INF1020 – Algoritmer og datastrukturer

Forelesning 15: Gjennomgang av eksamen vår 2001 oppgave 3

Arild Waaler

Institutt for informatikk, Universitetet i Oslo

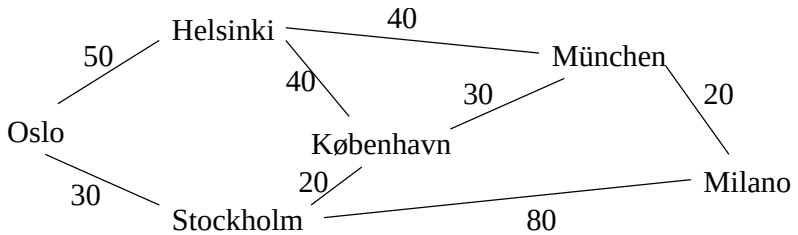
11. desember 2006

Oppgave 3–a. Antagelser i oppgaveteksten

Gitt en vektet, sammenhengende urettet graf.

- Nodene angir byer (flyplasser).
- En kant K mellom node A og node B angir at det går en direkte flyreise fra A til B og en direkte flyreise fra B til A.
- Vekten k til en kant mellom A og B angir *kostnaden* til en direkte flyreise mellom A og B (uansett vei).
- En *reiserute* fra A til B er en liste av kanter $K_1, \ldots, K_m$, $m \geq 0$, slik at man ved å følge kantene i angitt rekkefølge kommer fra A til B uten å besøke noen noder mer enn en gang.
- *Kostnaden* til reiseruten er $k_1 + \cdots + k_m$, der k_i er kostnaden til kant K_i.
- Kostnaden fra en node til seg selv (dvs. en reiserute uten kanter) er 0.
- *Prisen* fra A til B bestemmes som kostnaden til den reiseruten fra A til B som har minst kostnad.

Eksempelgraf



Oppgave 3–a

Bruk Dijkstras algoritme til å finne prisen fra Oslo til de andre byene.

	Initielt		Oslo blir kjent		Stockholm kjent		Sluttresultatet	
	Kjent	Avstand	Kjent	Avstand	Kjent	Avstand	Kjent	Avstand
Oslo	F	0	T	0	T	0	T	0
Helsinki	F	∞	F	50	F	50	T	50
Stockholm	F	∞	F	30	T	30	T	30
København	F	∞	F	∞	F	50	T	50
Munchen	F	∞	F	∞	F	∞	T	80
Milano	F	∞	F	∞	F	110	T	100

Oppgave 3–b. Antagelser i oppgaveteksten

Anta at prisen fra A til B er n. Det gis av og til flere mulige reiseruter fra A til B med pris n.

- Hvis N er en nabonode til B og prisen fra A til N er mindre eller lik n, tillater selskapet at man reiser via N uten tillegg i prisen.
- Man må da velge en reiserute fra A til N som har minimal kostnad, og etterpå reise direkte fra N til B.
- Vi sier da at ruten fra A til B via N er en *valgbar* reiserute fra A til B.
- Hvis C er på en valgbar reiserute fra A til B, kan man gå av i C og fortsette reisen senere uten tillegg i prisen.
- Vi sier da at man reiser fra A til B *med stopp i C*.
- Hvis C ikke besøkes i en valgbar reiserute fra A til B, kan man ikke reise fra A til B med stopp i C uten å løse ny billett.

Oppgave 3–b

(i) *Hvorfor er en reiserute med minimal kostnad alltid valgbar?*

- Ta en reiserute fra A til B med minimal kostnad og la N være den siste noden som besøkes før B.
- Merk at N kan være A.
- Da er denne ruten en reise fra A til B via N, dvs. at ruten er valgbar.

(ii) *Hva er prisen fra Oslo til München med stopp i Helsinki?*

- Prisen til München er 80
- Helsinki er nabo til München med pris ≤ 80.
- Helsinki er derfor på en valgbar vei fra Oslo til München.
- Svaret er derfor 80.

(iii) *Kan man reise fra Oslo til München med stopp i Milano?*

- Nei, for Milano ligger ikke på en valgbar reiserute fra Oslo til München.
- Dette fordi Milano er nabo til München, men prisen til Milano er større enn prisen til München.

Oppgave 3–c

Representasjonen av flyselskapets graf skal bruke nabolister (i en eller annen form). Definer en datastruktur for flyrute-grafen i form av Java-klasser.

```
class By {
  String navn;
  VektetKant nabo; // sortert etter stigende kostnad

  boolean kjent;   // til Dijkstra
  int pris;        // til Dijkstra (avstand)

  By sti;          // brukes i 3-d
  Kant stiliste;   // brukes i 3-e
}
```



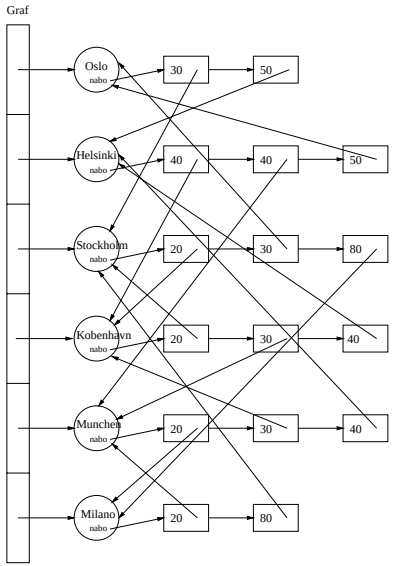
```
class Kant {
  Kant neste;
  By til;

  Kant( By b, Kant k ) {
    til = b; neste = k;
  }
}

class VektetKant extends Kant {
  int kostnad;

  VektetKant( By b, VektetKant k, int n ) {
    super( b, k );
    kostnad = n;
  }
}
```

Tegn så med "piler og bokser" hvordan en del av eksempelgrafen representeres i din datastruktur.



Oppgave 3–d

Programmer Dijkstras algoritme for å finne prisen fra en by til alle andre byer.

```
class Graf {
  By[] by;
  int antallUkjente;

  void initialiser( By start ) {
    antallUkjente = by.length;
      for ( int i = 0; i < by.length; i++ ) {
        by[i].kjent = false;
        by[i].pris = UENDELIG;
        by[i].sti = null;          // brukes i 3-d
        by[i].stiliste = null;     // brukes i 3-e
    }
    start.pris = 0;
  }
```



```
By minUkjent ( ) {
  int min = 0;
  for ( int i = 1; i < by.length; i++ ) {
    if ( !by[i].kjent && by[i].pris < by[min].pris ) {
      min = i;
    }
  }
  by[min].kjent = true;
  antallUkjente--;
  return by[min];
}

boolean flereUkjente () {
  return ( antallUkjente > 0 );
}
```



```
void dijkstra( By start ) {
  By v, w;
  VektetKant kant;
  initialiser( start );
  while ( flereUkjente() ) {
    v = minUkjent();
    for ( kant = v.nabo; kant != null; kant = kant.neste ) {
      int d = v.pris + kant.kostnad;
      w = kant.til;
      if( !w.kjent &&  d<w.pris ){
          w.pris = d;
          w.sti = v;
      }
    }
  }
}
```

Oppgave 3–e

Skisser en algoritme for en metode som tar inn tre byer A, B, C.

- *Metoden skal skrive ut alle valgbare reiseruter fra A til B med stopp i C, samt prisen.*
- *Vi tillater at C kan være lik* `null`*.*
- *Metoden skal da skrive ut alle valgbare reiseruter fra A til B, samt prisen.*

Merk: Dijkstras algoritme beregner avstandsfeltet `pris` og lagrer *én* reiserute til startnoden med minimal kostnad.

- Vi lagrer nå informasjon om *alle* valgbare reiseruter
- Selv om prisen til en node kan leses direkte fra verdien i nodens `pris`–felt, må gjøre bruk av `pris`–verdien til alle *nabonodene* for å finne frem til de valgbare reiserutene.
- "Via-noden" C hånderer vi under gjennomløpet av de valgbare rutene.

Modifikasjon av Dijkstras algoritme

- I hvert gjennomløp av Dijkstra gjør vi en ny node v kjent og undersøker enhver nabo w til v: Hvis vi får en kortere vei til w via v enn den vi hadde før, endrer vi w sin `sti` til v.
- Merk at `sti` alltid vil referere til *kjente* noder.
- For å kunne sporre alle reiserutene med minimal kostnad lar vi hver node har en variabel `stiliste`.
- Inn i denne lista legger vi alle nabonoder som man kan komme fra i en valgbar reiserute.
- Vi legger nå v til `w.stiliste` dersom prisen til v er mindre enn prisen til w *selv om* w *er kjent*.

Modifikasjon av Dijkstras algoritme: kode

```
for ( kant = v.nabo; kant != null; kant = kant.neste ) {
  int d = v.pris + kant.kostnad;
  w = kant.til;
  if( v.pris < w.pris )
    w.stiliste = new Kant( v, w.stiliste );
  if( !w.kjent &&  d<w.pris )
      w.pris = d;
  }
```


Utskrift av alle valgbare reiseruter fra A til B

- Veien blir ikke en enkel listestruktur, men en trestruktur.
- Bruk en lenket liste (eller stakk) `vei` av Kant–objekter
- Hvis den første noden i `vei` er startnoden A: skriv ut alle nodene i `vei` og backtrack.
- For hver node D i `B.stiliste`, legg D til `vei` og gå rekursivt dybde-først til D.

Utskrift av alle valgbare reiseruter via C

Kompliserende faktor: Vi skal bare skrive ut en vei dersom den enten inneholder C eller C er satt til null når metoden aktiveres:

- Sett en variabel `flagg` til true når vi kommer over noden C
- La `flagg` initielt være false dersom C ikke er null, ellers true.
- Bygg opp `vei` i de rekursive kallene idet vi traverserer trestrukturen.
- Skriv ut lista når vi kommer til A *og* flagget har blitt satt til true.

Oppgave 3–f

La K være en kant fra A til B med kostnad k. Kanten er feilpriset dersom det finnes en annen reiserute fra A til B med kostnad mindre eller lik k.

(i) *Finn alle feilprisede kanter i eksempelgrafen.*

- Kanten fra Stockholm til Milano er feilpriset.
- Kanten har kostnad 80, mens minste kostnad mellom de to byene er 70.

(ii) *Programmer en metode som finner alle feilprisede kanter i en graf.*

- Strukturen i løsningen er en ytre løkke som kaller opp en (variant av) Dijkstras algoritme for hver node:
- Kjør Dijkstras algoritme fra N.
- For hver kant i nabolisten til N, test om `kant.til.pris <= kant.kostnad`. I så fall er kanten feilpriset.
- Avskjæring: Stans algoritmen når kostnaden til `minUkjent()` er større enn kostnaden til alle nodens utkanter.
- Avskjæring: Sorter utkantene fra en node etter stigende kostnad.
- Vi kaller opp `testPris(N)` for hver node N.


```
void testPris( By start ) {
  int d;
  By v,w;
  VektetKant kant;
  VektetKant minNabo = start.nabo;

  initialiser( start );
  while ( flereUkjente() ) {
    v = minUkjent();
    while ( v.pris > minNabo.kostnad ) {
      // hvis minNabo er feilpriset, kan det oppdages naa:
      if ( minNabo.til.pris <= minNabo.kostnad ) {
        rapporterFeil( start, minNabo );
      }
      minNabo = minNabo.neste;
      if ( minNabo == null ) return;
    }
```

```
    for (kant = v.nabo; kant != null; kant = kant.neste ) {
      d = v.pris + kant.kostnad;
      w = kant.til;
      if ( d < w.pris ) w.pris = d;
    }
  }
}

void rapporterFeil( By start, VektetKant kant ) {
  System.out.print( start.navn + " -> " + kant.til.navn
                    + " feil: " );
  System.out.println( kant.til.pris + "/" + kant.kostnad );
}
```