پروژه سوم – الگوریتم کروسکال

کلاس Graph و Subset :

```java
package ir.shayandaneshvar;

import java.util.HashSet;
import java.util.Set;

public class Graph {
    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;
    }

    public static class Subset {
        private Integer parent;

        public Integer getParent() { return parent; }

        public void setParent(Integer parent) { this.parent = parent; }
    }
}
```

برای پیاده سازی الگوریتم کروسکال، گراف به این شیوه پیاده سازی شده که مجموعه یال ها را ذخیره می کنیم و هر یال دو راسی که به آن متصل است را نگه می دارد، از Hashset برای اینکار استفاده شده . متد hash برای کلاس edge اورراید شده که به آن میرسیم.

رئوس نیز از 0 تا n-1 شماره گذاری می شوند که vertices برابر n می باشد.

کلاس داخلی Subset نیز که برای پیدا کردن دور در گراف به کمک الگوریتم disjoint set که در کتاب نپولیتان ذکر شده است نوشته شده تنها یک فیلد دارد که مشخص کند پدر راس فعلی کدام راس است، پدر با شماره اش مشخص می شود، برای عدد راس فعلی جلوتر از آرایه استفاده شده و نیازی به شماره راس فعلی نبوده است.

کلاس Edge :

```java
package ir.shayandaneshvar;

import java.util.Objects;

public class Edge implements Comparable<Edge> {
    private Integer from, to, weight;
    public Edge(Integer from, Integer to, Integer weight) {
        this.from = from;
        this.to = to;
        this.weight = weight;
    }
    @Override public boolean equals(Object o) {
        if (this == o) {
            return true;
        }
        if (o == null || getClass() != o.getClass()) {
            return false;
        }
        Edge edge = (Edge) o;
        return (Objects.equals(from, edge.from) || Objects.equals(to, edge
                .from)) && (Objects.equals(to, edge.to) || Objects.equals(from,
                edge.to)) && Objects.equals(weight, edge.weight);
    }

    @Override public int hashCode() {
        return Objects.hash( ...values: from, to, weight) +
                Objects.hash(to, from, weight) + 7 * (to + from) + 11 * weight;
    }

    @Override public int compareTo(Edge o) {
        return weight - o.weight;
    }
    public Integer getFrom() { return from; }
    public Integer getTo() { return to; }
    public Integer getWeight() { return weight; }
}
```

این کلاس پیاده سازی یال هاست و دو راس متصل به آن و وزن خودش را نگه می دارد.

متد equals : در صورتی دو راس مساوی می شوند که دو راس متصل به آن با هم مساوی باشند و وزن آن ها یکی باشد.

متد hashcode : به این شکل نوشته شده تا فرقی بین اینکه کدام راس اول باشد و کدام راس دوم تفاوتی ایجاد نشود و hashset موجود در کلاس گراف دچار مشکل نشود.

اینترفیس Comparable نیز پیاده سازی شده چرا که در الگوریتم کروسکال میخواهیم بر اساس وزن یال مرتب کنیم، در متد compareTo نیز معیار مقایسه تنها وزن یال است.

کلاس KruskalFX:

```java
package ir.shayandaneshvar;

import javafx.application.Application;
import javafx.fxml.FXMLLoader;
import javafx.scene.Scene;
import javafx.scene.SceneAntialiasing;
import javafx.scene.layout.AnchorPane;
import javafx.stage.Stage;
import javafx.stage.StageStyle;

public class KruskalFX extends Application {
    public 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, v: 255, v1: 520, b: false, SceneAntialiasing.BALANCED);
    }

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

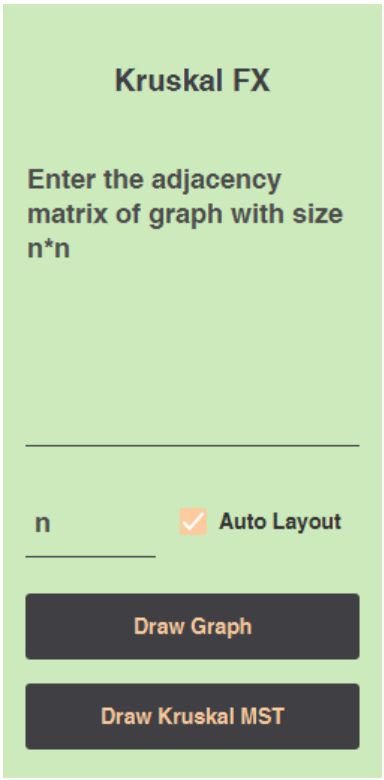
برنامه را شروع میکند و فایل fxml مربوط به گرافیک پروژه را لود میکند، و کنترل view بدست کنترلر تعریف شده در fxml که کلاس Presenter است میفتد.

فایل main.fxml : منوی گرافیکی در این فایل دیزاین شده است.

```xml
<?xml version="1.0" encoding="UTF-8"?>

<?import com.jfoenix.controls.JFXButton?>
<?import com.jfoenix.controls.JFXCheckBox?>
<?import com.jfoenix.controls.JFXTextArea?>
<?import com.jfoenix.controls.JFXTextField?>
<?import javafx.scene.control.Label?>
<?import javafx.scene.layout.AnchorPane?>
<?import javafx.scene.text.Font?>

<AnchorPane prefHeight="520.0" prefWidth="255.0" style="-fx-background-color: blue;" xmlns="http://javafx.com/javafx/11.0.1" xmlns:fx="http://javafx.com/fxml/1" fx:controller="ir.shayandaneshvar.Presenter">
   <children>
      <AnchorPane prefHeight="520.0" prefWidth="255.0" style="-fx-background-color: #cceabb;">
         <children>
            <Label layoutX="75.0" layoutY="40.0" text="Kruskal FX" textFill="#3f3f44">
               <font>
                  <Font name="Helvetica-Bold" size="20.0" />
               </font>
            </Label>
            <JFXTextArea fx:id="matrixArea" focusColor="#fdcb9e" layoutX="16.0" layoutY="101.0" prefHeight="194.0" prefWidth="223.0" promptText="Enter the adjacency         matrix of graph with size n*n" unFocusColor="#3f3f44">
               <font>
                  <Font name="Helvetica-Bold" size="18.0" />
               </font>
            </JFXTextArea>
            <JFXTextField fx:id="length" focusColor="#fdcb9e" layoutX="16.0" layoutY="325.0" prefHeight="44.0" prefWidth="87.0" promptText=" n" unFocusColor="#3f3f44">
               <font>
                  <Font name="Helvetica-Bold" size="18.0" />
               </font>
            </JFXTextField>
            <JFXButton layoutX="16.0" layoutY="395.0" onKeyPressed="#drawGraph" onMouseClicked="#drawGraph" prefHeight="44.0" prefWidth="223.0" style="-fx-background-color: #3f3f44;" text="Draw Graph" textFill="#fdcb9e">
               <font>
                  <Font name="Helvetica-Bold" size="14.0" />
               </font>
            </JFXButton>
            <JFXButton layoutX="16.0" layoutY="455.0" onKeyPressed="#drawKruskalMST" onMouseClicked="#drawKruskalMST" prefHeight="44.0" prefWidth="223.0" style="-fx-background-color: #3f3f44;" text="Draw Kruskal MST" textFill="#fdcb9e">
               <font>
                  <Font name="Helvetica-Bold" size="14.0" />
               </font>
            </JFXButton>
            <JFXCheckBox fx:id="autoLayout" checkedColor="#fdcb9e" layoutX="119.0" layoutY="338.0" prefHeight="18.0" prefWidth="122.0" selected="true" text="Auto Layout" unCheckedColor="#3f3f44">
               <font>
                  <Font name="Helvetica-Bold" size="14.0" />
               </font>
            </JFXCheckBox>
         </children>
      </AnchorPane>
   </children>
</AnchorPane>
```



کلاس presenter :

```java
package ir.shayandaneshvar;
import ...
public class Presenter {
    private Graph graph;
    private SmartGraphPanel<String, String> graphView;
    private GraphEdgeList<String, String> graphEdgeList;
    @FXML
    private JFXTextArea matrixArea;
    @FXML
    private JFXTextField length;
    @FXML
    private JFXCheckBox autoLayout;

    @FXML
    void drawGraph() {
        Platform.runLater(() -> {
            handleGraphWindow( title: "Graph");
            getGraph(Integer.parseInt(length.getText().trim()));
            prepareGraph(graph);
        });
    }

    @FXML
    void drawKruskalMST() {
        Platform.runLater(() -> {
            handleGraphWindow( title: "Minimum Spanning Tree");
            prepareGraph(Objects.requireNonNull(kruskalMSTAlgorithm()));
        });
    }
```

فیلد گراف برای نگه داشتن گراف وارد شده است، دو فیلد دیگر برای نمایش گراف و درخت پوشای کمینه است که اولی کانتینری برای نمایش گراف است و دومی لیستی از یال هایی که باید در صفحه رسم شود را نگاه می دارد.

فیلد هایی که دارای انوتیشن FXML هستند، عناصر گرافیکی موجود در گرافیک هستند.

متد drawGraph که به دکمه خود بایند شده ابتدا متد handleGraphWindow را صدا میزند تا صفحه دیگری برای نمایش گراف رسم شود، سپس ماتریس همجواری از ورودی خوانده می شود و به فرم گرافی که ما نوشتیم ذخیره می شود(متد getGraph) در نهایت متد prepareGraph فراخوانی میشود و گراف خوانده شده را رسم می کند.

متد drawKruskalMST مشابه متد قبل با این تفاوت که به جای گراف الگوریتم کروسکال را فراخوانی میکند و نتیجه آن که یک حالت خاص گراف (درخت) است را رسم می کند.

```java
private void handleGraphWindow(String title) {
    SmartPlacementStrategy strategy = new SmartCircularSortedPlacementStrategy();
    graphEdgeList = new GraphEdgeList<>();
    graphView = new SmartGraphPanel<>(graphEdgeList, strategy);
    graphView.automaticLayoutProperty.setValue(autoLayout.isSelected());
    Stage stage = new Stage();
    stage.setTitle(title);
    stage.getIcons().add(new Image( s: "file:///" + new File( pathname: "" +
            "src/main/resources/images/icon.png").getAbsolutePath()));
    Scene scene = new Scene(graphView,  v: 800,  v1: 800,  b: false,
            SceneAntialiasing.BALANCED);
    stage.setScene(scene);
    stage.show();
    graphView.init();
}
private void getGraph(int size) {
    String string = matrixArea.getText().replaceAll( regex: "\\D+",  replacement: "-");
    StringTokenizer tokenizer = new StringTokenizer(string,  delim: "-");
    if (tokenizer.countTokens() < size) {
        length.setText("?");
        return;
    }
    graph = new Graph(size);
    for (int i = 0; i < size * size; i++) {
        int weight = Integer.parseInt(tokenizer.nextToken());
        if (weight == 0) {
            continue;
        }
        graph.addEdge(new Edge( from: i / size,  to: i % size, weight));
    }
}
```

متد handleGraphWindow :

صفحه ای که گراف داده شده می خواهد در آن رسم شود را آماده می کند ، برای رسم هوشمند گراف از کتابخانه JavaFX smart graph استفاده شده که فایل jar آن در پوشه lib پروژه موجود است، برای قرار دادن node ها از استراتژی دایروی این کتابخانه استفاده شده که تنها استراژی است که در حال حاضر در آن وجود دارد (استراتژی رندوم نیز وجود دارد که عملا استراتژی محسوب نمی شود)

متد getGraph : اعداد موجود در ماتریس مجاورت را استخراج می کند و به ترتیب هر یال را می سازد (اگر غیر صفر باشد) و همه را اضافه می کند، میدانیم که اینطوری نصف یال ها دوبار اضافه می شوند، ولی چون از hashset استفاده کردیم و متد equals و hashcode را با هوشمندی نوشتیم یال تکراری به آن اضافه نمی شود.

همچنین تعداد رئوس گراف از textbox موجود در view خوانده می شود.

متد Initialize : مشابه متد موجود در نپولیتان اعضای disjoint set را می سازد و پدر هر خانه را در مرحله اول خودش قرار می دهد و بر میگرداند.

```java
private Graph kruskalMSTAlgorithm() {
    Queue<Edge> edges = graph.getEdges() Set<Edge>
            .stream() Stream<Edge>
            .sorted()
            .collect(Collectors.toCollection(LinkedList::new));
    Graph mst = new Graph(graph.getSize());
    Graph.Subset[] subsets = initialize(graph.getSize());
    while (mst.getEdges().size() < graph.getSize() - 1) {
        Edge next = edges.poll();
        if (next == null) return null;
        var p = find(subsets, next.getFrom());
        var q = find(subsets, next.getTo());
        if (p != q) {
            merge(subsets, p, q);
            mst.addEdge(next);
        }
    }
    return mst;
}
private void merge(Graph.Subset[] subsets, int p, int q) {
    int pRoot = find(subsets, p);
    int qRoot = find(subsets, q);
    subsets[pRoot].setParent(qRoot);
}
private int find(Graph.Subset[] subsets, Integer i) {
    if (!subsets[i].getParent().equals(i)) {
        subsets[i].setParent(find(subsets, subsets[i].getParent()));
    }
    return subsets[i].getParent();
}
private Graph.Subset[] initialize(Integer size) {
    var result = new Graph.Subset[size];
    for (int i = 0; i < size; i++) {
        result[i] = new Graph.Subset();
        result[i].setParent(i);
    }
    return result;
```

متد kruskalMSTAlgortihm : ابتدا یال هارا به کمک Stream API Java مرتب کرده و درون یک صف میریزیم و گراف mst را برای نتیجه نهایی می سازیم و Subset هارا نیز می سازیم برای تشخیص دور در گراف، حال شروع به انتخاب یال ها میکنیم، در هر مرحله عضو ابتدای یال ها را بر میداریم (کم هزینه ترین) سپس پدر دو راس موجود در این یال را پیدا می کنیم و با هم

مقایسه می کنیم اگر یکی نبودند یه این معنی است که این دو راس در یک Set نیستند، پس می توان یال انتخابی را به جواب اضافه کرد و نیز این دوSet موجود را یکی کرد، این کار را با یکی کردن پدر آن ها در متد merge انجام می دهیم، همین روال را ادامه می دهیم تا به تعداد یکی کمتر از تعداد رئوس یال انتخاب کرده باشیم تا هیچ راسی جدا نماند و در نهایت یک گراف همبند داشته باشیم و MST مورد نظر ساخته شود.

متد find : برای پیدا کردن پدر یک Set به کار می رود ، آن قدر پدر یک راس را استخراج می کنیم، تا به راسی برسیم که پدر آن خودش است، چنین راسی را در الگوریتم Disjoint Set را راس پدر یک Set می گویند و با آن می توان فهمید آیا دو راس در یک Set هستند یا خیر، اگر پدر یک Set با پدر دیگری برابر باشد به این معنی است که هر دو در یک Set هستند.

متد merge : دو راس را می گیرد و Set های آن دو را یکی می کند، یه این شکل که پدر یک Set را میگیرد این راس را به عنوان پدر راسی میگذارد که به عنوان پدر Set دیگر شناخته می شود، این گونه پدر هر راس در هر کدام ازSet ها یکی می شود و اینگونه نتیجه می شود که همه این رئوس در یک Set قرار گرفته اند.

نکات: با فعال کردن گزینه auto layout نوعی جاذبه (روی یال ها) و نوعی دافعه (روی رئوس) در گراف اعمال می شود وسعی می شود رئوس گراف روی هم نیفتد البته در بعضی از اوقات درست کار نمی کند ونیاز است روی رئوس مختلف یکبار کلیک شود و این جاذبه و دافعه اعمال شود تا راسی روی راس دیگر قرار نگیرد.

زمانی که این امکان خاموش است بدون اعمال هیچ نیرویی می توانید رئوس گراف را جا به جا کنید، البته به احتمال زیاد برخی رئوس روی هم میفتند که باید آن ها را با ماوس گرفته و بکشید تا رئوس که در زیر قرار گرفته اند مشخص شوند.

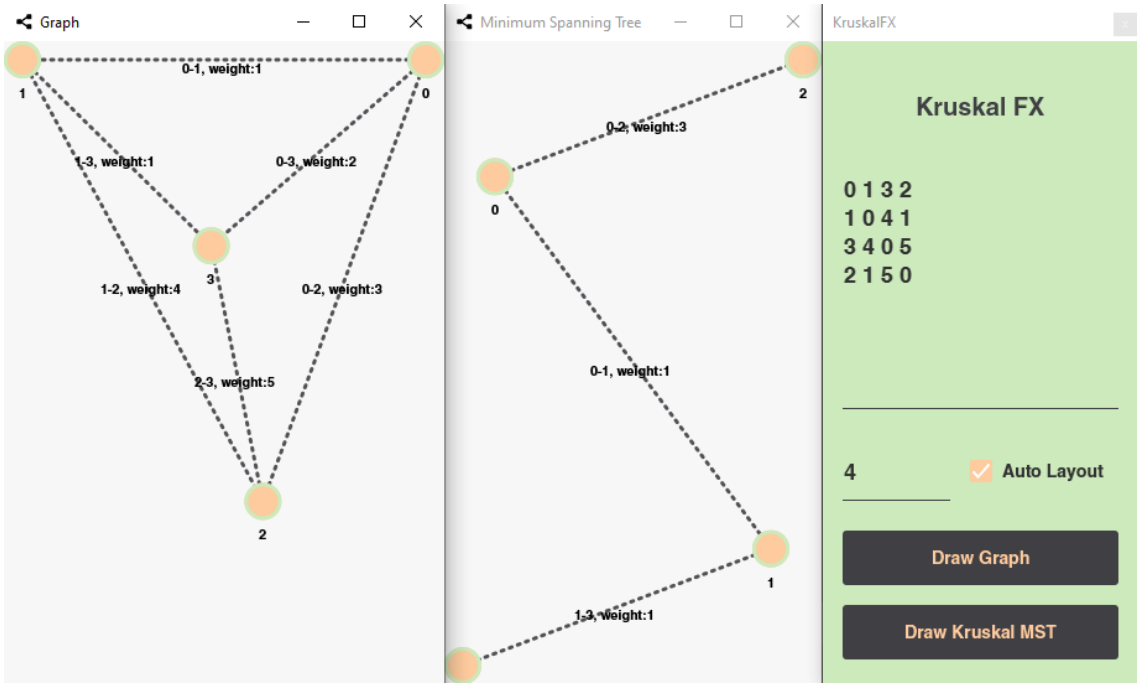
اطلاعات مربوط به جاذبه و دافعه در فایل smartgraph.properties موجود است که برای گراف های کوچکتر کانفیگ شده است، برای گراف های بزرگ تر بهتر است مقدار layout.repulsive-force را تا حدود 5000 کاهش داد تا گراف بهتر دیده شود، این اتوماتیک بودن یکی از خواص کتابخانه smart graph است.

رنگ اجزای مختلف گراف نیز در فایل smartgraph.css موجود است و می توان رنگ قسمت های مختلف را نیز تغییر داد.

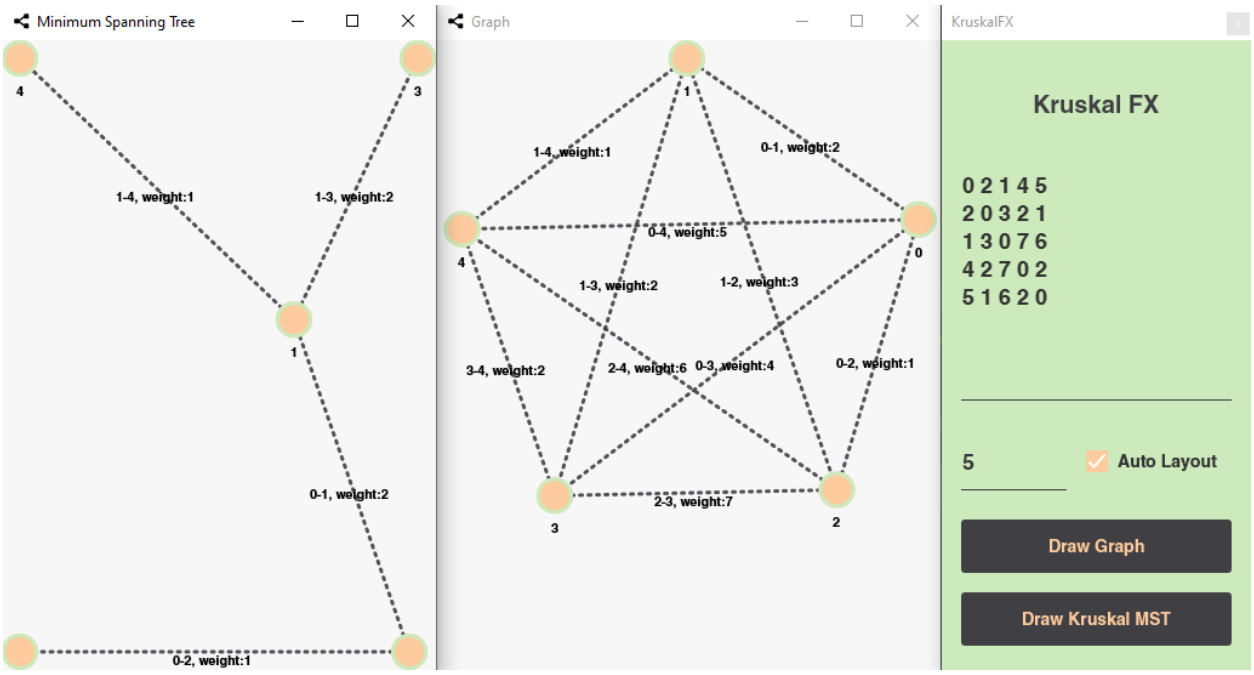
از کتابخانه jfoenix نیز برای UI بهتر (دکمه ها و textArea وTextBox و CheckBox استفاده شده است).

برای اجرا در Intellij کافی است کتابخانه smartgraph موجود در پوشه lib را به پروژه اضافه کنید و پروژه را کامپایل کنید.

نمونه اجرا :



نمونه اجرای دو:




نمونه سوم: