Datavetenksap

Mathias Andersson

Henrik Bäck

DAV B03: Data Strukturer & Algoritmer

Höstterminen 2006: Laboration 1 (20p)

Agreement

This report is submitted in partial fulfillment of the requirements for the course DAV B03, Data Structures & Algorithms. All material in this report which is not my own work has been identified.

This document describes the 20p course project (Lab 1) for the course DAV B03, Data Structures and Algorithms, to be submitted by the given deadline.

Please note

1. This lab project may give a maximum of 20 points for the course DAV B03, Data Structures and Algorithms.
2. Students who have submitted this lab report are given exemption from answering questions worth 10p in the examination (40p) **except for the questions on trees, which are obligatory**. **This applies only for students who took the course during the Fall Term 2006 or later.**
3. The report body should be at least 10 to 15 pages of single spaced A4 including figures and tables. I.e. the title page, Agreement, Abstract, Table of Contents, List of Figures and List of Tables are **not** counted as part of the 10 to 15 pages.
4. By signing the report, the student has indicated that he/she agrees to these conditions.

Mathias Andersson

Henrik Bäck

Date Submitted: Friday, 27 October 2006

Sammanfattning

Projektets mål har varit att skapa en modell av Paris Tunnelbanesystem där det skall vara möjligt att hämta information om stationer, linjer och få en resväg. Modellen som systemet är byggt kring baseras på att en modell för linjegrafen och en för stationsgrafen skapas. Dessa lagras med hjälp av adjacency-listor och adjacency-matriser. Ett antal frågeställningar har ställts upp. Alla dessa frågeställningar har inte systemet lyckats besvara med den aktuella modellen. Med hjälp av utvalda algoritmer för det aktuella ändamålet kan dock de flesta av dessa frågeställningar besvaras. Hur reser man från en station till en annan? Hur säkert är tunnelbanesystemet – vilka är de svaga punkterna? Svaret på dessa frågeställningar samt några ytterligare till har besvarats och systemets svaga punkter kan presenteras med hjälp av algoritmer som kan appliceras på grafen. Algoritmer för att avgöra en resväg mellan två stationer är även presenterade.

Abstract

The goal of this project was to create a model of Paris Subway system, where it should be possible to receive information about stations, lines and travel-ways. The model on which the system is built is based on one graph containing the lines and another one containing the stations. These graphs is represented by adjacency-lists and adjacency-matrises. The project was presented a couple of questions which to answer. The current model has not been able give answer to all these questions. Though using a few for the purpose chosen algorithms most of the questions have been answered. How to travel from a station to another? How secure is the subway-system -which is the systems weak-points? These and a few other questions have been answered. The systems weak-points is presented by using a few algorithms applied to the graph. Also algorithms for finding the travel-way between stations is presented.

Innehåll

Förteckning av figurer

Förteckning av tabeller

Inga tabeller i dokumentet.

1 Introduktion

I den här rapporten beskrivs hur datormodellen för Paris tunnelbana har byggts upp. En beskrivning av generell modellering finns samt en ingående beskrivning av modellen för Paris tunnelbana finns presenterat. En översikt över vilka algoritmer som använts samt en diskussion om systemets säkerhet, för- och nackdelar samt tekniker för att finna en väg mellan två stationer.

2 Generell modellering

När det gäller att abstrahera verkligheten med hjälp av ett datorsystem är viktigt att göra en korrekt modell. Eftersom verkligheten är väldigt komplex går det inte alltid att göra en total avbild. Istället är det att föredra att göra en förenklad modell av verkligheten och på så sätt få med de delar som är vitala för funktionen av modellen.

Det är även mycket viktigt att avgränsa vad man vill modellera eftersom mycket av den information som finns i verkligheten inte är relevant. Genom att göra en modell kan man även identifiera problem som finns och hur man tänkt lösa dem.

Modellen får ett antal entiteter som kan liknas med de objekt som modelleras. En stor del av modelleringsarbetet går ut på att finna de korrekta entiteterna i systemet med avseende på vad modellen skall nyttjas för. Varje entitet har ett antal egenskaper som beskriver dessa och entiteterna kan också relatera till varandra på ett eller flera sätt. Även relationerna mellan entiteterna är beroende av i vilket ändamål som modellen skall nyttjas.

En samling med entiteter med en gemensam egenskap kallas för en samling. En ren samling är ej möjlig att representera i dagens datorsystem. I sådana fall måste denna samling representeras av en lämplig datatyp.

3 Modellbeskrivning: Paris tunnelbana

Utifrån Paris tunnelbana [3] har information kring tunnelbanans topologi hämtats. Det är också från denna som systemets begränsningar och funktionalitet har satts. Det har antagits att det inte existerar några ringlinjer (linjer där ingen start eller salutstation kan bestämmas) samt att det inte finns några dubbletter på namn av stationer och linjer.

3.1 Generell beskrivning av tunnelbanan

När tunnelbanan modellerats har det utgåtts ifrån att det bästa sättet att modellera den är att modellera linjer och stationer var och en för sig. Varje graf, stationer och linjer, representeras av två olika implementationer i systemet.

Figur 1 – Översikt av modellen av tunnelbanan

Som kan ses i figur 1 finns två representationer för stationsgrafen. I adjacency-list representationen lagras information om alla stationer. På varje station i listan lagras även information om vilka linjer denna befinner sig på samt vilka andra stationer som finns i direkt anslutning till densamma samt information om en unik identifikationsreferens. Vad beträffar adjacency-matrix representationen lagrar denna endast vilka stationer som har en direkt anslutning.

Det finns, som synes, även två representationer av linjegrafen. Samma här, som för stationsgrafen, är att de representeras av en adjacency-list och en adjacency-matrix. I adjacency-list för linjerna lagras information om linjen, vilka linjer som korsar densamma och vilka stationer som befinner sig på linjen samt information om en unik identifikationsreferens. Om det är möjligt lagras även information om en startstation för linjen. Detta görs endast om linjen är hel och har minst en station som kan betraktas som änd- eller startstation. I adjacency-matrix finns enbart information om vilka linjer som skär varandra.

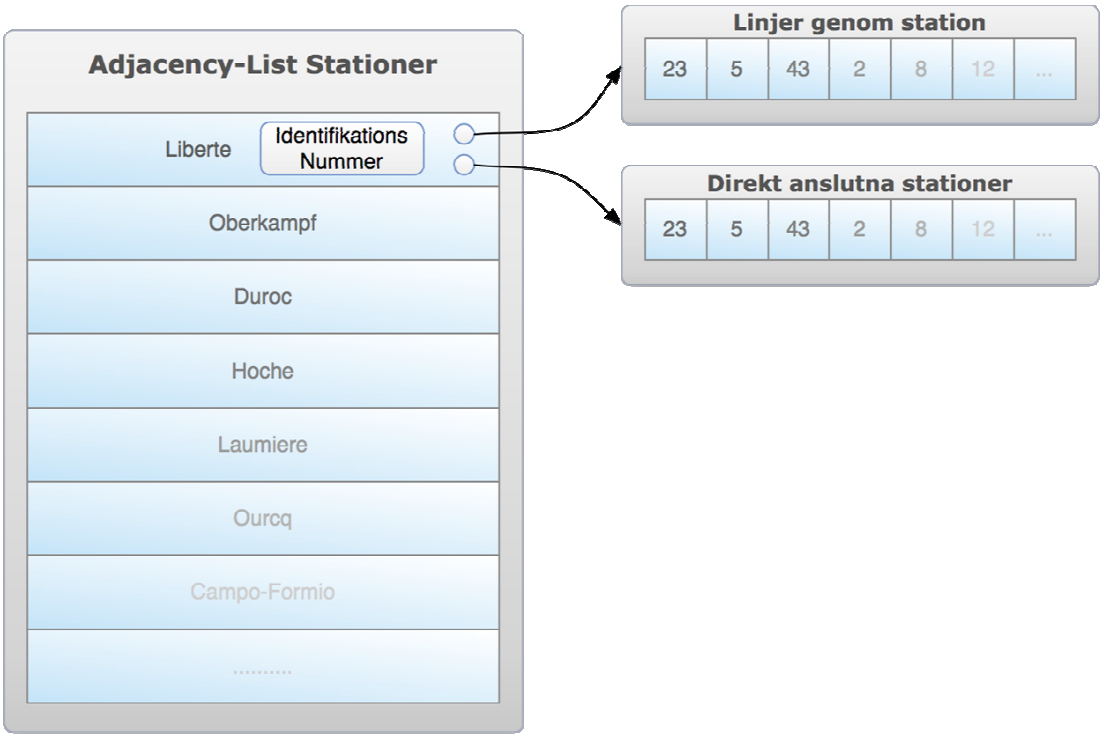
Ingen av representationerna av grafen för stationer eller linjer hanterar verkliga stationsnamn. För att kunna göra modellen användbar för en människa är detta ett krav. Därför finns det två stycken tabeller för att slå upp stationsnamn respektive linje namn. Varje station och linje har ett unikt identifikationsnummer. Detta används förutom för att relatera stationer och linjer för att hämta namn på de samma. Som synes i modellen, figur 1, kan man finna en uppslagningstabell för stationer och en för linjer.

3.2 Detaljerad beskrivning

3.2.1 Stationsgrafen

Grafen över stationer talar enbart om hur stationer relaterar till varandra. I själva grafen saknas information om vilka linjer som befinner sig i systemet. När det gäller adjacency-list representationen används en lista över stationer i systemet. För varje station i listan finns information om vilka andra stationer som det finns en direkt anslutning till. Det finns också, som tidigare nämnt, information om vilka linjer som passerar den aktuella stationen. Varje station i denna lista innehar också ett identifikationsnummer. Detta nummer används för att kunna slå upp ett namn på stationen samt vid lagring av direkta anslutningar till andra stationer. På så sätt behövs inga namn på stationer blandas in i själva grafen.

Som komplement till adjacency-list finns en adjacency-matrix för stationerna. I denna matris kan man enbart få reda på vilka stationer som har en direkt anknytning till varandra. Detta är mycket användbart när man enbart vill ta reda på om två stationer har en direkt anslutning. Även i denna lista används enbart identifikationsnummer för att representera stationerna.

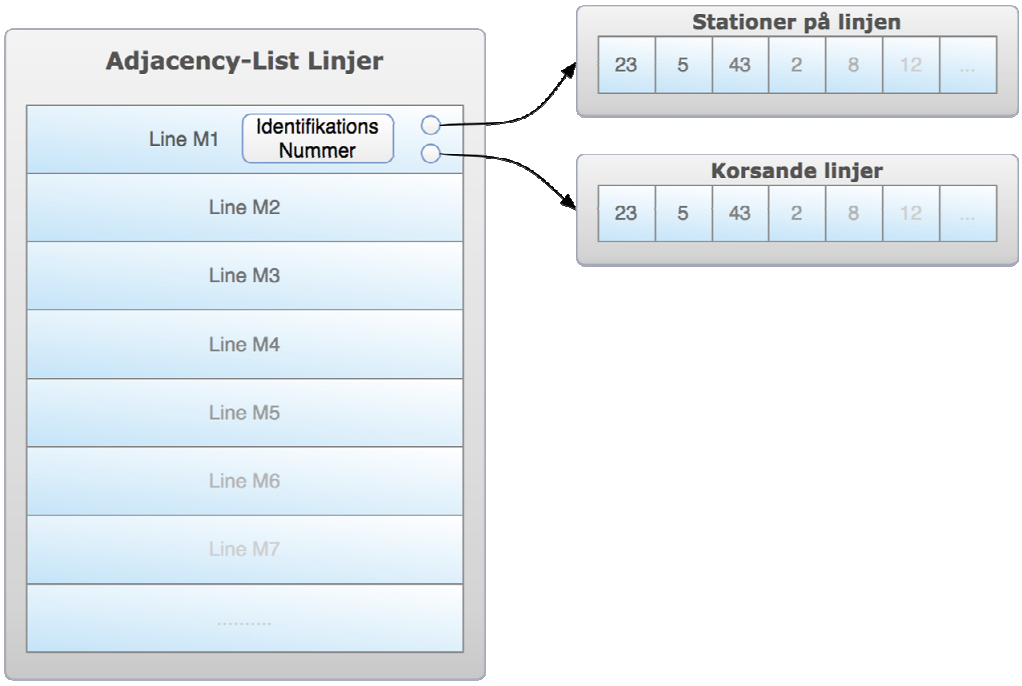


Figur 2 – Stationsgrafen modelleras på detta sätt

3.2.2 Linjegrafen

Grafen över linjer talar enbart om vilka linjer i systemet som skär varandra. På detta sätt får man ifall det finns en anslutning mellan två linjer. Användandet av denna information är väldigt begränsat i modellen eftersom den inte har varit med från början. Det finns en adjacency-list representation för linjegrafen. För varje linje i listan finns information om vilka linjer som korsar densamma samt information om vilka stationer den har. Dessutom finns ett unikt identifikationsnummer för linjen. Detta nummer används för att slå upp ett namn för linjen och för att representera linjen i grafen.

Adjacency-matrix har samma syfte som adjaceny-list för linjer med den stora skillnaden att matrisen enbart talar om vilka linjer som korsar varandra. Här finns ingen extra information att hämta. Även här används identifikationsnummer för att representera linjer.



Figur 3 – Linjegrafen modelleras på detta sätt

3.2.3 Uppslagstabeller

Som tidigare nämnt används uppslagstabellerna för att sätta verkliga namn på stationer eller linjer i grafen. Det finns en separat uppslagningstabell för linjer samt en för stationer. Dessa har, var och en för sig, unika identifikationer uppslagna mot namn. Funktionalitet för att hämta namn med avseende på identifikation samt identifikation med avseende på namn finns. Båda uppslagstabellerna är identiskt uppbyggda och enda skillnaden är att de representerar olika delar av systemet.

Uppslagstabellen bygger enkelt på en lista där alla namn och identifikationer sparas för att kunna matchas vid en förfrågan.

Uppslagstabell Stationer

ID	Namn
49	Odeon
203	Oberkampf
25	Plaisance
46	Vaneau
53	Jussieu
195	Ourcq
201	Campo-Formio
.....	.....

Figur 4 – Uppslagstabellernas gemensamma uppbyggnad

4 Svar till ställda frågor

4.1 Algoritmer

Vad det gäller de algoritmer som används i systemet har de valts ut på två kriterier, det första är att de uppfyller den funktion som önskas och det andra är att den är någorlunda enkel att implementera med avseende på den modell som har skapats. Detta är ganska viktigt eftersom det är svårt att ändra modellens uppbyggnad i detta skede. Det har använts både kända och mindre kända algoritmer. De algoritmer som relaterar till denna kurs och som använts är Warshall, Floyd-Warshall samt QuickSort.

4.1.1 Warshall

För att kunna besvara frågor kring om systemets samtliga stationer är kopplade till varandra behövs Warshall [4]. Algoritmen används på flera ställen i systemet för att kunna kontrollera ifall systemet är intakt.

4.1.2 Floyd-Warhall

För att kunna bestämma en resväg mellan två stationer behövs först en matris över avståndet (antal kanter) det är mellan stationerna. Detta görs genom att använda algoritmen Floyd-Warshall som kan bestämma dessa avstånd.

4.1.3 QuickSort

För att kunna presentera stationer och linjer i systemet på en värdefullt sätt har någon sorts sorteringsalgoritm behövt användas. QickSort har valts för att den både är hyfsat effektiv och samtidigt enkel att implementera. Sorteringen använts när stationer och linjer presenteras på skärm och då görs detta i bokstavsordning för att man enklare skall kunna hitta den sökta stationen eller linjen.

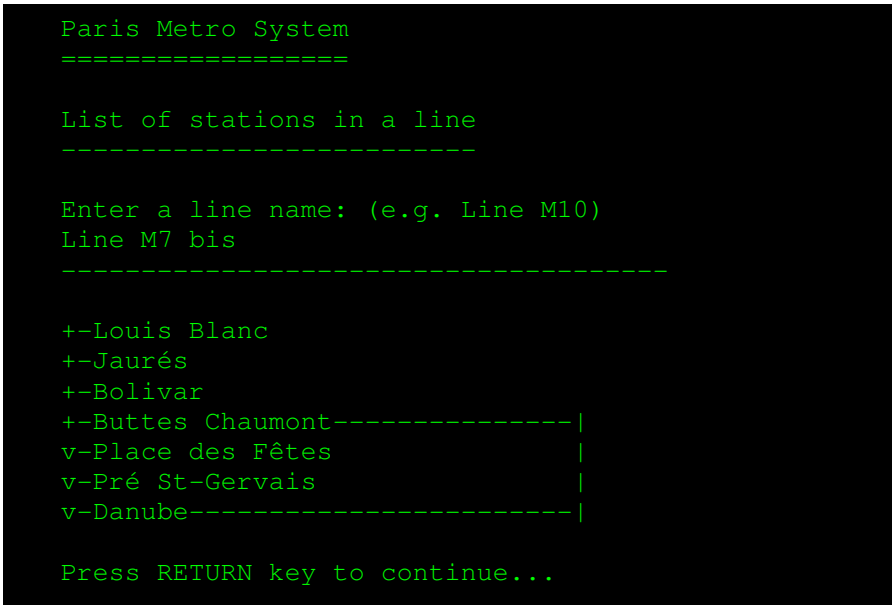
4.2 Icke besvarade frågor

4.2.1 Linjens topologi

Från början var tanken att visa information om hur en linje ser ut genom att rita den på skärmen. Trots att programmet arbetade i en terminalmiljö. Hur detta skulle gå till var löst och var visat att fungera för en linje.

Dock har modellen som skapats inte kunnat besvara frågan om hur en linjes topologi ser ut. Detta för att det saknas information i modellen om vilken linje en viss kant mellan två stationer hör till. Eftersom den informationen saknas blir det problem då en det finns flera kanter mellan stationer på linjen och dessa kanter inte tillhör linjen. Den information som finns är vilka stationer som finns på linjen vilket gör att även kanterna blir med.

Det fanns försök på att göra utskrift av topologi på linjer och dessa fungerade mycket bra så länge det inte fanns flera kanter mellan stationer där en eller flera kanter inte tillhörde linjen. Då detta uppstod i modellen fick denna ide slopas och linjens stationer skrivs i bokstavsordning. På grund att frågan är obesvarbar med nuvarande modell har problemet inte lösts ordenligt.



Figur 5 – Det tänkta upplägget för att presentera en linjes topologi (exempel)

4.2.2 Minst antal byten vid resa

När en färdväg mellan två stationer beräknats fram erhålls inte den väg som har minst antal byten utan istället visas den väg som har minst antal stationer. Under implementationen av dessa algoritmer konstaterades att det möjligtvis skulle gå fortare att byta mellan flera tåg och på så sätt resa färre stationer än att resa fler antal stationer och på så sätt riskera mindre byten. Så frågan om minst antal byten vid resa mellan två stationer besvaras inte av systemet utan istället besvaras frågan om minst antal stationer.

4.2.3 Raderandet av stationer och linjer

Möjligheten att radera stationer, linjer eller anslutningar mellan stationer i systemet. Att detta skulle gå att göra var inget som fanns i åtanke från första början. Tanken om att radera stationer ur ett tunnelbanesystem är en angelägen fråga, speciellt i dessa dagar. Som en följd av detta uppstår ett par andra frågor; innebär en borttagning av en station att stationen stänger och tåg passerar förbi, eller att stationen inklusive spår tas bort? Möjligt skulle vara att bygga om spår och tunnel.

Som sagt finns inte funktionaliteten för att radera en station ur systemet finns inte. Dels för att en radering av en station skulle vara avancerad att utföra i systemet då omslutande stationer måste kopplas om och dels för att raderandet av en station inte skulle vara möjligt i ett tunnelbanesystem enligt vår sak.

Om en station skulle exempelvis utsättas för ett attentat med sprängmedel kommer antagligen spår och elförsörjning på stationen att förstöras. Detta kommer att innebära passage förbi stationen inte är möjlig med ett tåg och om strömförsörjningen skadas kommer även flera, närliggande stationer, på den aktuella linjen att påverkas eftersom någon drivström för motorvagnarna inte kan levereras. Mer problematik om stationer som vid raderande delar upp systemet i flera mindre system finns under rubriken ”4.4.1 Artikuleringspunkter och biconnected”.

4.3 Färdväg mellan två stationer

För att kunna besvara frågan om färdvägen mellan två stationer i systemet används först Floyd-Warshall [1] för att bestämma avståndet från alla noder till alla noder. Denna algoritm har adderats med en matris för att lagra billigaste vägen mellan stationer, se Bilaga A. Med hjälp av informationen från denna matris kan man sedan använda algoritmen funnen i Bilaga B för att hitta de stationer man ska färdas genom för att finna målet. Teorin och funktionen för denna algoritm är hämtad från [2].

Genom att sedan gå genom den beräknade resvägen går det avgöra på vilka stationer som resenären behöver byta linje. Genom att kontrollera den aktuella stationen mot de den föregående kan en gemensam eller gemensamma linje bestämmas. Detta ger den kortaste resvägen, utan att ta hänsyn till antal byten.

4.4 Säkerhet i systemet

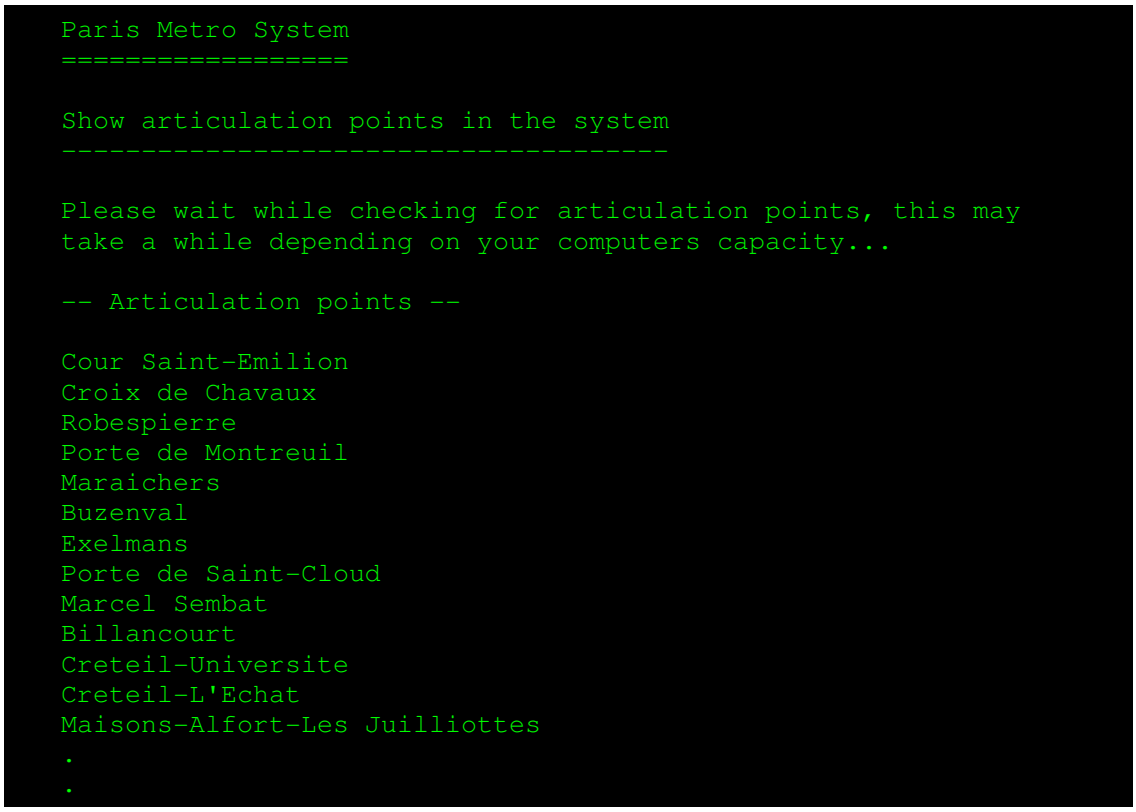
4.4.1 Artikuleringspunkter och biconnected – svaga punkter.

En artikuleringspunkt är en sådan punkt i ett system, så att om denna raderas så kommer systemet att bli till flera delar. Detta hänger även samman med huruvida systemet är biconnected. Med biconnected menas att det från början finns ett system där man måste ta bort minst två noder för att systemet skall bli i två eller fler skilda delar.

Paris tunnelbanesystem är inte biconnected och detta konstateras genom att räkna antalet artikuleringspunkter. Ifall det existerar artikuleringspunkter och därmed är systemet inte biconnected. För att erhålla systemets svaga punkter finns ett menyalternativ

Konsekvenserna av ett sådant system kan bli att ifall en station av någon anledning inte längre kan trafikeras får vi två tunnelbanesystem. Lösningen på ett sådant problem är inte alltid enkelt, mycket beroende på vilka system som tunnelbanan är byggd kring. Att vända ett tåg, där det inte är meningen att det ska vända, ger exempelvis system som ATC[1] och ATO[2] problem. (Dock är det okänt om paris tunnelbana har sådana system). Systemen kommer antagligen inte tillåta att detta sker, eftersom varje spår endast är menat till att gå åt ett visst håll.

En fullständig lista över artikuleringspunkter finns i bilaga C.



Figur 6 – Skärmen för artikuleringspunkter, fullständig lista finns i bilaga C.

4.4.2 Sammankopplat

Med hjälp av Warshall [4] bestäms om systemet är sammankopplat. Vid uppstart kontrolleras samtliga stationer i systemet, som just lästs från fil, för att kunna säkerställa att alla stationer är i kontakt med varandra. Dessutom kontrolleras det samma så fort en ny station med anslutningar till andra stationer har lagts till. Warshall används också för att kontrollera om alla stationer på en linje har kontakt med varandra, detta bland annat för att bestämma ifall linjen är intakt och i så fall kunna bestämma en begynnelsestation.

[1] Automatic Train Control – ett system för att kontrollera att fordonet följer signalering och hastighet.
[2] Automatic Train Operation – ett övergripande system som efter signaler och hastighetsinformation kör tåget.

5 Utvärdering

Efter den första prototypen visade sig modellen, som skapats, inte hålla för ytterligare ökad funktionalitet enligt specifikationerna. Modellen behövde därpå utökas med mer representationer för att över huvud taget göra modellen brukbar. Att modellen var så pass dåligt gjord berodde mycket på bristande kunskaper i modellering. I och med att projektet fortskred insågs det att mer information behövdes. Inför andra fasen uppdaterades modellen med mycket mer information och därmed gick det att besvara de olika frågeställningarna. Att kunskapen var så låg när den första modellen togs fram berodde främst på att ingen av deltagarna i projektet var speciellt vana vid att modellera system. Dessutom antogs, baserat på tidigare kunskaper, att redundans av information skulle undvikas till största möjliga mån – dessa tankar förkastades senare och modellen uppdaterades.

Efter att modellen hade uppdaterats blev systemet mycket mer lätthanterligt, även om det fanns fler delar av systemet som behövde uppdateras. Dock fanns ingen tid till detta. I det inledande skedet valdes en implementation av en lista som inte var effektiv nog. Till en början verkade inte detta ha någon inverkan på projektet men efter hand när systemet storlek ökade blev systemet långsamt. När detta var ett faktum var projektet så långt skridet att det var en omöjlighet att förändra representationen utan att äventyra att systemet blev klart.

Vad det gäller tiden som ägnats att få systemet så bra som möjligt har denna varit ganska stor trots att resultatet inte blivit det bästa. Vilken mängd tid som lagts ner exakt finns inga förteckningar över dock har mer än 100 timmar per person (två personer) lagts på detta moment. Systemet är långsamt och ineffektivt trots detta. Många av de vägval som gjorts på vägen fram till den slutgiltiga produkten har varit förödande för effektivitet hos systemet. Exempelvis behövs många upprepade beräkningar göras trots att dessa kunde ha undvikits med en annan lösning. Dock har inte den tid som varit utsatt mellan deadlines tillåtit allt för stora ändringar i systemet utan all tid har lagt på att få ett system som fungerar över huvud taget. Mycket tid, från projektdeltagarnas sida, har lagts på att försöka få systemet att presentera data på ett bra sätt. Att bara dumpa data till skärm ger varken datorn eller användaren något och att presentera data i en konsolapplikation är inte alltid en enkel uppgift. Det har under projektets gång funnits en lösning på representationen av en linje i något så där grafisk form i konsolen. Dock visade sig denna lösning inte fungera när hela systemet var implementerat, vilket var synd. Dock fanns inte tiden

att lösa problemet heller. Systemet borde egentligen inte alls köras i någon konsol utan i en grafisk miljö. Vi lever trots allt i informationsåldern och är en bit in på 2000-talet.

Det svåraste momentet under projektet var att förstå vad det hela gick ut på, att sätta sig in i systemet och förstå vad som krävs av det slutgiltiga resultatet. Till att börja med var det svårt att sätta sig in i hur man kunde modellera system då detta inte hade gjorts i någon större utsträckning tidigare. Det var viktigt att försöka komma på hur man kunde modellera en tunnelbana som den i Paris. Efter att ha kommit över den tröskeln var det mindre detaljer kvar att komma på. Det stora var att förstå varför vissa saker behövde modelleras på ett visst sätt. Tyvärr tog det tid att förstå detta och mycket dyrbar tid gick förlorad inför skapandet av prototypen. Detta resulterade i att prototypen fick en lite sämre implementation.

Lärdomen dragen av den här laborationen är förutom mer kunskap inom programspråket C++ även kunskap om hur man kan modellera olika fall av verkligheten. Att man måste göra antaganden och att allt inte är trivialt varje gång. Att ett och samma problem kan lösas på flera sätt har än en gång visat sig. Det gäller att vi varje skede göra ett så bra beslut som möjligt utifrån de förutsättningar man för tillfället har.

Referenser

[1] Floyd-Warshall algorithm: http://en.wikipedia.org/wiki/Floyd-Warshall_algorithm
[2] The Floyd-Warshall: http://www.cs.ucf.edu/~reinhard/classes/cop3503-fall03/floyd.pdf
[3] Paris tunnelbana: http://www.cs.kau.se/cs/education/courses/davb03/p1/metro.pdf
[4] Warshall algorithm: http://students.ceid.upatras.gr/~papagel/project/warshal.htm

Bilaga A – Floyd-Warshall

```
//Initiera
  for(int i = 1; i <= stations.size(); i++)
    for(int j = 1; j <= stations.size(); j++)
      pred[i][j] = 0;

  //Utför
  for(int k = 1; k <= stations.size(); k++)
    for(int i = 1; i <= stations.size(); i++)
      for(int j = 1; j <= stations.size(); j++)
        {
          if(dist[i][j] > (dist[i][k] + dist[k][j]))
            {
              dist[i][j] = dist[i][k] + dist[k][j];
              pred[i][j] = k;
            }
        }
```

Bilaga B – TravelPath

```
void travelPath(int i, int j,  map<int, map<int, int> > pred, int *tway)
{
  if(pred[i][j] == 0)
    {
      //find empty pos
      int emPos;
      bool found = false;

      for(int k = 0; k < stations.size() && !found; k++)
         if(tway[k] == 0)
           {
             emPos = k;
             found = true;
           }
      tway[emPos] = stations.getElementPointer(i)->getStationId();
    }
  else
    {
      travelPath(i, pred[i][j], pred, tway);
      travelPath(pred[i][j], j, pred, tway);
    }
}
```

Bilaga C – Artikuleringspunkter

Cour Saint-Emilion
Croix de Chavaux
Robespierre
Porte de Montreuil
Maraichers
Buzenval
Exelmans
Porte de Saint-Cloud
Marcel Sembat
Billancourt
Creteil-Universite
Creteil-L'Echat
Maisons-Alfort-Les Juilliottes
Maisons-Alfort-Stade
Ecole Veterinaire de Maisons-Alfort
Charenton-Ecoles
Liberte
Porte de Charenton
Porte Doree
Michel Bizot
Commerce
Felix Faure
Boucicaut
Lourmel
Pierre Curie
Porte d'Ivry
Porte de Choisy
Porte d'Italie
Villejuif - Paul Vaillant Couturier
Villejuif - Leo Lagrange
Le kremlin-Bicetre
Maison Blanche
Tolbiac
Riquet
Crimee
Corentin Cariou
Porte de la Villette
Aubervilliers-Pantin Quatre Chemins
Fort d'Aubervilliers
Danube
Pre-Saint-Gervais
Place des Fetes
Botzaris
Daumesnil
Bercy
Place d'Italie
Laumiere
Ourcq
Porte de Pantin
Hoche
Eglise de Pantin
Bobigny-Pantin - Raymond Queneau
Alesia
Mouton-Duvernet
Denfert-Rochereau
Simplon
Porte de Bagnolet
Gambetta
Malesherbes
Wagram
Pereire
Porte de Champerret

Louise Michel
Anatole France
Berault
Saint-Mande-Tourelle
Porte de Vincennes
Argentine
Porte Maillot
Les Sablons
Pont de Neuilly
Esplanade de La Defense
Corentin Celton
Porte de Versailles
Convention
Vaugirard
Volontaires
Pasteur
Marcadet-Poissonniers
Marx Dormoy
Nation
Jaures
Stalingrad
Villiers
Charles de Gaulle-Etoile
Victor Hugo
La Motte-Picquet-Grenelle

Porte d'Auteuil
Michel-Ange-Auteuil
Javel-Andre Citroen
Mirabeau
Chardon-Lagache
Michel-Ange-Molitor
Boulogne-Jean Jaures
Malakoff-Rue Etienne Dolet
Malakoff-Plateau de Vanves
Porte de Vanves
Plaisance
Pernety
Gaite
Montparnasse-Bienvenue
Place de Clichy
Brochant
Porte de Clichy
La Fourche
Guy Mouet
Porte de Saint-Ouen
Garibaldi
Mairie de Saint-Ouen
Carrefour Pleyel
Saint-Denis-Porte de Paris
Basilique de Saint-Denis

Antal artikuleringspunkter: 113