Datastrukturuppgift 2

Syfte

- Programmeringsträning
- Graf och heap
- Dijkstras algoritm, kortaste vägar

Instruktioner

För godkänt resultat *ska* varje uppgift redovisas för laborationshandledaren muntligen och godkännas av denne. Uppgifterna är en stor del av examinationen på kursen så lägg mycket tid på att utföra dem noggrant och med god struktur. Det är inte bara lösningens funktion som bedöms utan också hur du gått tillväga för att lösa uppgiften. Onödigt komplicerade eller ostrukturerade lösningar underkänns även om de fungerar.

Uppgifterna ska utföras *enskilt*. Givetvis får du fråga dina kurskamrater om hjälp ifall du har frågor om någonting men uppgifterna ska lösas och redovisas på egen hand.

Uppgift

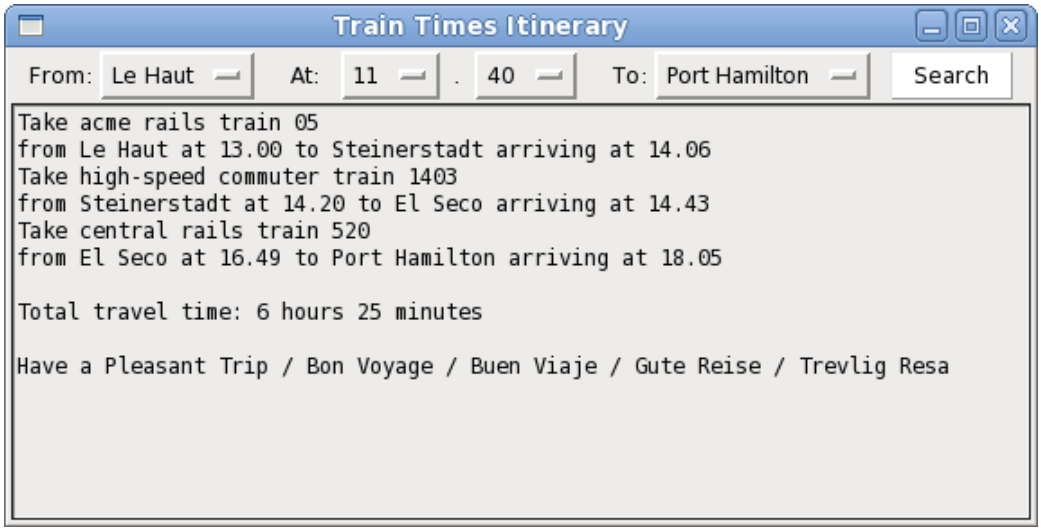
I det lilla bergslandet *Steinerland* opererar fem järnvägsbolag, *Acme rails*, *Central rails*, *Western rails*, *Transmontagnas* samt det statliga lokaltrafikbolaget. Efter långa förhandlingar har bolagen kommit överens om en gemensam enhetstariff som gör det möjligt att åka från start- till slutdestinationen med alla nödvändiga tågbyten på samma biljett.

I och med detta vill järnvägsbolagen ta fram en tillämpning som kan användas av de resande för att ta reda på vilka tågsträckor man ska resa för att komma fram till slutdestinationen som fort som möjligt. Du har fått uppdraget att utveckla en första prototyp av tillämpningen. Efter att du producerat ett antal grafiska mock-ups har kunden bestämt sig för utseendet på prototypen enligt figur 1.

Slutför uppgiften och producera en prototyp som beräknar kortaste restiden mellan start- och slutdestination givet ett klockslag då man vill påbörja resan. Det är inte nödvändigt att lösa uppgiften grafiskt men interaktionen ska följa samma mönster som i den grafiska mock-upen även om man får mata in uppgifterna för hand.

Genomförande

Uppgiften är en tillämpning på Dijkstras algoritm men kommer också att kräva korrekt hantering av klockslag och tidpunkter vilket är arbetsamt men nyttig träning.



Figur 1: Prototypens grafiska utseende. Start-, slutdestination och klockslag väljs från dropdown-listor för enkel interaktion. Resvägen skrivs ut i textfältet undertill i prototypen men kommer i den slutliga tillämpningen att skrivas ut på en pappersremsa tillsammans med biljetten.

Lös uppgiften i små steg så blir det inte så mycket på en gång.

1. Börja med att implementera en nodklass innehållande all den information som du kan tänkas behöva till din nod. Du kan behöva lägga till attribut till denna under arbetets gång.

2. Implementera sedan stubbarna till en "viktad riktad graf"-ADT enligt nedan:

WeightedDirectedGraph
+ insertNode(Node node, String key)
+ findNode(String key) : Node
+ deleteNode(Node node)
+ insertArc (Node from, Node to, int weight)
+ getNeighbours(Node node) : Node`[]`
+ getWeight(Node from, Node to) : int // undefined if nodes are not neighbours
+ shortestPath(Node from, Node to) : WeightedDirectedGraph

   Även i denna uppgift är det snyggast att implementera grafstrukturen som en klass `WDGimpl` som ärver från ett interface `WeightedDirectedGraph` som i sin tur specificerar signaturerna för metoderna i klassdiagrammet. Huvudprogrammet arbetar sedan enbart med `WeightedDirectedGraph`-objekt.

3. Implementera sedan de enkla operationerna (de sex första) i `WDGimpl`. Du får själv välja om du vill representera grafen med en närhetsmatris eller en närhetslista. För att snabbt

kunna söka efter en nod i strukturen med `findNode` kan du behöva en sekundär struktur (t.ex. ett lexikon) som håller reda på vilken nod som har vilken nyckel.

Notera att `deleteNode` även måste ta bort de bågar som är anslutna till den, både dem som är riktade från noden och dem som är riktade till noden.

4. Konstruera därefter första delen av huvudprogrammet som läser in en tidtabell och konstruerar grafen utifrån denna. Tidtabellen följer med uppgiften i filen `timetable.tbl` och har formatet:

```
high-speed commuter train 0501
Steinerstadt:   05.40 - El Seco:        06.03
El Seco:        06.05 - Neubergstadt:   06.39
Neubergstadt:   06.41 - Steinerstadt:   06:58

high-speed commuter train 0601
Steinerstadt:   06.00 - El Seco:        06.23
El Seco:        06.25 - Neubergstadt:   06.59
Neubergstadt:   07.01 - Steinerstadt:   07:18

...
```

   Dvs. ett tågnamn följt av en eller flera sträckor som tåget följer. Sträckorna anges på varsin rad enligt formatet

   <stad>: <klockslag> - <stad>: <klockslag>

   En tom rad anger att en tidtabell för ett nytt tåg börjar.

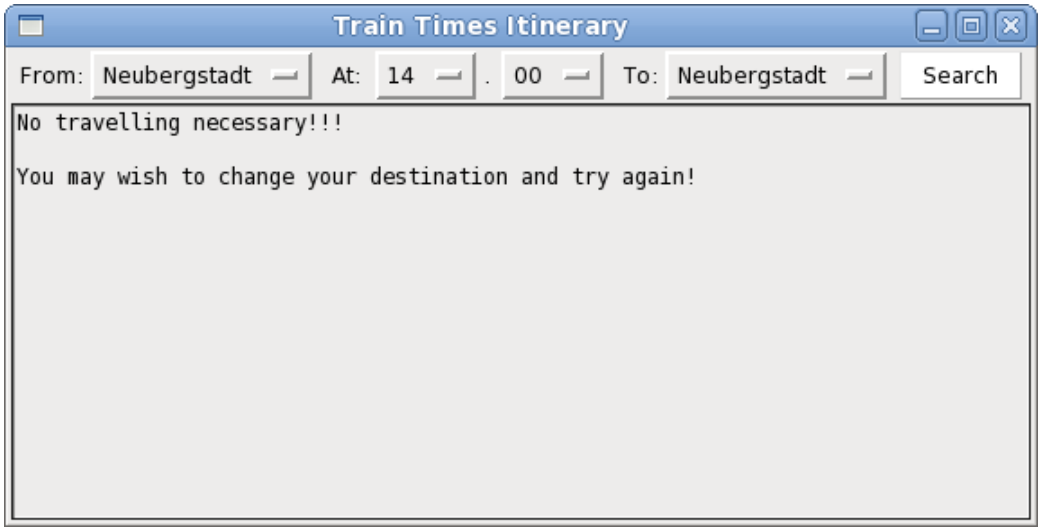
5. Man kan representera tidtabellen i grafen på många olika sätt och att få detta att fungera är nog den svåraste delen i uppgiften, att välja en bra representation av informationen i grafen, så lägg tid på att göra denna bra. Det sparar enormt mycket tid och felrättning i det fortsatta arbetet.

   Den enklaste lösningen är kanske att låta en nod i grafen representera en stad vid ett visst klockslag. Vikten hos en båge som sammanbinder två noder blir då naturligt antalet minuter mellan nodernas tidsangivelser. Den interna representationen av en stad blir då i form av ett kluster av noder som representerar staden vid alla möjliga klockslag.

6. Implementera en heap som du kan använda som prioritetskö till Dijkstras algoritm. Med ovanstående representation räcker det med plats för 1500 noder.

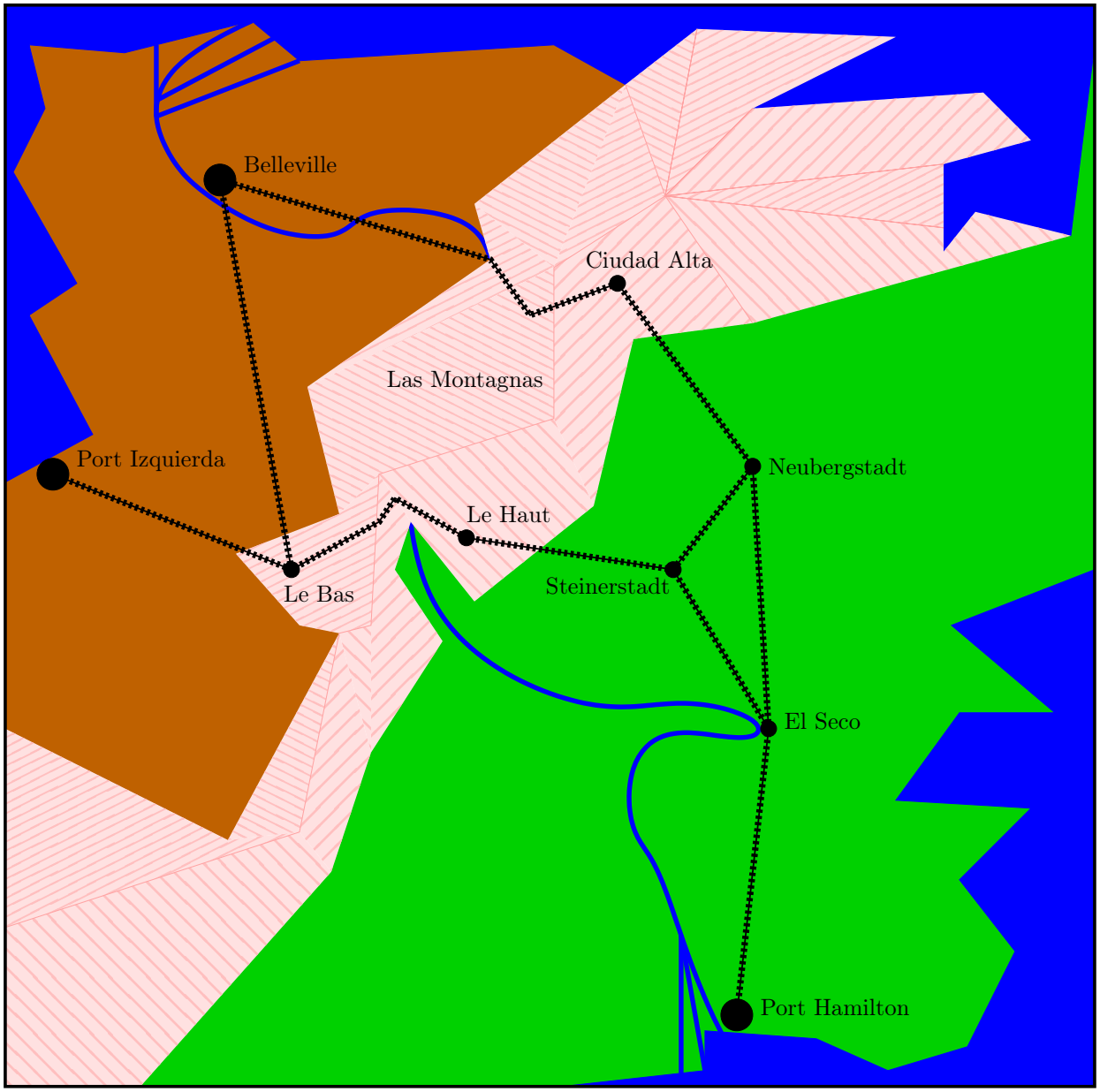
7. Implementera grafoperationen `shortestPath` med Dijkstras algoritm i `WDGimpl`. Notera att operationen ska returnera en `WeightedDirectedGraph`, dvs. den graf som representerar den kortaste vägen från startnoden till slutnoden. Huvudprogrammet kan traversera vägen genom successiva anrop till `getNeighbours` (i vägen finns då högst en granne) och få vikten på varje sträcka med operationen `getWeight`.

Bygg också på interaktionsdelarna i huvudprogrammet så att interaktionen med grafen kan göras enligt kundens önskemål. Notera att programmet ska hantera felaktiga indata på ett kontrollerat sätt, t.ex.



Redovisning

Redovisningen av uppgiften görs muntligen till laborationshandledaren. Programfiler och körningsexempel ska finnas tillgängliga vid redovisningen.



Figur 2: Karta över städerna i Steinerland.