                reverse(P, m - 1);
            }
        }
    }
}

```

```

public void print() {
    for (int i = 0; i < size; i++) {
        for (int j=0;j<size;j++){

```

```

        System.out.print(mat[i][j]+ " ");
    }
    System.out.println();
}
System.out.println("-----");
}

public void swapLeafs(int x, int y){
    for (int i=0;i<size;i++){
        boolean k=mat[i][x];
        mat[i][x]=mat[i][y];
        mat[i][y]=k;
    }
    for (int j=0;j<size;j++){
        boolean k=mat[x][j];
        mat[x][j]=mat[y][j];
        mat[y][j]=k;
    }
}

}

private static class Eji {
    public int A;
    public int B;
    private Eji(int a, int b){
        A=a;
        B=b;
    }
}

public static void readGraphFromFile(Eji[] graph,String filename){
    try(FileReader reader = new FileReader(filename))
    {
        int c;
        int numbOut=0, numbIn=0, i=0;
        while((c=reader.read())!=-1){

```

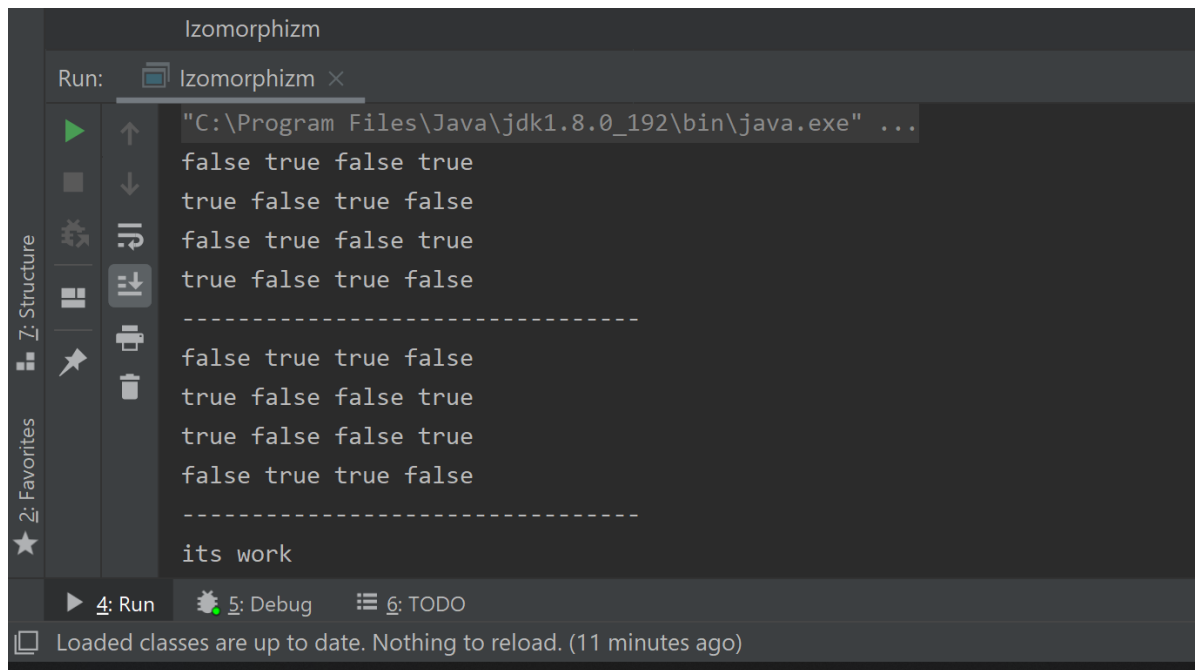
```

        if ((char)c=='\n') {
            graph[i]=new Eji(numbOut,numbIn);
            i++;
        }
        else if (c >= 65) {
            numbOut= Transformer.upperLetterToNum((char) c);
            while((c = reader.read())<65);
            numbIn= Transformer.upperLetterToNum((char) c);
        }
    }
} catch (FileNotFoundException e) {
    e.printStackTrace();
} catch (IOException e) {
    e.printStackTrace();
}
}
}

private static class Transformer {
    public static char numToUpperLetter(int num) {
        return (char)(num+65);
    }
    public static int upperLetterToNum(char s) {
        return (int)s-65;
    }
}
}

```

Результат виконання програми:



```
Izomorphizm
Run: Izomorphizm x
"C:\Program Files\Java\jdk1.8.0_192\bin\java.exe" ...
false true false true
true false true false
false true false true
true false true false
-----
false true true false
true false false true
true false false true
false true true false
-----
its work
```

4: Run 5: Debug 6: TODO

Loaded classes are up to date. Nothing to reload. (11 minutes ago)

Висновок: На цій лабораторній роботі я ознайомився з алгоритмом повного перебору визначення ізоморфності графів.