Øving 7, algoritmer og datastrukturer

Vektede grafer

Implementer enten Dijkstras eller Prims algoritme.

Jeg anbefaler å bruke en heap-basert prioritetskø. Ikke bare er det raskere, men det kan bli nyttig på en senere øving også.

For en enklere øving, går det an å bruke en usortert tabell som prioritetskø. Kjøretiden blir i utgangspunktet $O\left(N^2\right)$ på dette viset. Men hvis man ikke legger noder inn i prioritetskøen før de blir funnet første gang, vil kjøretiden på et typisk veikart holde seg på $O\left(N\sqrt{N}\right)$.

Format for graf-filer

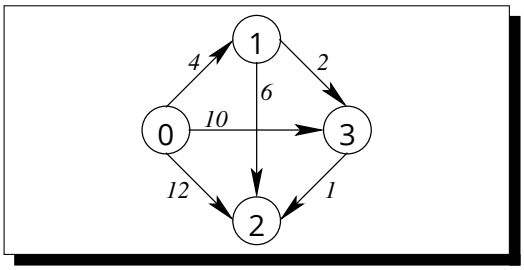
Filformat

```
Nodeantall Kantantall
franode tilnode vekt
franode tilnode vekt
```

Eksempelfil (vg1)

```
4 6
0 2 12
0 1 4
0 3 10
1 2 6
1 3 2
3 2 1
```

Tegning (vg1)



Resultater

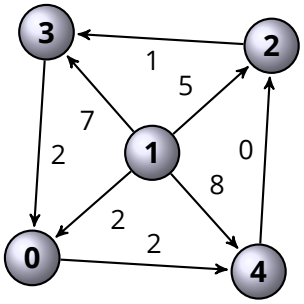
Programmet må legge frem resultatet av algoritmen, f.eks. ved å skrive det ut.

Dijkstras algoritme

Eksempel hvis Dijkstras algoritme kjøres på eksempelfiguren med start i node 1:

```
Node forgjenger  distanse
   0            nåes ikke
   1       start         0
   2           3         3
   3           1         2
```

Grafen vg5




```
Dijkstras algoritme på vg5
Node forgjenger  distanse
   0          1         2
   1      start         0
   2          4         4
   3          2         5
   4          0         4
```

Prims algoritme

Prims algoritme opererer på urettede grafer, men datastrukturen vi bruker gir oss en rettet graf. Den kan gjøres urettet ved å legge inn hver kant i begge retninger. Hvis f.eks. graf-fila har en kant fra node 1 til node 7, legger du altså inn en kant fra 7 til 1 med samme vekt. (Gjøres i innlesingsprogrammet, ikke i fila!) På det viset får vi en «urettet» graf, fordi kantene virker begge veier.
Vis hvilke kanter som blir med i spenntreet, og den samlede vekten. Resultat for eksempelfiguren (vg1):

```
Kanter:
Fra - Til  vekt
  0 - 1       4
  1 - 3       2
  3 - 2       1
Sum vekter:   7
```

Godkjenningskrav

- Et program som implementerer enten Dijkstras eller Prims algoritme. Send med utskrift for kjøring på grafene vg1 og vg5, med node 1 som startnode.
- Programmet kan behandle grafene vg1, vg2, vg3, vg4 og vg5. Det skal kunne kjøres med hvilken som helst node som startnode.

Vektede grafer

Vektet, 4 noder	`https://www.idi.ntnu.no/emner/idatt2101/v-graf/vg1`
Vektet, 5 noder	`https://www.idi.ntnu.no/emner/idatt2101/v-graf/vg5`
Vektet, 50_noder	`https://www.idi.ntnu.no/emner/idatt2101/v-graf/vg2`
Vektet, 4k_noder	`https://www.idi.ntnu.no/emner/idatt2101/v-graf/vg3`
Vektet, 50k_noder	`https://www.idi.ntnu.no/emner/idatt2101/v-graf/vg4`
Skandinavia	`https://www.idi.ntnu.no/emner/idatt2101/v-graf/vgSkandinavia`

Fil	str	noder	kanter	
vg1		4	6	Eksempelgraf vg1
vg5		5	8	Eksempelgraf vg5
vg2	1,5 kB	50	100	Tilfeldige koblinger
vg3	79 kB	4 000	5 000	Ring med 1 000 ekstra kanter
vg4	16 MB	50 000	1 000 000	Tilfeldige koblinger
vgSkandinavia	192 MB	4 426 216	10 046 924	Norge, Sverige, Finland, Danmark

Vektingen i Skandinaviakartet er kjøretid målt i hundredels sekunder. Hvis programmet deres håndterer Skandinaviakartet bra, har dere et godt utganspunkt for siste øving også.

Om du vil finne kjøretiden for Oslo–Trondheim, er Oslo node nr. 143917, og Trondheim node nr. 347370.

Min testmaskin bruker 5 s på å behandle vg4, med Dijkstras algoritme og heapbasert prioritetskø. Estimert 12 min for den enklere varianten med en usortert tabell…

Tips, feil i boka

Det er en feil i javakoden i boka på side 191. Linje 2 med «`Vkant neste;`» skal ikke være med. «Vkant» arver nemlig «neste» fra «Kant», og skal derfor ikke deklarere «neste» på nytt. (Slik det står, får «Vkant» *to* variabler kalt «neste». En av typen «kant», og en av typen «Vkant». Ved kjøring vil typisk innlesing fra fil oppdatere det ene feltet, mens algoritmen bruker det andre feltet som bare inneholder null. Dermed virker ingenting, og feilen er veldig vanskelig å begripe medmindre man forstår detaljene i hvordan arv virker i Java.)

Etter å ha fikset dette, vil Java protestere her og der på at «neste» er en «Kant» når det forventes en «Vkant». Dette ordnes med typecasting: `(Vkant)objekt.neste`

Tips om algoritmene

Dijkstra og heap

For å bruke heap som prioritetskø, husk at Dijkstra trenger en min-heap, ikke maks-heap. Koden i boka må tilpasses ved å bytte om «<» og «>» i koden.

For å bruke en heap som prioritetskø, må det være sammenheng mellom heap'en og grafen.

En veldig enkel (og kjapp) måte er å bruke nodetabellen som heap, og utføre heap-operasjonene direkte på den. Men nodene vil da ende opp i en helt annen rekkefølge, som gjør det vanskelig å presentere resultatet til slutt. (Node[0] er ikke den opprinnelige første noden, osv) Dette kan løses ved å legge inn en «String navn» i nodestrukturen, dette navnet kan da brukes til utskriften til slutt.

Man kan bruke en annen tabell som heap. Den vil da inneholde referanser til nodene i nodetabellen. Dette vil fungere greit for å hente grafnoder ut av heapen. Men det er i utgangspunktet ikke mulig å gå andre veien. Når vi endrer distansen på en node i grafen, må vi gjøre en tilsvarende prioritetsendring i heapen. Men hvordan finne riktig node i heapen? Et søk vil øke kompleksiteten så mye at fordelen med heap går tapt! Men hver grafnode kan inneholde indeksnummeret til

sin plass i heapen, da kan programmet finne rett heap-node på en enkel måte. Hver gang en node flyttes i heapen må grafnodens indeksnummer oppdateres tilsvarende.

Prim

For prioritetskøen gjelder de samme betraktningene som for Dijkstras algoritme.