Laboration 10 – Graf

Grundläggande

I uppgift 1 till 4 ska du komplettera klassen WFController. Uppgifterna går i stort ut på att söka vägen mellan olika orter i Skåne. I uppgifterna används ett antal klasser och ett antal filer. Klasserna är:

MapView – Visar en karta. MapView kan visa valda vägar. När ett Map-objekt skapas anger man den bild som ska visas samt koordinater för kartan.

Place(name, position) – Innehåller kartinformation om en ort. Informationen är namn på orten och ortens koordinater på kartan.

Road(from, to, cost, path) – Innehåller ort där vägen startar, ort där vägen slutar, hur lång vägen är och vägens sträckning på kartan.

Position(longitude, latitude) – används för att ange en position på kartan.

GraphSearch – Innehåller klass-metoderna *depthFirstSerach*, *breadthFirstSearch* och *dijkstraSearch* för att söka i en graf.

Graph – En graf

Edge(from, to, weight) – båge i grafen

WFController(graph, map) – klass att komplettera med kod.

Placera klasserna MapView, Place, Road, Position och WFController i paketet ***laboration15***. Placera filerna *skane.jpg*, *places.txt* och *roads.txt* i mappen ***M:\filer***.

Uppgift 1

Första uppgiften går ut på att kontrollera att filerna placerats på korrekt plats.

Exekverar main-metoden i WFController. Nu ska ett fönstret visa sig (se figuren till höger). Det är bilden *skane.jpg* som visas tillsammans med alla Road-objekt som lästs in från filen *roads.txt* (röda linjer på kartan).

Om du inte får körresultatet till höger har du placerat filerna på fel plats (hittas ej av programmet).

Avmarkera raden

```
map.showRoads( roads );
```

och kör programmet på nytt. Nu ska det visa sig en karta utan röda linjer

Uppgift 2

För att kunna söka efter vägen mellan två orter måste du skapa en graf där orterna är noder och vägarna är bågar.

I **WFController** läses orter och vägar in i konstruktorn:

```
ArrayList<Place> places = WFController.readPlaces(placeFile);
ArrayList<Road> roads = WFController.readRoads(roadFile);
```

Din uppgift är att färdigställa metoden

```
makeGraph(ArrayList<Place> places, ArrayList<Road> roads)
```

i klassen *WFController*.

I metoden hittar du kommentarer som berättar vad du ska göra. För att kontrollera att makeGraph verkligen fyller grafen med korrekt information så ska du anropa metoden *graph.printGraph()* direkt efter makeGraph-anropet:

```
makeGraph(places, roads);
graph.printGraph();
```

Du ska få en utskrift som den i bilaga 1 när du kör programmet.

Uppgift 3

Nu är det dags att börja söka efter vägen mellan två orter. Den första varianten av sökning ska använda sig av metoden *GraphSearch.depthFirstSearch*.

Följande gäller för metoden:

Beskrivning:	Metoden ska söka efter en väg från en ort (from) till en annan ort (to). Om metoden hittar en väg mellan orterna ska vägen visas på kartan.
Klass:	WFController
Namn:	***search1***
Parametrar:	En graf och två orter, startort och slutort.
Parameterlista:	(String ***from***, String ***to***)
Retur-värde:	-
Algoritm:	Deklarera variabeln ***path*** av typen *ArrayList<Edge<String>>*
	Deklarera variabeln ***roadList*** av typen *ArrayList<Road>*. Skapa ett objekt av typen *Arraylist<Road>* som du tilldelar variabeln ***roadList***.
	Om ***from*** finns i grafen så
	Anropa metoden *GraphSearch.depthFirstSearch* med argumenten ***graph*** (instansvariabel i klassen WFController), ***from*** och ***to***. Metoden returnerar ett objekt av typen *ArrayList<Edge>*, vilket du ska tilldela variabeln ***path***.
	För varje *Edge*-objekt i ***path*** så
	Hämta Road-objekt ur roads som matchar from och to i Edge-objektet …roads.get(edge.getFrom()+"-"+edge.getTo()) Lägg till *Road*-objektet i arraylisten ***roadList***.
	Anropa metoden *showRoads* i *Map*-objektet med argumentet ***roadList*** så sökresultatet visas på kartan: `map.showRoads( roadList );`

Uppgift 4

Komplettera metoden ***shortestPath*** med kod. Metoden ska fungera på samma sätt som metoden *search1*. Enda skillnaden är att metoden *GraphSearch.dijkstraSearch* ska anropas för sökningen mellan två orter.

Fördjupande

Uppgift 5

Komplettera metoden ***randomSearch*** med kod. Metoden ska slumpmässigt söka sig runt bland orterna tills korrekt ort är funnen. Ungefärlig algoritm:

Om grafen innehåller *from* och grafen innehåller *to*
  Medan *from* inte är sammas som *to*
    Hämta alla bågar i *from*
    Slumpa fram en båge
    Lagra ett *Road*-objekt som motsvarar den framslumpade bågen i en ArrayList
    Ändra *from* till den ort som bågen leder till
  Visa de lagrade *Road*-objekten på kartan

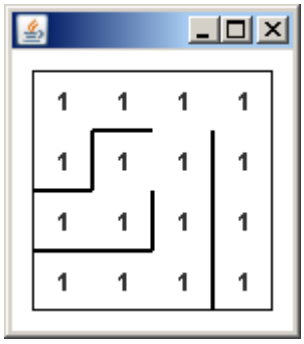
Uppgift 6

Skriv ett program vilket låter användaren mata in två orter och som sedan visar kortaste väg mellan orterna.

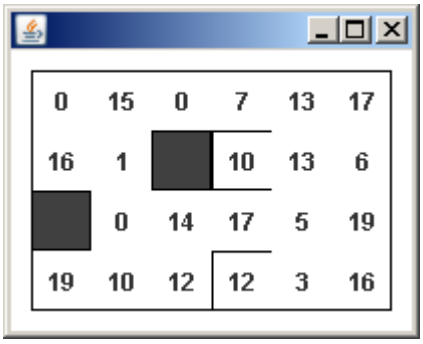
Uppgift 7

Placera klasserna ***Maze***, ***Room*** och ***Laboration15*** i paketet ***laboration15***. Med klassen *Maze* kan man skapa en labyrint i vilken man kan söka efter vägen. Varje del av labyrinten har en vikt vilken kan användas vid sökning. Labyrinten lagras som en graf.

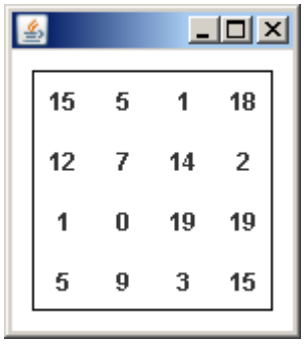
Exempel på labyrinter:



En labyrint där varje del har vikten 1. Väggar gör att man inte kan förflytta sig hur som helst i labyrinten.

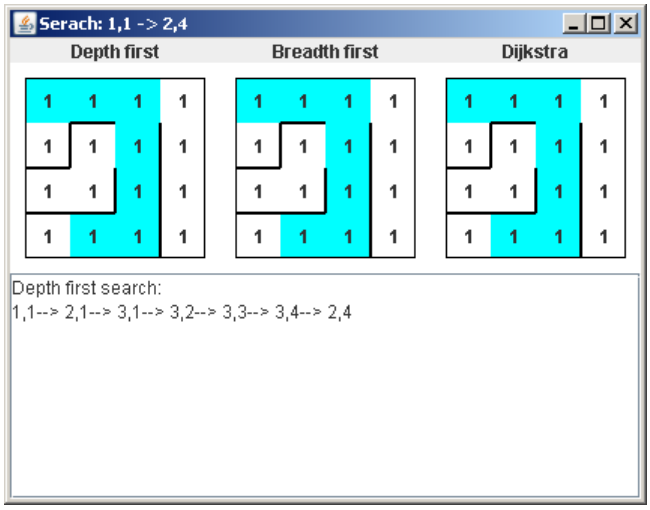


En labyrint med olika vikter i olika delar. En del med väggar åt alla riktingar skuggas



En labyrint som saknar väggar. Man kan förflytta sig hur som helst i labyrinten.

Kör *main*-metoden i *Laboration15*. Ett fönster liknande det på nästa sida kommer att visa sig.



Fönstret visar resultatet av sökning från ruta 1,1 till ruta 2,4. Sökningen sker på tre sätt:
* djupsökning
* breddsökning
* optimerad breddsökning för lägsta vikt

Eftersom det endast går att komma till ruta 2,4 på ett sätt ger samtliga sökalgoritmer samma resultat. Ljusblå rutor visar vägen.

Prova gärna att ändra på startruta och slutruta för sökningen och studera resultatet.

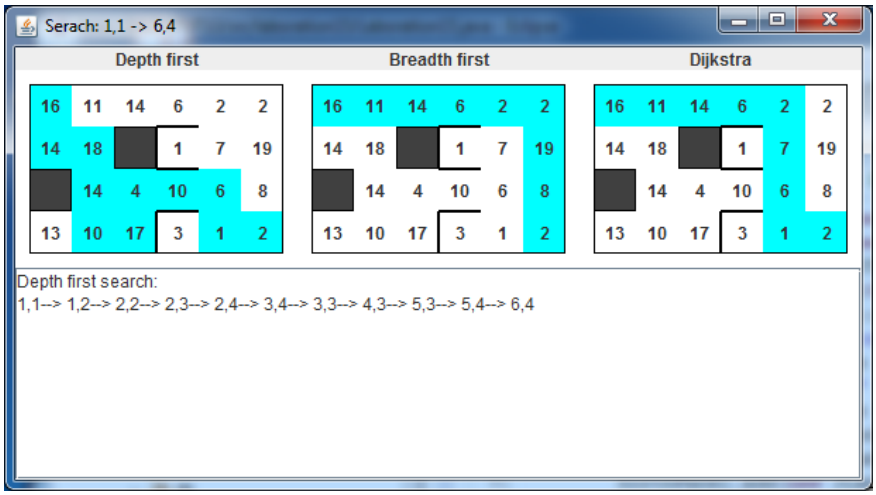
Uppgift 7

Avmarkera de två översta raderna i *main*-metoden i ***Laboration15*** och aktivera i stället rad 3 och rad 4:

```
Maze maze = Laboration15.maze2();
Laboration15.compareSearch(maze, "1,1", "6,4");
```

En något större labyrint skapas med en del väggar. I de skuggade områdena går det inte att komma in. Det går dock att komma ut från dem (endast vägg in).

Kör main-metoden. Ett fönster liknande detta visar sig:



Nu finns det olika vägar att komma till den sökta rutan 6,4. Även om du kör programmet många gånger så kommer djupsökningen och breddsökningen ge samma resultat (grafen byggs på samma sätt varje gång). Men den optimerade sökningen anpassar sig varje gång efter de viktade värdena i ruterna.

Prova gärna att ändra på startruta och slutruta för sökningen och studera resultatet.

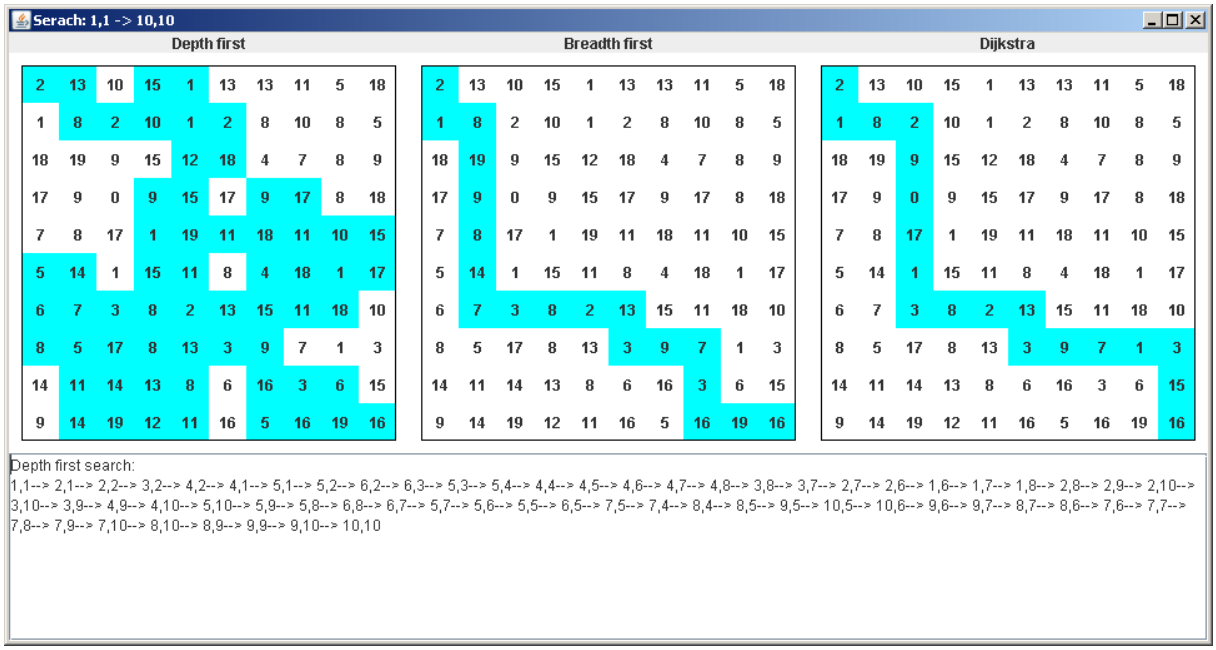
Uppgift 8

Avmarkera raderna som var aktiva i uppgift 7 och aktivera i stället rad 5 och rad 6 i main:

```
Maze maze = Laboration15.maze2();
Laboration15.compareSearch(maze, "1,1", "6,4");
```

En något större labyrint skapas utan väggar.

Kör main-metoden. Ett fönster liknande detta visar sig:



I denna variant av labyrint byggs grafen på olika sätt varje gång. Därför kommer djupsökningen och breddsökningen att variera sin lösning. Men breddlösningen kommer varje gång att innehålla lika många steg för att nå till målet (om målet inte ändras).

Prova gärna att ändra på startruta och slutruta för sökningen och studera resultatet.

Lösningar

Uppgift 2

```
public void makeGraph(ArrayList<Place> places, TreeMap<String,Road> roads) {
    Iterator<Road> values = roads.values().iterator();
    Road road;
    for (Place place : places) {
        graph.addVertex(place.getName());
    }
    while (values.hasNext()) {
        road = values.next();
        graph.addEdge(road.getFrom(), road.getTo(), road.getCost());
    }
}
```

Uppgift 3

```
public void search1(String from, String to) {
    ArrayList<Edge<String>> path;
    ArrayList<Road> roadList = new ArrayList<Road>();
    if (graph.containsVertex(from)) {
        path = GraphSearch.depthFirstSearch(graph, from, to);
        for (Edge<String> edge : path) {
            roadList.add(roads.get(edge.getFrom()+"-"+edge.getTo()));
        }
        map.showRoads(roadList);
    }
}
```

Uppgift 4

```
public void shortestPath(String from, String to) {
    ArrayList<Road> roadList = new ArrayList<Road>();
    if (graph.containsVertex(from)) {
        for (Edge<String> edge : GraphSearch.dijkstraSearch(graph, from, to)) {
            roadList.add(roads.get(edge.getFrom()+"-"+edge.getTo()));
        }
        map.showRoads(roadList);
    }
}
```

Uppgift 5

```
public void randomSearch(String from, String to) {
    Random rand = new Random();
    ArrayList<Road> roadList = new ArrayList<Road>();
    ArrayList<Edge<String>> adjacentList;
    Edge<String> edge;
    if (graph.containsVertex(from) && graph.containsVertex(to)) {
        while( !from.equals(to)) {
            adjacentList = graph.getAdjacentList(from);
            edge = adjacentList.get(rand.nextInt(adjacentList.size()));
            roadList.add(roads.get(edge.getFrom()+"-"+edge.getTo()));
            from = edge.getTo();
        }
        for(Road road: roadList)
            System.out.println(road);
        map.showRoads(roadList);
    }
}
```

```
------------------------------------------------
Åhus
   Edge, from=Åhus, to=Kristianstad, weight=19
   Edge, from=Åhus, to=Simrishamn, weight=55
------------------------------------------------
------------------------------------------------
Malmö
   Edge, from=Malmö, to=Landskrona, weight=44
   Edge, from=Malmö, to=Lund, weight=20
   Edge, from=Malmö, to=Skanör-Falsterbo, weight=31
   Edge, from=Malmö, to=Trelleborg, weight=33
   Edge, from=Malmö, to=Ystad, weight=60
------------------------------------------------
------------------------------------------------
Simrishamn
   Edge, from=Simrishamn, to=Lund, weight=83
   Edge, from=Simrishamn, to=Ystad, weight=39
   Edge, from=Simrishamn, to=Åhus, weight=55
------------------------------------------------
------------------------------------------------
Lund
   Edge, from=Lund, to=Eslöv, weight=21
   Edge, from=Lund, to=Kristianstad, weight=78
   Edge, from=Lund, to=Landskrona, weight=34
   Edge, from=Lund, to=Malmö, weight=20
   Edge, from=Lund, to=Simrishamn, weight=83
------------------------------------------------
------------------------------------------------
Skanör-Falsterbo
   Edge, from=Skanör-Falsterbo, to=Malmö, weight=31
   Edge, from=Skanör-Falsterbo, to=Trelleborg, weight=24
------------------------------------------------
------------------------------------------------
Ystad
   Edge, from=Ystad, to=Malmö, weight=60
   Edge, from=Ystad, to=Simrishamn, weight=39
   Edge, from=Ystad, to=Trelleborg, weight=47
------------------------------------------------
------------------------------------------------
Helsingborg
   Edge, from=Helsingborg, to=Hässleholm, weight=79
   Edge, from=Helsingborg, to=Höganäs, weight=21
   Edge, from=Helsingborg, to=Landskrona, weight=28
   Edge, from=Helsingborg, to=Ängelholm, weight=30
------------------------------------------------
------------------------------------------------
Kristianstad
   Edge, from=Kristianstad, to=Hässleholm, weight=32
   Edge, from=Kristianstad, to=Lund, weight=78
   Edge, from=Kristianstad, to=Åhus, weight=19
------------------------------------------------
------------------------------------------------
Trelleborg
   Edge, from=Trelleborg, to=Malmö, weight=33
   Edge, from=Trelleborg, to=Skanör-Falsterbo, weight=24
   Edge, from=Trelleborg, to=Ystad, weight=47
------------------------------------------------
------------------------------------------------
Höganäs
   Edge, from=Höganäs, to=Helsingborg, weight=21
   Edge, from=Höganäs, to=Ängelholm, weight=22
```

```
-----------------------------------------------
-----------------------------------------------
Landskrona
   Edge, from=Landskrona, to=Eslöv, weight=32
   Edge, from=Landskrona, to=Helsingborg, weight=28
   Edge, from=Landskrona, to=Lund, weight=34
   Edge, from=Landskrona, to=Malmö, weight=44
-----------------------------------------------
-----------------------------------------------
Hässleholm
   Edge, from=Hässleholm, to=Eslöv, weight=56
   Edge, from=Hässleholm, to=Helsingborg, weight=79
   Edge, from=Hässleholm, to=Kristianstad, weight=32
   Edge, from=Hässleholm, to=Ängelholm, weight=74
-----------------------------------------------
-----------------------------------------------
Ängelholm
   Edge, from=Ängelholm, to=Helsingborg, weight=30
   Edge, from=Ängelholm, to=Hässleholm, weight=74
   Edge, from=Ängelholm, to=Höganäs, weight=22
-----------------------------------------------
-----------------------------------------------
Eslöv
   Edge, from=Eslöv, to=Hässleholm, weight=56
   Edge, from=Eslöv, to=Landskrona, weight=32
   Edge, from=Eslöv, to=Lund, weight=21
-----------------------------------------------
```