INDU: Tåg

Kristoffer Sahlin

Innehåll

Introduktion

I det här projektet ska du skriva ett Python-program för att simulera tåg som kör fram och tillbaka på olika linjer (tänk tunnelbanan). Vi kommer titta på en enkel diskret ("turn-based") modell där angränsande stationer befinner sig en tidsenhet ifrån varandra. I varje "steg" (tidsenhet) i modellen befinner sig ett tåg vid en station. Från ett steg till ett annat kan två utfall ske för ett tåg: antingen har tåget nått nästa station, eller så har det blivit stående på samma station (försening). Kartan över tågstationer kan visualiseras som en graf där varje nod är en station och varje kant beskriver att två stationer är anknutna med tågräls (se exempel i fig. 1).

Det här projektet har två uppgifter vilka bygger på varandra. För att få betyg E-C räcker det att göra uppgift 1. För högre betyg behöver man även göra uppgift 2. Se de generella instruktionerna i "INDU-Individuell Uppgift i Programmering" för detaljer kring betygssättningen.

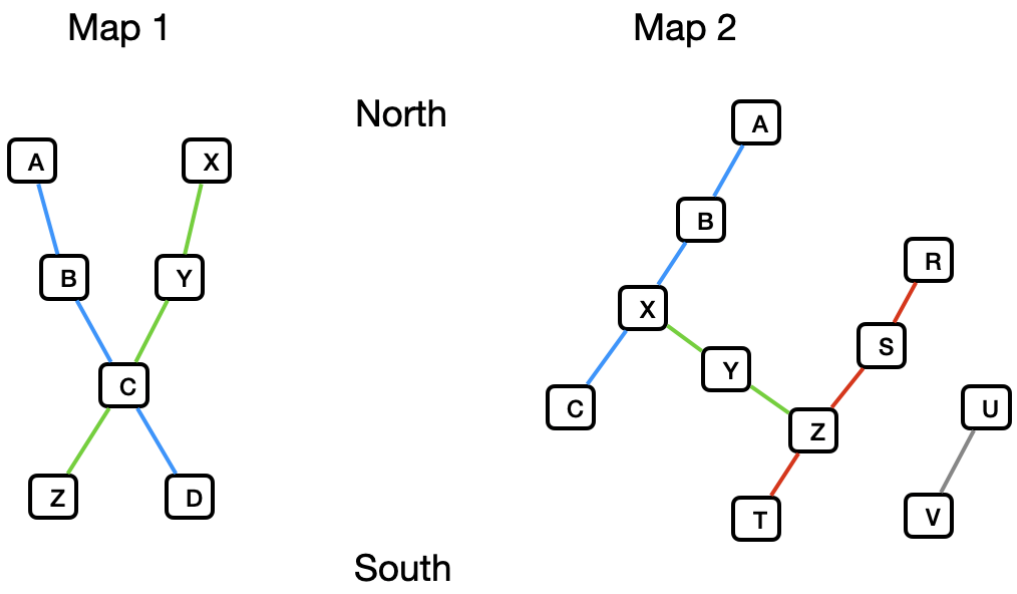
Modellen

Vi beskriver nu detaljerna av modellen. Varje tåg är bunden till en linje. Det finns endast räls mellan stationer på samma linje. När ett tåg når antingen den nordliga eller sydliga slutstation på linjen vänder de och åker tillbaka.

Vid start av programmet ska vi slumpa fram n tåg, där varje tåg slumpas till en station samt med en riktning (nordlig eller sydlig riktning). Om ett tåg slumpas till en slutstation, t.ex. den nordliga slutstationen, kan de bara få en sydlig riktning. I denna modell tillåter vi att det kan stå flera tåg på varje station (tänk oändligt antal spår). Storleken på n är därför oberoende av antal stationer, och alla tåg kör oberoende av varandra.

Antag att tåget Bobby befinner sig på gröna linjen vid station X i Map 1, fig. 1 åkandes i sydlig riktning i tidsenhet t. Det finns då två möjliga utfall för Bobby i tidsenhet $t + 1$: Antingen kommer Bobby fram till station Y, eller så blir Bobby stående på station X (försening). Eftersom station X har en sydlig granne (Y) som inte är en slutstation behåller Bobby *sydlig riktning* i båda fallen.

Antag istället att Bobby befinner sig på gröna linjen vid station C i Map 1, fig. 1 åkande i sydlig riktning i tidsenhet t. Station C har en sydlig granne som är en slutstation. I det fallet skiljer sig de två möjliga utfallen för



Figur 1: Två kartor på möjliga tåglinjer med olika färger. Varje tåg kör fram och tillbaka på endast en linje. En station kan knyta samman flera linjer (t.ex. station C i Map 1 eller station X i Map 2).

Bobby något från tidigare scenario i tidsenhet $t + 1$: antingen kommer Bobby fram till slutstationen Z och byter då riktning från sydlig till *nordlig* riktning, eller så blir Bobby ståendets på station C (försening) och behåller sydlig riktning.

Varje tågstation har en specifik sannolikhet $p \in [0, 1]$ att det blir försening. Simuleringen stegar fram en tidsenhet i taget. Vid varje omgång ska användaren kunna fråga var tåg $k \in [1, n]$ befinner sig och vilken riktning det har.

Format på indata

Indata består av två filer. En kommaseparerad fil innehållande tågstationer (kolumn 1) och sannolikhet för försening (kolumn 2). Exempel för Map 1 i fig. 1 visas nedan (ni bör kopiera över detta till en fil som heter `stations.txt`). Vi kan tolka från denna fil att gröna linjen har större sannolikhet för förseningar (t.ex. signalfel) och att station C är värst drabbad.

```
A,0.001
B,0.001
C,0.2
D,0.001
X,0.1
Y,0.1
Z,0.1
```

Den andra filen är också en kommaseparerad fil innehållande tågstation 1 (kolumn 1), tågstation 2 (kolumn 2), linje (kolumn 3), och riktning (kolumn 4). S står för ”Syd” och N för ”Norr”. Denna fil beskriver vilka kanter som finns i grafen och deras linje (ni bör kopiera över detta till en fil som heter `connections.txt`).

```
A,B,blue,S
B,C,blue,S
C,D,blue,S
X,Y,green,S
Y,C,green,S
C,Z,green,S
```

Första raden i filen beskriver att:

i. Det finns tågräls mellan *A* och *B*,

ii. de båda stationerna är angränsande stationer på blå linjen, och

iii. att åka från *A* till *B* betyder att vi åker söderut.

Obs: programmet ska fungera för andra indata än de ni har fått här. Hitta gärna på era egna tågnät och inkludera i inlämningen.

Uppgift 1 (E-C)

Vid varje steg i modellen ska programmet fråga användaren om ett val, där valen är att:

1 Simulera en tidsenhet framåt i modellen utan att visa tåginfo.

2 Visa nuvarande plats på ett tåg.

q Avsluta.

Exempelkörning

Vi visar här ett exempel med både försening och byte av färdriktning. Programmet förväntas ha följande ungefärliga interaktion:

```
Enter name of stations file: stations.txt
Enter name of connections file: connections.txt
Enter how many trains to simulate: 3

Continue simulation [1], train info [2], exit [q].
Select an option: 2
Which train [1 - 3]: 1

Train 1 on GREEN line is at station C heading in North direction

Continue simulation [1], train info [2], exit [q].
Select an option: 1

Continue simulation [1], train info [2], exit [q].
Select an option: 2
Which train [1 - 3]: 1

Train 1 on GREEN line is at station C heading in North direction (DELAY)

Continue simulation [1], train info [2], exit [q].
Select an option: 1

Continue simulation [1],train info [2], exit [q].
Select an option: 2
Which train [1 - 3]: 1

Train 1 on GREEN line is at station Y heading in North direction.

Continue simulation [1], train info [2], exit [q].
Select an option: 1

Continue simulation [1], train info [2], exit [q].
```

```
Select an option: 2
Which train [1 - 3]: 1

Train 1 on GREEN line is at station X heading in South direction.

Continue simulation [1], train info [2], exit [q].
Select an option: q
Thank you and goodbye!
```

Krav uppgift 1

- Korrekt implementation av modellen. För högre betyg måste man även uppfylla de betygskriterier som beskrivs i de generella instruktionerna i *INDU—Individuell Uppgift i Programmering*.
- **Obs:** lösningen måste fungera för andra kartor och parametrar än de ni fått här.
- Uppgifterna ska lösas utan att importera icke-standard bibliotek (exempel på icke-standard bibliotek är såna som måste installeras). Ni får använda vissa standardbibliotek, t.ex. `sys` och `random`. Fråga kursledaren om ni är osäkra om ni får använda ett specifikt bibliotek. Redogör för alla använda bibliotek i projektrapporten.
- Ni behöver inte förutse alla möjliga fall som kan ske i t.ex. indatafiler (det är många) men ni ska visa att ni tänkt på *några* för att få poäng på god felhantering. Sedan finns det många hörnfall på möjliga kartor och parametrar; visa att ni tänkt på några för att få poäng på felhantering och testning.

Tips

1. Använd standardbiblioteket `random` för att slumpa, både förseningar och tåg.
2. Tänk på designen innan ni börjar programmera. Bör ni ha några specifika klasser? Vilka funktioner bör ni ha? Kom ihåg att separarera interaktion från beräkning då det verkligen underlättar debugging och leder till bättre kod vilken är lättare att testa.

Uppgift 2 (E-A)

Vi ska nu utöka programmet i uppgift 1. Programmet ska kunna svara på om det teoretiskt går att ta sig från station p till station q inom T tidsenheter. Valen är:

1 Simulera en tidsenhet framåt i modellen utan att visa tåginfo.

2 Visa nuvarande plats på ett tåg.

3 Svara på om vi kan nå station q från station p inom T tidsenheter (givet vår karta).

q Avsluta.

Alternativ 1 och 2 är från uppgift 1 och alternativ 3 är uppgift 2. Du ska inte anta förseningar eller inkludera tågens nuvarande platser i beräkningen för alternativ 3. Du ska alltså bara undersöka om station q ligger högst T stationer bort i kartan från station p. Detta ska kunna göras genom att byta tåg, så t.ex. station D är 3 steg från station X på Map 1 i fig. 1. Om det inte finns några kanter mellan två stationer går det inte att åka mellan dem, oavsett T (t.ex. stationerna U och V kan inte nå några andra stationer i Map 2 i fig. 1).

Tips: implementera antingen en form av djupet först eller bredden först sökning.[1]. Dessa är enklast att implementera med hjälp av rekursion och det kan underlätta att skriva en funktion/metod som returnerar alla grannar till en station i stil med `neighbors` på labb 6.

[1] Se https://en.wikipedia.org/wiki/Depth-first_search och https://en.wikipedia.org/wiki/Breadth-first_search Vilken som är lättast att implementera beror lite på hur ni strukturerat upp resten av programmet.

Exempelkörning

Vi visar här ett exempel på användning av alternativ 3.

```
Enter name of stations file: stations.txt
Enter name of connections file: connections.txt
Enter how many trains to simulate: 5

Continue simulation [1], train info [2], route info [3], exit [q].
Select an option: 3
Select a start station: A
Select an end station: X
Select timesteps: 3

Station X is not reachable from station A within 3 timesteps.

Continue simulation [1], train info [2], route info [3], exit [q].
Select an option: 3
Select a start station: A
Select an end station: Z
Select timesteps: 3

Station Z is reachable from station A within 3 timesteps.

Continue simulation [1], train info [2], route info [3], exit [q].
Select an option: q
Thank you and goodbye!
```

Krav uppgift 2

- Man måste ha löst uppgift 1 och alla krav som gäller för uppgift 1 gäller även uppgift 2.
- I övrigt gäller även att om du gör båda uppgifterna ska du lämna in endast ett program. Dvs du ska utöka funktionaliteten i ditt program från uppgift 1 med uppgift 2. Dock är det bra att spara en kopia av uppgift 1 när du är klar med den innan du försöker implementera uppgift 2.

Betygssättning

Detaljer finns i de generella instruktionerna i *INDU—Individuell Uppgift i Programmering*.