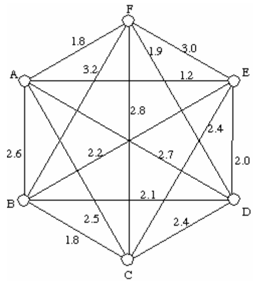
Uke 10,
Forelesning 1

Korteste vei algoritmer med negative kanter

Aktivitets grafer og handelsesgrafer
Dybde-først søk
Løkkeleting og omvendt topologisk sortering

TEMA: GRAFER



Fortsetter med Grafer…

OVERSIKT – *Uke 10, Forelesning 1 (W10.L1)*

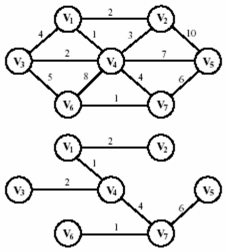
Vi tar opp: **GRAFER**

- Minimale spenntrær
  - — Prim (Kapittel 9.5.1)
  - — Kruskal (Kapittel 9.5.2)
- Korteste vei alle-til-alle
  - — Floyd (kapittel 10.3.4)

- **Huffman koder (kapittel 10.1.2)**

Minimalt spenntre

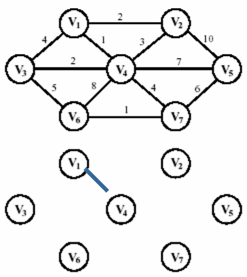
- Et minimalt spenntre for en **urettet** graf G er et tre bestående av kanter fra grafen, slik at alle nodene i G er forbundet til lavest mulig kostnad.
- Minimale spenntrær eksisterer bare for sammenhengende grafer.
- Generelt finnes det flere minimale spenntrær for samme graf.

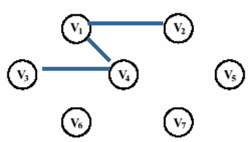


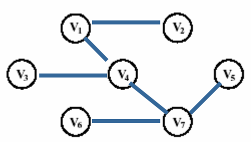
??? Hvor mange kanter får spenntreet i det generelle tilfellet?

Prims algoritme

- Treet bygges opp gradvis. I hvert steg legges en kant (og dermed en tilhørende node) til treet.
- På ethvert tidspunkt har vi to typer noder: De som er med i treet, og de som ikke er det.
- Nye noder legges til ved å velge en kant (u, v) med **minst** vekt slik at u er med i treet, og v ikke er det.





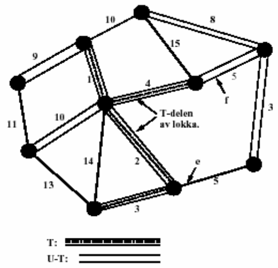


GRAFER – *Minimalt spenntre, Forelesning 1 (W10.L1)*

Hvorfor virker Prim?

Løkke-lemmaet:
Anta at T er et spenntre for en graf, og at kanten e ikke er med i treet T.
Om vi legger kanten e til treet T, vil det dannes en entydig bestemt enkel løkke.
Om vi fjerner en vilkårlig av kantene i denne løkken, vil vi igjen ha et spenntre for grafen.

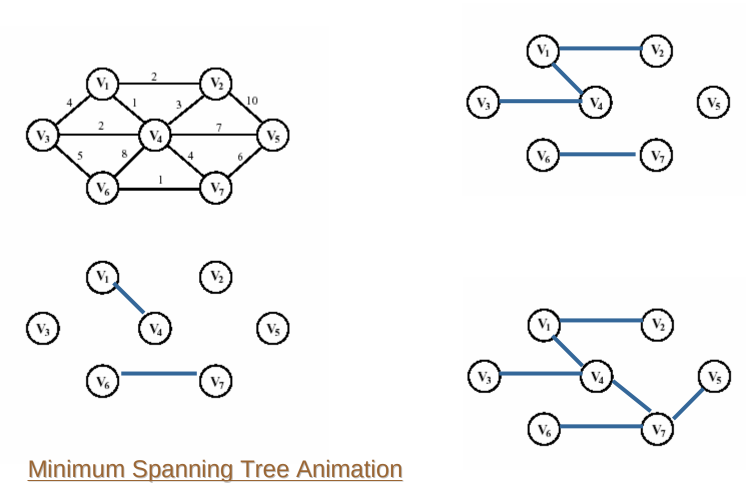
Prim-invarianten:
Det treet T som dannes av de kantene (og deres endenoder) vi til nå har plukket ut, er slik at det finnes et minimalt spenntre for grafen som inneholder (alle kantene i) T.



Minimum spanning tree algorithms

Kruskals algoritme

- Ser på kantene en etter en, sortert etter minst vekt:
  - En kant aksepteres hvis den ikke fører til noen løkke.
- Kruskals algoritme opprettholder dermed en skog, (en **samling** trær):
  - Initielt: $|V|$ trær med en node hver.
  - Legger til en kant: To trær slås sammen.
  - Ved terminering: Bare ett tre.
- Bruker disjunkte sett (kapittel 8):
  - Invariant: To noder tilhører samme sett hviss de er sammenhengende i den nåværende spenn-skogen.
  - Å velge en kant (u, v) tilsvarer å gjøre en union på u og v.
- Sorteringen av kantene gjøres mest effektivt ved å bruke en prioritetskø. Gjentatte **deleteMin** gir da kantene i den rekkefølgen de skal testes.



Minimum Spanning Tree Animation

Dynamisk programmering

- Brukes først og fremst når vi ønsker optimale løsninger.
- Må kunne dele det globale problemet i delproblemer.
  - Disse løses typisk ikke-rekursivt ved å lagre del-løsningene i en tabell.
- En optimal løsning på det globale problemet må være en sammensetning av optimale løsninger på (noen av) delproblemene.
- Vi skal se på ett eksempel: Floyds algoritme for å finne korteste vei alle-til-alle.

Korteste vei alle-til-alle

Vi ønsker å beregne den korteste veien mellom ethvert par av noder i en rettet, vektet graf.

- Grunnleggende idé:
  Hvis det går en vei fra node i til node k med lengde ik, og en vei fra node k til node j med lengde kj, så går det en vei fra node i til node j med lengde $ik + kj$.
- Floyds algoritme: Denne betraktningen gjentas på en systematisk måte for alle tripler i, k og j:
  - Initielt: Avstanden mellom to noder er det samme som vekten på kanten mellom dem, eventuelt uendelig.
  - Steg 0: Ser etter mulige forbedringer ved å velge node 0 som mellomnode.
  - Steg k: Avstanden mellom to noder er den korteste veien som bare bruker nodene $0, 1, \ldots, k$.

Department of Informatics, University of Oslo, Norway
INF110 – Algorithms & Data Structures

Floyds algoritme

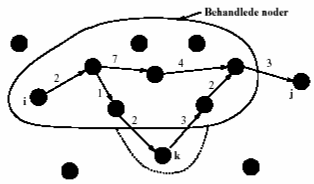
```
public static void kortesteVeiAlleTilAlle(int[ ][ ] nabo,
                                          int[ ][ ] avstand,
                                          int[ ][ ] vei) {
    int n = nabo.length;

    // Initialisering:
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            avstand[i][j] = nabo[i][j];
            vei[i][j] = -1;             // Ingen vei foreløpig
        }
    }

    for (int k = 0; k < n; k++) {
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                if (avstand[i][k] + avstand[k][j] < avstand[i][j]) {
                    // Kortere vei fra i til j funnet via k
                    avstand[i][j] = avstand[i][k] + avstand[k][j];
                    vei[i][j] = k;
                }
            }
        }
    }
}
```

Tidsforbruk:

Department of Informatics, University of Oslo, Norway
INF110 – Algorithms & Data Structures

Hvorfor virker Floyd?

Floyd-invarianten: *avstand*[*i*][*j*] vil være lik lengden av den korteste veien fra noden *i* til noden *j*, som har alle sine indre noder behandlet.



FØR:
A(i,j)=16
A(i,k)=5
A(k,j)=8

ETTER:
A(i,j)=13
....
....

- http://csci.biola.edu/math112/shortestPaths.pdf is a website with nice explanation of dynamic programming

- Huffman codes

Oppsummering — datastrukturer

Grafer

- Dette er den mest generelle datastrukturen — «alt» kan representeres som grafer
- Enkle operasjoner blir kostbare «Finn» er typisk en $O(n)$ operasjon
- Grafer brukes oftest til kompliserte, men ofte statiske, datastrukturer (her finnes mange unntak)
- DAG (rettet asyklisk graf) er et viktig spesialtilfelle som bl.a. brukes til aktivitetsanalyser og i Javas klassebibliotek
- Trær er et viktig spesialtilfelle av DAG

Trær

- Velegnet for dynamiske datastrukturer hvor *ordning* av dataelementene er viktig
- «Finn» er typisk en $O(\log n)$ operasjon, og det er «SettInn» og «TaUt» også
- I *søketrær* er dataelementene fullstendig ordnet (sortert)
- B-trær er søketrær beregnet på stor dynamikk og data lagret på langsomt medium (som disk)
- Hvis vi ikke trenger ordning (> og <), er hashing raskere enn søketrær
- Rekkefølgen i rekursiv traversering av binærtrær:
  prefix – vsub – infix – hsub – postfix
- En «heap» er ikke et søketre, men en effektiv implementasjon av en prioritetskø optimalisert for å finne minste element
- *Lister* er spesialtilfeller av trær

ALMIRA KARABEG foreleser!
Vi introduserer NP-komplethet

- NP-komplethet (kapittel 9.7)