Kapittel 13, Grafar

Uretta grafar (1)

- Ein uretta graf
  - Mengde nodar
  - Mengde kantar som er eit uordna par av nodar
- To nodar er naboar dersom dei er knytta saman med einkant
- Ein node kan ha kant til seg sjølv.
- Ein uretta graf er komplett dersom det finst ein kant mellom alle par av nodar

Uretta grafar (2)

- Talet på kantar i ein komplett graf med n nodar
  $(n-1) + (n-2) + \ldots + 1 = ((n-1) + 1) / 2 * (n-1) = n(n-1)/2$
- Ein sti er ein sekvens av kantar som knyter saman to nodar
- Lengda av ein sti er talet på kantar i stien
- Ei uretta graf er samanhengande dersom det finst ein sti mellom eit vilkårleg par av nodar
- Ein sykel er ein sti der første og siste node er den same og ingen av kantane kjem fleire gonger
- Ein graf utan syklar blir kall asyklisk

Retta grafar (1)

- Kantane er ordna par av nodar. Det betyr at kanten(a, b) er ulik (b, a)
- Ein sti i ein retta graf er ein sekvens av retta kantar som bind saman to nodar
- Ein retta graf er samanhengande dersom det finst ein sti mellom eit vilkårleg par av nodar (som før, men definisjonen på sti er endra)

Retta grafar (2)

- Dersom ein retta graf er asyklisk, er det mulig å ordne nodane slik om der er ein kant frå A til B, så kjem A før B. Blir kalla ei topologisk ordning
  - Topologisk ordning er viktig i forbindelse med prosjektplanlegging
- Eit retta tre er ein retta graf med ein spesiell node som er rot og følgjande eigenskapar
  - Ingen nodar har kant til rota
  - Alle ikkje-rot nodar har nøyaktig ein kant inn
  - Der finst ein sti frå rota til alle andre nodar

Nettverk (vekta graf)

- Eit nettverk er ein retta graf der kvar kant har ei vekt eller ein kostnad
- Kan kantar med eller utan retning

Grafgjennomgangar

- Breidde-først
  - Liknar nivåordning gjennomgang av tre
- Djupne-først
  - Liknar preorden gjennomgang av tre
- Skilnaden er at grafen ikkje har ei rot. I staden for tek vi utgangspunkt i ein startnode

Breidde-først

```
breidde-først(t)
    lagKø(q)
    q.leggTil(t)
    så lenge q ikkje-tom
            v = q.taUt()
            viss v ikkje er besøkt
                    legg alle v sine umerka naboar til q
                    merk v som besøkt
                    behandle v
    gjenta
```

Djupne-først

djupne-først(t)
    lagStabel(s)
    s.leggTil(s)
    så lenge t ikkje-tom
        v = s.taUt()
        viss v ikkje er besøkt
            legg alle v sine umerka naboar til s
            merk v som besøkt
            behandle v
    gjenta

Djupne-først, rekursiv

```
djupne-først(node x)
   besøk(x)
   resultatliste.leggTilBak(x)
   for kvar node y som er nabo til x
         viss y ikkje er besøkt
                djupne-først(y)
   gjenta
```

Teste om ein graf er samanhengande

- Uretta graf
  - Kan bruke breidde-først eller djupne-først og sjå om vi får besøkt alle nodane med ein vilkårleg node som startnode
- Retta graf
  - Kan bruke breidde-først eller djupne-først med alle nodar som startnode og sjå at vi får besøkt alle andre nodar
  - Finst meir effektive algoritmer

Minste spenntre (MST)

- Eit spenntre for ein graf inneheld alle nodane i grafen og ei undermengde av kantane slik at dei dannar eit tre
- Eit minste spenntre er eit spenntre det summen av vektene er minst mulig

Prims algoritme for MST

prim()

s = vilkårleg startnode
lagMinHaug(h)
legg alle kantar som går ut frå s til h
så lenge h ikkje tom
    e = h.fjernMin()
    viss ikkje begge endepunkta til e er med i MST
        - la w vere endepunktet til e som ikkje er med i MST
        - legg e til MST
        - legg alle kantar mellom w og nodar som ikkje er med i MST til h
gjenta

Kortaste sti

- To muligheiter
  - Færrast mulig kantar
    - Breidde-først
  - Billegaste stien i ein vekta graf
    - Dijkstras algoritme
      - Effektivi algoritme
      - Liknar Prims algoritme
      - Detaljar er ikkje pensum

Implementasjon av grafar

- Operasjonar
  - Legge til og fjerne nodar
  - Legge til og fjerne kantar
  - Grafgjennomgangar (breidde-først og djupne-først)
  - Storleik, erTom, toString, find
  - Kortast sti frå ein node til ein eventuelt alle andre
  - Bestemme om to nodar er naboar
  - Finne alle naboane til ein node
  - Konstruere eit minste spenntre
  - Teste om grafen er samanhengande
- Skisserer to implementasjonar
  - Nabomatrise
  - Naboliste

Naboliste (adjacency list)

- Todimensjonal Boolsk tabell av storleik n x n der n er talet på nodar i grafen
- Ordnar nodane frå 0 til n-1
- Elementet i posisjon (i, j) er sann viss der er ein kant frå i til j, usann elles
- Om grafen er uretta, vil elementa (i, j) og (j, i) ha same verdi. Matrisa blir symmetrisk
- Kan også brukast for nettverk eller vekta grafar.
  - Har ei heiltalsmatrise, eventuelt flyttalsmatrise
  - Brukar 0, eventuelt -1 for å markere at der ikkje er ein kant

Oppgåve

- Nodar
  - 11, 12, 13, 14, 15, 16, 17
- Kantar
  - (11,12), (11,14), (12,13), (12, 14), (13, 15), (13,17), (14,16), (14,17), (15,16), (15,17) (16,17)
- Teikn opp grafen
- Vis korleis den blir representert i ei nabomatrise

Naboliste

- Kvar node har ei liste som inneheld alle naboane
- God datastruktur når vi treng finne alle naboane til ein node og kvar node ha ”få” naboar
  - Sentral operasjon både i Prims algoritme for minste spenntre og Dijkstras algoritme for å finne kortaste sti frå ein node til alle andre
- Ikkje pensum