Projekt - Järnvägsnät DV1490

Av: Erik Thomasson

Inledning

Jag fick välja att lösa ett problem utav tre stycken möjliga. Jag valde att implementera lösningen till det första problemet vilket var att skapa ett program som hittar billigaste vägen från en bestämd tågstation till en annan bestämd tågstation. Andra problemet var att man skulle implementera en algoritm som lägger den billigaste rälsen mellan alla stationer. Alla stationerna skall även vara sammanlänkade. Den tredje implementationen var att hitta en rutt för turister att åka på alla spår en och endast en gång.

Beskrivning av problemet

Om man kollar översiktligt på problemet jag valde att implementera så skall programmet ta emot en textfil och en textsträng som innehåller start station och slut station. När programmet är klart skall det finnas en ny textfil som innehåller informationen om vägen från start stationen till slutstation. Alltså vilka stationer som turisten åker förbi och i vilken ordning och information om kostnaden från start stationen till slut station skall också finnas i den nya textfilen.

Först måste man hantera informationen man får och lagra det i lämpliga variabler. För att lösa det så måste man öppna och läsa filen som är angiven till programmet. Man kan även lämpligtvis spara start stationen och slut stationen i två text strängar.
Efter man har hanterat informationen man får från användaren skall man använda den informationen för att skapa stationer och förhållandet mellan stationerna för att kunna hitta närmaste vägen från start station till slut station. Detta kan man lösa på olika sätt men den generella iden är att börja från start stationen och sedan ta sig utåt för att hitta den närmsta vägen från start stationen till alla andra stationer. När man väl har kartlagt närmsta vägen från start stationen till alla andra stationer. Så kan man enkelt se kostnaden från start stationen till slutstationen om man har en variabel i varje station som anger kostnaden från första stationen till sista stationen. Utöver kostnaden från start stationen till slutstationen skall även vägen från start stationen till slut stationen anges i svars filen. Detta har jag löst genom att varje station vet vilken station tåget kom ifrån. Man kan då börja från slutstationen och börja med att ta sig till stationen innan och därefter spara stationens namn. Gör man detta sedan rekursivt för varje station tills man kommer till start stationen så får man vägen som tåget tog sig från start stationen till slut stationen.
När man väl har denna information så avslutas programmet med att skriva den informationen till en ny textfil.

Beskrivning och motivering av tillvägagångssätt

Datastrukturer

Datastrukturerna jag använde var Prioritetskö och en egen implementerad datastruktur för en tågstation.
En Prioritetskö är en datastruktur som är precis som en kö men med prioritet alltså när man lägger in ett nytt element i kön så placeras det i förhållande till de andra elementen alltså blir kön sorterad. Prioritetskön som jag använde prioriterade minsta värdet till högsta värdet alltså att minsta värdet är först i kön och högsta värdet är sist i kön. Förutom att lägga in ett element i kön på rätt plats så att det är sorterad kan man även ta bort elementet först i kön, kolla på det första värdet i kön och kolla om kön är tom. En Prioritetskö använde jag för att kunna hitta den närmsta vägen från start stationen till varje annan station.

Den andra datastrukturen implementerade jag specifikt till mitt problem. Datastrukturen namngav jag till Vertex vilket skall representera vad varje tågstation innehåller. En Vertex innehåller variablerna “distance” som håller värdet för kostnaden till start stationen, “path” som är en tågstations pekare vilket innehåller tågstationen som tåget kom ifrån, “name” som innehåller namnet på tågstationen och “adj” som innehåller alla tågstationer som är anhörig till tågstationen och kostnaden till dem. Jag har även överlagrat mindre än(<) operatorn för att kunna interagera med Prioritetskön. Jag skapade Vertex för att kunna spara information om varje station på ett enkelt sätt.

Algoritmer

Jag har använt mig av de egen implementerade algoritmerna readFile, write2File, printPath och dijkstra. Funktionen readFile läser innehållet i filen som skrivs in vid användning av programmet. Funktionen tar in en sträng som argument vilket är tänkt att vara vägen till informations filen och sedan returnerar den en dictionary med nyckel typen sträng och värdet Vertex(tågstation).Funktionen börjar med att öppna filen om den finns och sedan läser av första raden i textdokumentet och kollar om grafen är riktad eller inte. sedan läser jag tills en rad är tom och tar de värdena och skapar nya Vertex för varje värde, jag sätter även värdet och den nya Vertexen in i den returnerade dictionarien. Efter den tomma kommer informationen om vilka stationer som angränsar till varandra och kostnaden mellan dem. Om filen inte är riktad så lägger jag till att både angränsar till varandra och med likadan kostnad.
Efter det lagrar jag all information från filerna i Vertex och dictionarien med namnet på tågstationen och informationen den håller. Därefter tar jag den informationen från dictionarien och en sträng som representerar start stationen som argument i funktionen dijkstra.

Dijkstra funktionen börja med att sätta distance i start stationen till 0 funktionen sätter sedan in alla stationer i en prioritetskö. Det skapas sedan en pekare som pekar på stationen som har lägst distans och tar sedan bort den från prioritetskön. Sedan kollar funktionen igenom

alla angränsande stationer och kollar om den närmsta vägen från start stationen och de angränsande noderna är den närmsta vägen och ändrar isåfall distance i den noden och sätter även in den noden i prioritets kön. Detta görs tills prioritets kön är tom på det viset kollar funktionen om det finns en kortare väg till varje nod på alla möjliga sätt.

När dijkstra funktionen är klar så vet programmet närmsta vägen från start stationen och slutstationen. Programmet har även informationen om vilken väg som tåget har gått men det behövs en textsträng med den informationen för att kunna skriva in det i svarsfilen.
Då används funktionen printPath som är en rekursiv funktion som tar in en Vertex och en referens till en sträng. Funktionen är menat att ta in slutstationen först. Funktionen kollar om det finns en föregående station till stationen och isåfall anropar funktionen på den föregående stationen. funktionen lägger sedan in namnet på stationen. När funktionen sedan är klar så har textsträngen som sattes in, informationen om vilken väg tåget tog från start stationen till slut stationen.
Programmet avslutar med att använda funktionen write2File. Funktionen tar en int och en sträng som argument och har inget returvärde. Inten används för att skriva distansen från start stationen till slut stationen. Strängen beskriver vägen tåget tog från start stationen till slut stationen. Funktionen öppnar en ny textfil och skriver in den informationen som är givet i argument listan därefter är programmet klart.

Analys av implementationen

Jag använde mig utav min egna tillämpade prioritetskö vilket använder en länkad lista detta gör den både långsammare i aspekten att den använder sig utav en länkad list och inte en heap. men också eftersom den inte är optimerad fullständigt vilket förmodligen standard bibliotekets prioritetskö är. Den andra data strukturen jag använde var Vertex vilket även den skulle man kunna optimera inte lika mycket som prioritetskön men lite iallafall. tex skulle man kunna använda sig utav en osorterad dictionary och några variabler i datastrukturen är säkert överflödiga.

Jag anser att jag inte gör något onödigt invecklat i mina algoritmer men jag har inte heller haft fokus på optimering eftersom det inte var ett krav i uppgiften. visa delar av programmet är ett måste vilket gör att man har vissa begränsningar t.ex. måste man öppna informations filen, hitta närmaste vägen från start till slut och sedan skriva information till svars filen.

• Reflektioner över arbetet
– Hur mycket tid har du lagt ner på projektet och hur är tiden fördelad?

Reflektioner över arbetet

Jag har lagt ner mer tid än jag trodde eftersom varje rad slutade med “\r\n” i textfilerna vilket skapade problem hur jag läste in noderna och alla tillhörande stationer. Jag utgick först med att alla rader slutade med “\n” men efter jag upptäckte att de slutade med “\r\n” fick jag tänka om hur jag skulle använda mina std::getlines. Jag trodde det skulle ta 7 timmar att göra klart programmerings delen men det tog närmare 11 timmar. rapporten trodde jag skulle ta 3-4 timmar vilket det också tog.

Jag gjorde inga speciella förberedelser förutom att jag förde anteckningar på föreläsningen om dijkstra funktionen och hur man kunde hitta den närmsta vägen från en nod till en annan i en graf.

Jag använde mig av min egen implementerade prioritetskö vilket använder sig utav en länkad lista. Det gör så att programmet blir långsammare och detta märks mer om man har fler stationer. Hade jag använt priority queue från standard biblioteket hade programmet blivit snabbare men det var inte ett krav i uppgiften.