Grafer

viktor björkén

October 2022

1 Introduktion

I denna uppgift ska vi med olika algoritmer hitta olika bästa vägen med hjälp av Järnvägnsnätverket.

Vi kommmer använda oss utav olika sökmetoder samt andra förendringar för att se vad som fungerar bäst för våran ändamål.

2 Tåg från en punkt

I denna uppgift ska vi med hjälp av en CVS fil ta alla värden för att visualisera detta med en graf. Vi kommer göra detta för att hitta den snabbaste vägen. Vi kommer vi att använda det svenska järnvägsnätverket för att ha något verkligt att utgå ifrån när vi jobbar med denna uppgift. Vi kommer ha 52 städer med 75 olika förbindelser som alla kommer kunna gå åt båda hållen.

2.1 Grafen

Vi börjar med att läsa av filen med alla förbindelser samt städer för att sedan addera så att alla förbindelser är dubbelriktade. Som vi sedan visar med en graf.

Vi kan göra detta genom att representera våra städer som strukturer där vi har namnet på staden samt en array för att represenetera dess grannar med den första staden samt avståndet i minuter.

Vi börjar med skapa en klass för städerna som också har en meotd för att lägga till nya förbindelser. Sedan skapar vi också en egen klass för förbindelserna.

Därefter implementerar vi våran karta som vi kommer att anävda för att kunna använda en loopup metod för att addera flera städer till kartan. Denna metod kommer också att användas när vi ska ”korsar” kartan för att hitta 2 olika noder. När vi gjort detta har vi allt vi behöver från objektet. Vi börjar med att konstruerar våran lookup metod och återkommer till kartan i ett senare styckte.

```
Public City lookup(String name){
```

```
    Integer index = hash(name);

while (true){
    if (cities[index] == null) {
        break;
    }
    if (cities[index].name.equals(name)){
        return cities[index];
    }

    }

}
```

2.2 Hash

I denna uppgift kommer vi också att använda oss utav hashtabeller om vi också gjorde i den föregående uppgiften.

Det vi gör är att vi först skapar oss en hash funktion som tar en en String för att skapa unika hash värden detta gör att vi får en relativ bra fungerade hashfunktion. Det vi också måste tänka på när vi konstuerar denna är vårat mod värde. Det får inte vara för stort för att inte slösa minne men heller inte för litet för att få med alla städer. Ett bra värde för detta är primtalet 541 då det passar dessa kriterier bra.

2.3 kartan

Vi böjrar med att bygga upp våran karta som läser av CVS filen och stoppar in. Vi kommer att först och främst se till och kolla i filen rad för rad efter staden vi åker från, staden vi åker till, därefter lägga till våran förbindelse.

Vi lägger till den kod som krävs med hjälp av den bas som vi får i uppgiften vilket gör att vi får en kod som ser ut på föjande sätt.

```
public class Map {

    private City[] cities;
    private final int mod = 541;

    public Map(String file) {

        cities = new City[mod];

        try (BufferedReader br = new BufferedReader(new FileReader(file))) {
            String line;
            while ((line = br.readLine()) != null) {
```

```
            String[] row = line.split(",");
            City one = lookup(row[0]);
            City two = lookup(row[1]);
            Integer dist = Integer.valueOf(row[2]);
            //System.out.println("adding " + row[0] + " to " + row[1]);
            one.connect(two, dist);
            two.connect(one,dist);
            }
        } catch (Exception e) {
            System.out.println(" file " + file + " not found or corrupt");
        }
    }
    //hash();
    //lookup();
}
```

3 Kortaste vägen

Det vi ska göra nu är att försöka hitta den kortaste vägen mellan 2 olika städer, vilket är det huvudsakliga syften med denna uppgift. Våran första implementation kommer att vara till den enklare graden där vi endast visar hur lång tid våran tur kommer att ta och tar inte hänsyn till vägen vi åker.

3.1 Djupet först söking

Vi kommer börja med att använda oss utav en djupet först sökning för att hitta den kortaste vägen.

När vi använder denna metod kommer det att uppstå vissa problem därför behhöver vi också ha ett max värde. Detta värde kommer vara ett värde på max hur lång tid en resa får ta. Vi använder oss av detta värde för att vi inte ska hamna i en oändlig loop.

Som tidigare använder vi kodsnippar vi har fått i uppgiften och bygger vidare på dessa.

```
Integer dist = shortest(conn.city, to, max - conn.dist);

if((dist != null) && ((shrt == null) || (shrt > dist + conn.dist))){
    shrt = dist + conn.dist;
    }

if((shrt != null) && (max > shrt)) {
    max = shrt

    }
```

```
    }
}
return shrt;
}
```

4 Benchmark

Det vi vill göra nu är att kolla och jämföra olika värden på de kortaste vägarna.

Det vi måste tänka på är att skriva ett tillräckligt högt max värde så att vi inte fastnar i en loop.

Avgång	Destination	resetid (minuter)	exikveringstid (us)
Malmö	Göteborg	153	1045
Göteborg	Stockholm	211	954
Malmö	Stockholm	273	476
Stockholm	Sundsvall	327	1364
Sundsvall	Umeå	190	1523
Umeå	Göteborg	705	1894

De resor som jag inte har valt är resor som ja valde att avbryta de tog för lång tid.

4.1 Vägar

För att undvika flera loppar i programmet kan vi skriva om det. Vi kommer att skriva ett nytt program som kommer hantera looparn på ett annat sätt. Vi gör en kopia på vårat föregånede program men nu skapar vi en array som är tillräckligt stor som antalet städer. När vi nu söker så ser vi till att staden vi befinner oss i inte finns i våran path array.

Det som gör denna version bättre är att vi nu inte behöver skriva in vårat max värde längre.

En annan sak att förbättra är hur vi kan sätta restriktioner i framtiden efter man hittat en väg. Med detta menas att om A är ansluten till B,C,D och när vi ska hitta en väg till B, hittar man en väg från B till O som är 460 minuter. Om avståndet mellan A och B är 60 samt att alla andra väger från A till O är kortare än 520 minuter. Om vi nu uppdaterar detta för vårat max värde när vi testar olika förbindlerser behövs inget intitialt värde.

Om vi nu gör om några av våra benchmarks kan vi jämföra och se vad som skiljer.

Avgång	Destination	resetid (minuter)	exikveringstid (us)
Malmö	Umeå	790	110211
Malmö	Kiruna	1162	93483
Kiruna	Malmö	1162	1403
Kiruna	Jönköping	1068	1023
Kiruna	Halmstad	1145	1432

5 Saker att tänka på

Den lösning vi tidigare har konstuerar fungerar bra för att göra jobbet men vi kommer stäta på problem när vi ska jobba med större kartor. Vi kan göra en tabell med större avstånd och se hur lång tid det tar till olika städer med längre och löngre avstånd från malmö. Vi kan ta hjälp av google maps för att hitta den bästa tågresan.

Ett annat problem som vi måste hanskas med är att våran algoritm inte kommer iåhg någonting sen innan. Ett exempel på varför detta kan vara problematiskt är om vi till exempel om vi söker den kortaste resan mellan Malmö och Kiruna kommer vårtat program tillslut nångång gå förbi Stockholm, Därefter kommer den att söka efter kortaste vägen mellan Stockholm och kiruna. Programmet kan då använda denna data för att beräkna snabbaste resan från södertälje till kiruna. Detta gör det viktigt att komma ihåg bästa vägen från Stockholm till Kiruna.