سید شایان دانشور به نام خدا 9726523

داک پروژه پنجم – پیاده سازی الگوریتم مین کات – کارگر-اشتاین

پیش نیاز برای اجرا:

JDK11+

نحوه اجرا:

gradlew run در پوشه پروژه

کلاس Main :

```java
package ir.shayandaneshvar;

import ...

public class Main extends Application {
    private Scene scene;

    public static void main(String[] args) {
        launch(args);
    }

    @Override
    public void init() throws Exception {
        AnchorPane root = FXMLLoader.load(getClass().getResource( name: "/view/main.fxml"));
        scene = new Scene(root, width: 255, height: 520, depthBuffer: false, SceneAntialiasing.BALANCED);
    }

    @Override
    public void start(Stage stage) {
        stage.initStyle(StageStyle.UTILITY);
        stage.setResizable(false);
        stage.setScene(scene);
        stage.setTitle("Karger-Stein");
        stage.show();
        stage.setAlwaysOnTop(true);
        stage.setOnCloseRequest(status -> System.exit( status: 0));
    }
}
```

در این کلاس برنامه شروع شده و فایل گرافیک main.fxml خوانده شده و برنامه به کلاس presenter که به گرافیک متصل است هدایت می شود.

main.fxml پنل گرافیکی برنامه است که در پروژه سوم نیز مشابه آن استفاده شده است.

کلاس های Graph و Edge : دقیقا همان ساختمان داده های مورد استفاده در پروژه سوم (کروسکال) هستند، با این تفاوت که بخش های مربوط به پیدا کردن دور در گراف از آن حذف شده اند و edge به جای نگهداری دو راس، قابلیت نگهداری چند راس را دارند، که البته در الگوریتم ابتدا سعی بر آن شد تا عملیات merge روی edge صورت بگیرد که با مشکل مواجه شد، الگوریتم را عوض کردیم و دیگر نیازی به لیست نبود و می توان آن را ریفاکتور کرد و به شکل پروژه قبل در آورد. (در آینده این بخش ریفاکتور شده در گیتهاب قرار خواهد گرفت)

```java
package ir.shayandaneshvar;

import java.io.Serializable;
import java.util.HashSet;
import java.util.Set;

public class Graph implements Serializable {
    private Integer vertices;
    private HashSet<Edge> edges;

    public Graph(Integer numberOfVertices) {
        vertices = numberOfVertices;
        edges = new HashSet<>();
    }

    public void addEdge(Edge edge) { edges.add(edge); }

    public Set<Edge> getEdges() {
        return (Set<Edge>) edges.clone();
    }

    public Integer getSize() { return vertices; }

}
```

```java
package ir.shayandaneshvar;

import ...

public class Edge implements Serializable {
    private ArrayList<Pair<Integer, Integer>> edges;
    private Boolean marked = false;

    public Edge(Integer from, Integer to) {...}

    public Edge() { edges = new ArrayList<>(); }

    public void addEdge(Integer from, Integer to) {...}

    public void setMarked(boolean bool) { marked = bool; }

    public Boolean marked() { return marked; }

    public ArrayList<Pair<Integer, Integer>> getEdges() {...}

    public Integer getFrom() { return edges.get(0).getKey(); }

    public Integer getTo() { return edges.get(0).getValue(); }

    @Override
    public boolean equals(Object o) {...}

    @Override
    public int hashCode() {...}
}
```

کلاس Presenter :

```java
import ...
public class Presenter {
    private Graph graph;
    private SmartGraphPanel<String, String> graphView;
    private GraphEdgeList<String, String> graphEdgeList;
    private ArrayList<Integer>[] minCutGraph = null;
    private Integer minCut = Integer.MAX_VALUE;
    private Integer[][] matrix;
    @FXML private JFXTextArea matrixArea;
    @FXML private JFXTextField length;
    @FXML private JFXCheckBox autoLayout;
    @FXML private JFXCheckBox optimized;
    @FXML void drawGraph() {...}
    private void prepareGraph(Graph graph) {...}
    private void handleGraphWindow(String title) {...}
    private void getGraph(int size) {...}
    private static double log2(double x) { return Math.log(x) / Math.log10(2); }
    @FXML void drawMinCut() {...}
    private Graph getMinCut() {
        final double delta = 0.05;
        minCutGraph = new ArrayList[graph.getSize()];
        for (int i = 0; i < graph.getSize(); i++) {...}
        for (int i = 0; i < (optimized.isSelected() ?
                log2(graph.getSize()) * log2( x: 1.0 / delta) : 1); i++) {
            kargerSteinAlgorithm(minCutGraph);
        }
        Graph result = new Graph(graph.getSize());
        graph.getEdges().forEach(result::addEdge);
        for (var i : minCutGraph[0]) {
            for (var j : minCutGraph[1]) {
                result.getEdges().stream().filter(x -> (x.getFrom().equals(i) &&
                        x.getTo().equals(j)) || (x.getFrom().equals(j) && x.
                        getTo().equals(i))).forEach(y -> y.setMarked(true));
            }
        }
        return result;
    }
```

متد drawGraph : این متد به دکمه موجود در گرافیک متصل است و وظیفه ایجاد صفحه جدید جهت ترسیم گراف را مهیا کرده و به کمک متد getGraph ورودی ها خوانده می شوند.

متد های prepareGraph و handleGraphWindow : این دو متد که درون متد drawGraph صدا زده می شوند، وظیفه ایجاد یک پنجره جدید و نیز رسم گراف در آن را دارند که کاملا مشابه پروژه کروسکال است.

(فیلد های graphEdgeList و graphView برای تبدیل ساختمان داده ایجاد شده به ساختمان داده قابل قبول کتابخانه گرافیکی و نیز رسم هوشمند گراف در این دو متد به کار رفته اند)

متد getGraph : این متد ورودی را خوانده در دو فرم ماتریسی و لیست مجاورت (لیستی از یال ها) در دو فیلد matrix و graph نگهداری می کند. (دقیقا مشابه پروژه سوم (کروسکال) با این تفاوت که فرم ماتریسی برای راحت تر شدن محسبات در الگوریتم نیز نگداری می شود.

متد drawMinCut : این متد نیز به دکمه موجود در گرافیک متصل است و در آن گراف خوانده شده را به متد getMinCut داده سپس نتیجه را به متد های قبلی ذکر شده می دهد تا رسم شود.

فیلد optimized که به checkbox موجود در گرافیک متصل است اضافی است به این معنی که اگر تیک این گزینه نخورد تنها یکبار الگوریتم کارگراشتاین فراخوانی می شود و در صورتی که بخورد به تعداد log(n)log(1/delta) که دلتا احتمال عدم بدست آمدن حالت بهینه است اجرا می شود.

متد getMinCut همانطور که توضیح داده شد برای optimized این متد بسته به تیک یکبار یا چندین بار متد kargerSteinAlgorithm اجرا می شود، پس از اتمام اجرا نتیجه نهایی در فیلد minCutGraph ذخیره می شود که نشان دهنده رئوس است. پس از اجرای الگوریتم، با پیچیدگی n^2 یال های حذف شده را استخراج میکنیم و به فرم ساختمان داده گراف در می آوریم تا قابل رسم شود.

متد kargerSteinAlgorithm : الگوریتم اصلی برنامه در این متد قرار دارد.

در اولین قسمت کد شرط خروج از تابع بازگشتی قرار دارد که بدیهتا زمانی است که دو راس مانده باشد، در این قسمت، گراف دو قسمت شده و باید یال های موجود بین این دو قسمت که جواب هستند شمرده شوند و در صورتی که تعداد این یال ها کمتر از دفعات قبل بودند در فیلد های minCut و minCutGraph قرار گیرند.

این بخش که شامل دو حلقه for هستند، در بدترین حالت دارای پیچیدگی n^2 می باشند یا دقیق تر هر گراف حداکثر n^2/2 یال دارد پس پیچیدگی این قسمت در بدترین حالت (که عملا هیچ وقت اتفاق نمی افتد) دارای پیچیدگی n^2 است.

بخش دوم که حلقه while قرار دارد به قدری تصادفی انتخاب می کند و merge میکند تا سایز آن به $\frac{n}{\sqrt{2}}$ برسد (و یا$\frac{n}{\sqrt{3}}$) سپس به صورت بازگشتی متد را صدا میزند، انتخاب تصادفی به کمک متد getRandomVertices صورت میگیرد که مطمئن میشود دو راس رندوم قبلا با هم merge نشده اند و نیز بین این دو راس یالی وجود دارد. خود merge نیز با پیچیدگی n انجام می شود پس این بخش دارای پیچیدگی n است و چون در حلقه while قرار داریم، حداکثر n^2 بار اجرا می شوند، پس حداکثر پیچیدگی در هر مرحله از $O(n^2)$ است.

پس تابع بازگشتی پیچیدگی ما برابر $T(n) = 2T\left(\frac{n}{\sqrt{2}}\right) + O(n^2)$ است که اگر با قضیه اساسی (master theorem) آن را حل کنیم پیچیدگی n^2lg(n) را به ما می دهد و اگر حالت جذر 3 را نظر بگیریم داریم:$T(n) = 2T\left(\frac{n}{\sqrt{3}}\right) + O(n^2)$ که طبق قضیه اساسی پیچیدگی آن برابر $O\left(n^{\log_{\sqrt{3}} 2}\right)$ که برابر $O(n^{1.26})$ می شود که بهتر از حالت قبل است.

اگر حالت مازاد بر پروژه، یعنی حالت جواب بهینه را بدست آوریم، پیچیدگی در یک لگاریتم n نیز ضرب می شود.

کتابخانه های گرافیکی استفاده شده در پروژه :

JavaFX 11.0.6

JavaFX SmartGraph - JFoenix 9

برای راحتی اجرا (کتابخانه JavaFX SmartGraph در Maven Central موجود نیست) از پکیج منیجر Gradle استفاده شده است،از آنجا که ممکن است این برنامه را نصب نداشته باشید، gradlewrapper در کنار پروژه گذاشته شده است تا مشکلی برای اجرا نداشته باشید. (در صورتی که دارید با دستور gradle run میتوانید برنامه را اجرا کنید)

```
134 @   private void kargerSteinAlgorithm(ArrayList<Integer>[] temp) {
135         ArrayList<Integer>[] tempGraph = temp;
136         if (temp.length == 2) {...}
151         while (tempGraph.length > (int) (temp.length / Math.sqrt(2))) {
152             var edge = getRandomVertices(tempGraph);
153             ArrayList<Integer> first = Arrays.stream(tempGraph)
154                     .filter(z -> z.contains(edge.getValue()))
155                     .findAny().orElseThrow();
156             ArrayList<Integer> second = Arrays.stream(tempGraph)
157                     .filter(z -> z.contains(edge.getKey()))
158                     .findAny().orElseThrow();
159             ArrayList<Integer>[] newTemp = new ArrayList[tempGraph.length - 1];
160             for (int j = 0, i = 0; i < tempGraph.length; i++) {...}
167             first.addAll(second);
168             newTemp[newTemp.length - 1] = first;
169             tempGraph = newTemp;
170         }
171         kargerSteinAlgorithm(tempGraph);
172         kargerSteinAlgorithm(tempGraph);
173     }
174 @   private Pair<Integer, Integer> getRandomVertices(ArrayList<Integer>[] inputGraph) {
175         int rand = (int) Math.abs(Math.random() * 1000000) % inputGraph.length;
176         int rand1 = (int) Math.abs(Math.random() * 100000) % inputGraph.length;
177         int u = inputGraph[rand].get((int) Math.abs(Math.random() * 100000) %
178                 inputGraph[rand].size());
179         int v = inputGraph[rand1].get((int) Math.abs(Math.random() * 100000) %
180                 inputGraph[rand1].size());
181         if (rand == rand1 || graph.getEdges().stream().flatMap(x -> x.getEdges()
182                 .stream()).noneMatch(z ->
183                 (z.getKey().equals(v) && z.getValue().equals(u)) || (z.
184                         getKey().equals(u) && z.getValue().equals(v)))) {...}
187         return new Pair<>(u, v);
188     }
189 }
```

فایل های css و properties برای تنظیمات کتابخانه گرافیکی و رنگ های موجود در آن هستند و اطلاعات موجود در آن ها توسط کتابخانه javaFX SmartGraph مورد استفاده قرار میگیرند، در صورتی که پاک شوند رسم گراف به مشکل میخورد.

نتیجه : پیچیدگی زمانی در حالت دوم (رادیکال 3) بهتر است.

مابقی قسمت های توضیح داده نشده شامل فیلد ها و متد ها، کاربرد ساده و بدیهی دارند و برخی نیز دقیقا به همین شکل در پروژه کروسکال مورد استفاده قرار گرفته اند، بخش های مربوط به گرافیک نیز بار ها در پروژه های قبل توضیح داده شده اند و پیچیدگی خاصی ندارند.

نمونه اجرا:

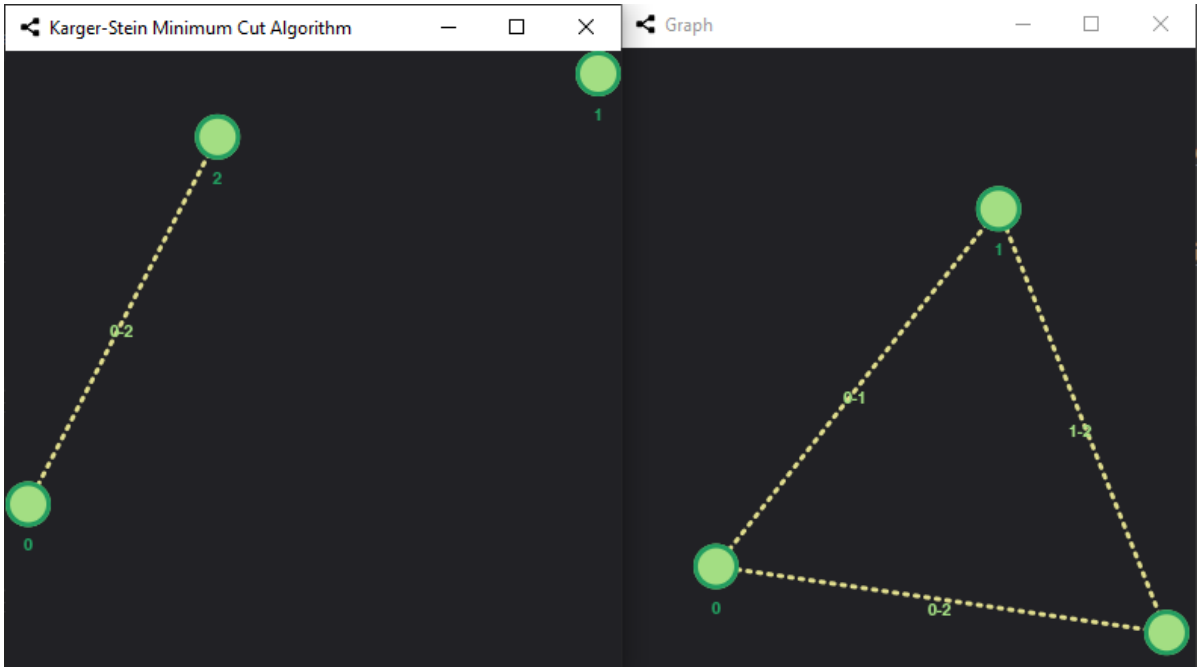
ورودی و خروجی:

سید شایان دانشور به نام خدا 9726523

011

101

110



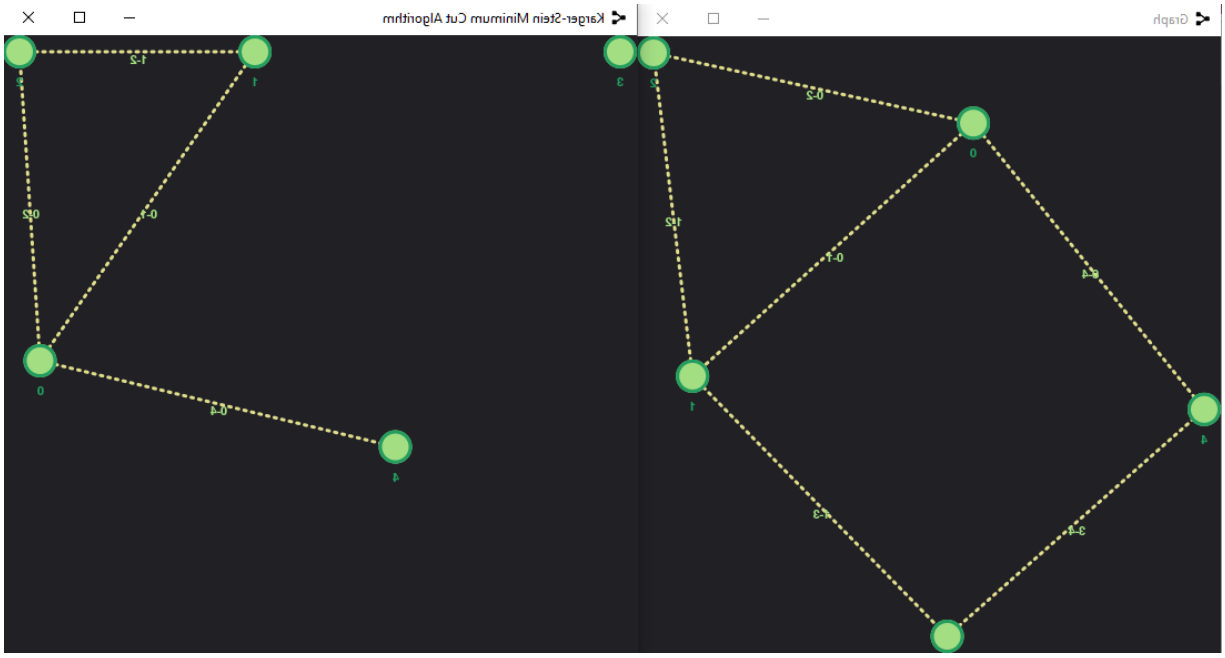
01101

10110

11000

01001

10010



0 1 1 1

1 0 1 0

1 1 0 1

1 0 1 0

0 1 1 0 1 0

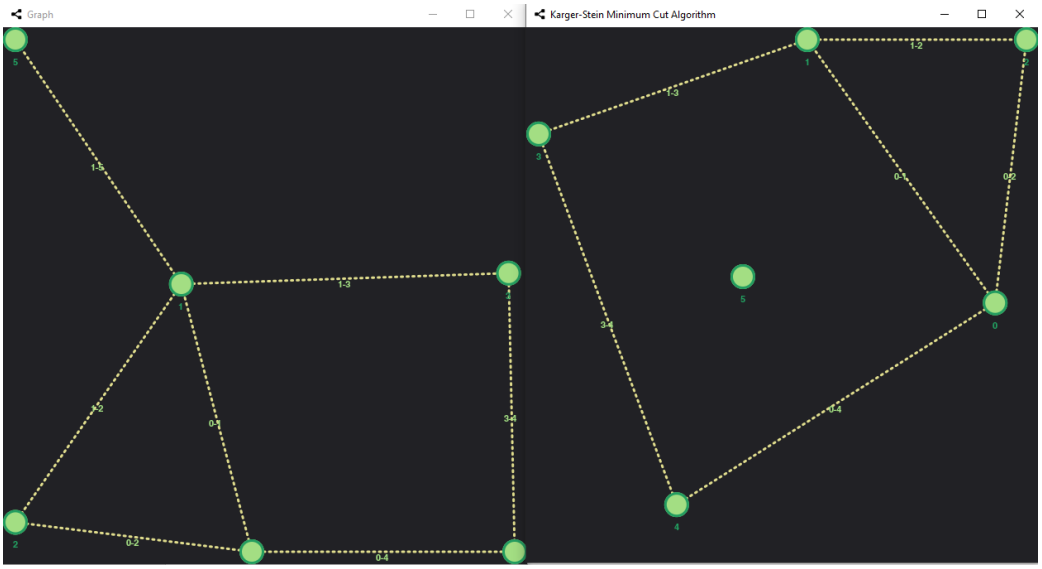
1 0 1 1 0 1

1 1 0 0 0 0

0 1 0 0 1 0

1 0 0 1 0 0

0 1 0 0 0 0



0 1 1 0 1 0 1

1 0 1 1 0 1 0

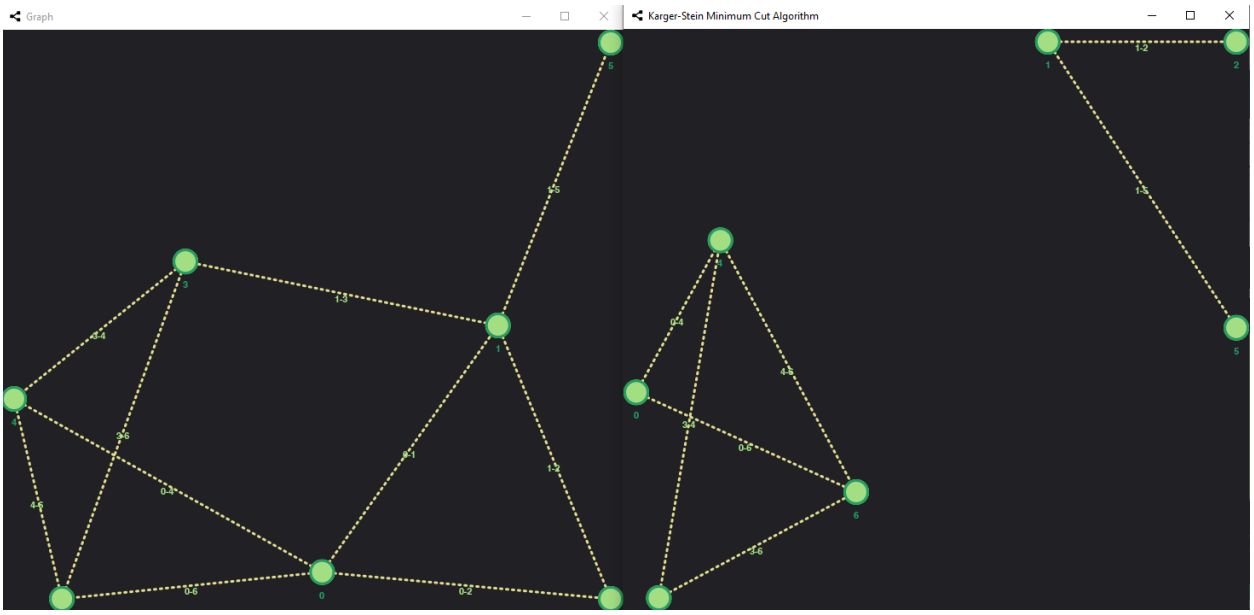
1 1 0 0 0 0 0

0 1 0 0 1 0 1

1 0 0 1 0 0 1

0 1 0 0 0 0 0

1 0 0 1 1 0 0



0 1

1 0