Artificiell intelligens, KV och ID, HT-08
Laboration 2: Sökning
Preliminär t.o.m. 10 okt. kl. 08:00

7 oktober 2008

1 Introduktion

Denna inlämningsuppgift går i huvudsak ut på att givet en karta över några platser i Umeå konstruera en datatyp som representerar denna, samt implementera ett antal olika sökalgoritmer för att hitta de bästa vägarna mellan dessa platser. Vad som påverkar den bästa vägen är inte bara avståndet mellan platserna, utan även hastighetsbegränsningarna som gäller på vägarna som binder dem samman. Viktiga aspekter på sökningar av det slag som beskrivits ovan är minneskonsumtion och hastighet. Informerade sökningar försöker åstadkomma en så billig sökning som möjligt genom att använda problem-specifik kunskap (till exempel att kortaste avståndet mellan två punkter är en rät linje) på problemet. På så sätt kan antalet potentiella lösningar som behöver sökas igenom minskas, vilket resulterar i en snabbare och mindre minneskrävande sökning.

Laborationen ska lösas i grupper om 2 personer.

2 Syfte

Syftet med uppgiften är att:

- Ge en inblick i hur man givet ett problem i verkliga livet kan överföra detta till en representation i datorn.
- Få en förståelse för sökalgoritmer i praktiken och hur heuristik kan användas för att minska kostnaden för en sökning.
- Ge inblick i ett antal vanliga sökalgoritmer och förstå skillnaderna dem emellan.
- Öva er färdighet i rapportskrivning.
- Träna på att skriva välkommenterad och tydlig kod.

3 Beskrivning

Inlämningsuppgiften består egentligen av två delproblem. Den första deluppgiften går ut på att konstruera en datastruktur som representerar kartan och koordinatsystemet, se Sektion 3.1 och 3.2. Den andra delen handlar om att implementera följande sökalgoritmer för att hitta bästa rutten mellan två platser på kartan.

- Bredden först
- Djupet först
- A*
- Greedy Search

Tänk på att ni måste ta hänsyn till både avståndet och hastighetsbegränsningen. Slutligen ska ni skriva ett testprogram som evaluerar de olika sökrutinerna för olika uppgifter. Programmet ska göra tydliga utskrifter av hur sökalgoritmerna traverserar datastrukturerna (får ni tid över kan ni göra en grafisk implementation av detta).

De olika sökalgoritmerna är byggstenar i så kallad *Klassisk AI*, och själva uppgiften är väldigt aktuell i t.ex. GPSer med ruttplanering.

Kapitel 3 och 4 i kursboken hanterar oinformerad och informerad sökning.

3.1 Karta

Programmet ska kunna använda en godtycklig karta specificerad i XML-format. Kartan byggs upp av ett antal platser eller *noder*. Varje nod har en x- och y-koordinat, samt ett unikt *id*. Koordinatsystemet beskriver de olika platsernas inbördes placering utifrån referenssystemet RT-90[1]. Mellan varje nod går dessutom vägar som har en viss hastighetsbegränsning.

I Figur 1 visas en karta över Umeå. De första raderna i XML-filen som bygger upp kartan visas i Figur 2.

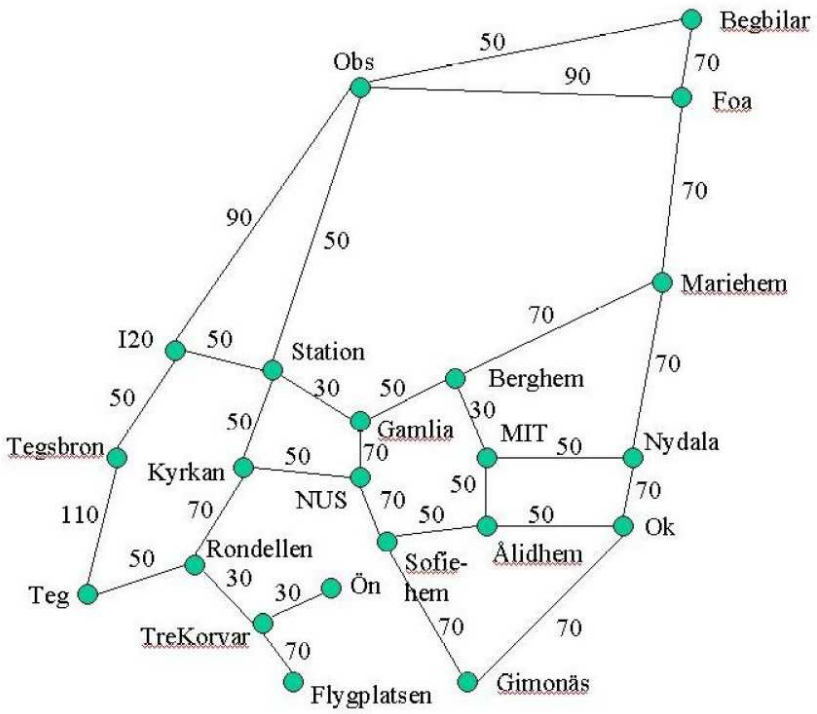
Hela XML-filen för kartan över Umeå finns att ladda hem från kursens hemsida `http://www.cs.umu.se/kurser/5DV063/HT08/utdelat/umea_map.xml`. Denna karta kan kan användas för testning, men er lösning måste klara en generell karta specificerad enligt samma format.

3.2 Implementation

Lösningen ska implementeras i Java. Vi rekommenderar att ni använder Eclipse[2] som utvecklingsmiljö, det är en kraftfull IDE för Java som är vanlig på många företag. Dessutom är den gratis och finns installerad i alla labb. Om ni så önskar är det tillåtet att använda andra utvecklingsverktyg.

[1] Mer info om RT-90 finns på RT-90 `http://www.lantmateriet.se/templates/LMV_Page.aspx?id=4766`

[2] Eclipse finns att ladda hem från `http://www.eclipse.org/`



Figur 1: Karta över Umeå. De nummer som står på strecken som binder samman noderna är hastighetsbegränsningar.

```
<?xml version='1.0' encoding='ISO-8859-1' ?>
<map>
    <node id="Teg" x="1720099" y="7072732">
        <road to="Rondellen" speed="50" />
        <road to="Tegsbron" speed="110" />
    </node>
    <node id="Tegsbron" x="1720359" y="7074215">
        <road to="Teg" speed="110" />
        <road to="I20" speed="50" />
    </node>
    .
    .
    .
</map>
```

Figur 2: Exempel på XML-struktur som bygger upp kartan.

Er implementation av sökalgoritmerna ska ärva klassen *MapSearcher*. *MapSearcher* har fyra abstrakta metoder vilka måste överlagras och implementera nödvändig funktionalitet:

setMap Specificerar kartan. Denna metod anropas alltid före sökning och ska lagra kartan som ett privat attribut i klassen. Hur kartan representeras är upp till er att bestämma.

greedySearch Utför sökning med Greedy Search.

aStar Utför sökning med A*.

breadthFirst Utför sökning med bredden först.

depthFirst Utför sökning med djupet först.

Metoderna greedySearch, aStar, breadthFirst och depthFirst ska returnera en sträng med namnen på de noder som besöks på vägen från start till mål, alltså resultatet av sökningen. Varje nod-id ska avskiljas med ett komma. Exempelvis: *Teg, Rondellen, Kyrkan, Nus* representerar vägen från Teg till Nus. Dessutom ska programmet skriva ut information om vilka noder som algoritmen expanderar under sökningen, även då de inte ingår i den slutgiltiga rutten.

MapSearcher har en metod vid namn loadXmlMap som redan är implementerad. Denna laddar en XML-fil och representerar den som ett DOM-dokument. DOM står för *Document Object Model* och är ett generellt sätt att representera XML dokument som objekt. Således behöver ni inte läsa XML-filen som ett textdokument, utan kan i stället plocka ut informationen i dokumentet genom att läsa av attribut på respektive DOM objekt. Ett exempel visas i Algoritm 1. Koden skriver ut attributet x på andra elementet i dokumentet *doc*. Om *umea_map.xml* används kommer resultatet bli x-koordinaten för Tegsbron, eftersom den noden ligger på plats två i dokumentet.

Algorithm 1 Exempel på användning av DOM dokument.

```
List nodes = doc.getRootElement().getChildren();
System.out.print(((Element)nodes.get(1)).getAttributeValue("x"));
```

Implementationen av DOM som vi kommmer använda här heter *JDOM*, vilket leder till att *jdom.jar* måste bifogas vid kompilering av ert program. Information om hur DOM används finns på JDOMs hemsida `http://www.jdom.org/`.

Koden till MapSearcher finns att ladda hem från kursens hemsida `http://www.cs.umu.se/kurser/5DV063/HT08/utdelat/lab2kod.zip`. Här inkluderas även en fil vid namn *SearchInterface.java*. Denna klass kan ni om ni vill använda för att testköra programmet. Information om hur SearchInterface används hittar ni i dokumentationen för lab 2 `http://www.cs.umu.se/kurser/5DV063/HT08/assignments/lab2doc/`.

4 Rapport och källkod

Er lösning ska finnas tillgänglig från ert konto. Eftersom labbrättaren använder script för att rätta måste koden placeras i `~/edu/ai/lab2/`. Kom ihåg att sätta lämpliga rättigheter så att labbrättaren kommer åt filerna. All källkod ni skrivit ska vara kommenterad med kommentarer som följer JavaDoc standard. Generera även en JavaDoc-sida och placera under `~/public_html/ai/lab2/doc`.

Rapporten ska innehålla följande:

Framsida: Kursens namn och termin, uppgiftsnummer, ditt namn och användar-id, handledarnas namn samt inlämningsdatum.

Problemspecifikation: Beskriv med egna ord vad uppgiften gick ut på. Är det någonting som varit oklart och ni gjort egna tolkningar så beskriv dessa.

Användarhandledning: Förklara var programmet ligger samt hur man startar och använder det.

Algoritmbeskrivning: Beskriv de algoritmer som inte kan anses självklara (beskriv hellre för mycket än för litet).

Begränsningar: Vilka problem och begränsningar har din lösning av uppgiften?

Reflektioner: Var det något som var speciellt krångligt? Vilka problem uppstod och hur löste ni dem? Allmänna synpunkter.

Testkörningar: Noggranna testkörningar där man ser att programmet fungerar som det ska. Följande problem måste finnas med:

- Tegsbron – Nydala
- Flygplatsen – Begbilar
- MIT – Flygplatsen
- Nydala – Gamlia
- Teg – Foa

Testkörningarna ska täcka alla algoritmer och vara välkommenterade. För såväl greedy som bredden och djupet först ska en väg per problem redovisas. För A* ska två vägar, den **kortaste** och den **snabbaste**, finnas med.

Diskussion: Redogör för de olika sökalgoritmernas för- och nackdelar, samt hur de presterade på den givna uppgiften.

Källkod: Bifoga utskriven källkod. Rekommenderat är att ni skriver ut den med kommandot:

```
a2ps -T4 -Afill <filnamn1> <filnamn2> ... | lpr -P<skrivare>
```

Mer information om hur och varför man skriver en labrapport finns på cs allmänna labspecifikation `http://www.cs.umu.se/information/misc/labspec.html`. Sista inlämningsdatum står i schemat.

Lycka till!