## Noen turer, øving 9

Programmet skal løse korteste-vei problemet ved hjelp av utlagte kartdata. «kjøretid» skal brukes som vekting i grafen.

Programmet må både kunne gjøre ALT-søk og rent Dijkstra-søk, altså med estimat=0. Disse to søkene kan prosessere ulikt antall noder og ta ulik tid, men de skal resultere i nøyaktig samme kjøretid for bilen[1]. Det er bare en *korteste kjøretid.*

I tillegg til reiserute (liste over noder) og reisetid, skal programmet også:

- Telle opp prosesserte noder (de som ble tatt ut av prioritetskøen)
- Kjøretid for søkene. Lesing fra fil skal ikke være med i tidtakingen. Programmet må klare å finne en av rutene som er lenger enn en time, på under 10s (uten filoperasjoner) (Tidskravet gjelder bare kompilerte språk.)

For noen veier tar ALT kortere tid, fordi ALT er flinkere til å lete i «riktig retning». Antall noder blir (mye) lavere, pga. interessepunktene.

Hvis kjøretid stemmer «nesten, men ikke helt»: Sjekk om turen går baklengs. Rundkjøringer, og enveiskjørte gater gir da små forskjeller. Rot med retningen er ikke så farlig.

Gruppa må kunne visualisere reiseruter, men det kan gjøres så enkelt som å lime inn koordinater på nettsiden jeg beskriver i øvingen.

### Noen turer og returer

| *Tur* | *Fra node* | *Til node* | *Noder i veien* | *Tid (bil) tt:mm:ss* |
|---|---|---|---|---|
| Meråker–Stjørdal | 6579983 | 1693246 | 228 | 39:00 |
| Stjørdal–Meråker | 1693246 | 6579983 | 233 | 39:06 |
| Kårvåg–Gjemnes (mye kurver) | 6368906 | 6789998 | 316 | 40:36 |
| Gjemnes–Kårvåg | 6789998 | 6368906 | 316 | 40:36 |
| Malmö–Roskilde | 1048698 | 3287445 | 320 | 47:21 |
| Roskilde–Malmö | 3287445 | 1048698 | 307 | 48:27 |
| Trondheim–Oslo | 6861306 | 2518118 | 1781 | 5:55:33 |
| Oslo–Trondheim | 2518118 | 6861306 | 1788 | 5:55:35 |
| Snåsa–Mehamn | 5379848 | 2951840 | 4270 | 17:39:43 |
| Mehamn–Snåsa | 2951840 | 5379848 | 4272 | 17:39:58 |
| Stavanger–Tampere | 3447384 | 136963 | 5072 | 19:49:58 |
| Tampere–Stavanger | 136963 | 3447384 | 5013 | 19:49:11 |

[1] Om kjøretiden virker for kort, er det fordi beregningsgrunnlaget forutsetter at man ligger eksakt på fartsgrensen hele veien. Det er ikke tatt hensyn til skarpe svinger og kryss.

## Noen vanlige problemer

- Dijkstra rett, men A* har feil rute
  - Glemt å regne om til radianer. Crazy ruting!
  - Bytter om lengde- og breddegrader.
  - Bruker float, trenger double
- Programmet tar lang tid, flere minutter på søket
  - Programmet stopper ikke når det finner målnoden, og bruker tiden på å finne veiene til *alle* steder på kartet.
  - Dårlig prioritetskø. Denne anvendelsen *trenger* heap eller PriorityQueue.
  - God prioritetskø, men programmet bruker masse tid på å søke etter noder i heap (ofte for finne dem for å endre prioriteten.) Ikke gjør dette med lineært søk! Ha heller en referanse fra kartnoden i nodetabellen til prioritetskøen, og en referanse fra noden prioritetskøen tilbake til kartnoden. Referansen til fra kartnoden prioritetskøen må da oppdateres hver gang noden flyttes i heapen, men det er verdt det: søkene blir $O(1)$ i stedet for $O(n)$.
- Feil ruter, litt for lange

  Når en node får endret sitt estimat til lavere verdi, må den ofte flyttes litt oppover i heapen. Hvis man har glemt dette, blir det mye småfeil. Java sin PriorityQueue har ingen metode for å endrer prioritet, men man kan ta noden ut av køen og dytte den inn igjen etter endringen.