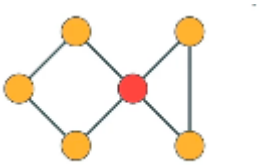
Introduksjon

I sammenhengende grafer så har vi såkalte “kritiske feilpunkter”. Disse er noder hvor om vi fjerner disse, så mister vi sammenkoblingen til ulike deler av grafen og omhandler “bikonnektivitet”.

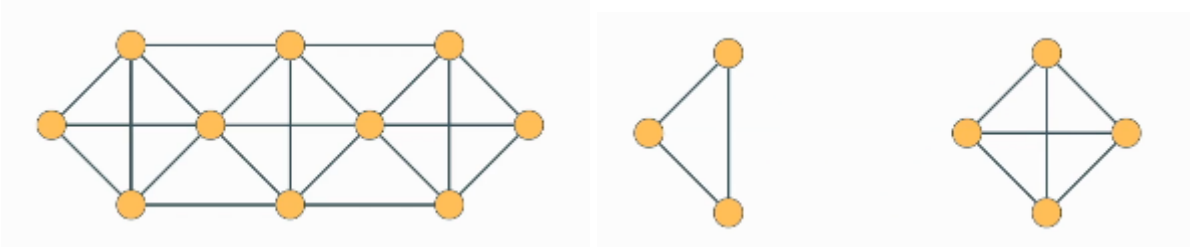


Her ser vi at, om vi fjerner den røde noden så vil vi ikke koble de to delene sammen.

I tillegg finnes det deler av grafen hvor alle noder innenfor den en gitt del, kan nå hverandre og disse kalles for “sterkt sammenhengende komponenter”.

<u>K-sammenhengende grafer</u>

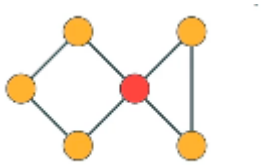
Definisjon: En sammenhengende graf “G” er “k-sammenhengende” eller “k-node-sammenhengende”, hvis “G” forblir sammenhengende om færre enn “k” noder blir fjernet.



Grafen over er 3-sammenhengende ettersom hvis vi fjerner 3-noder, så slutter grafen å være sammenhengende. Men de grafene som er 2-sammenhengende vi er ute etter som er knyttet til bikonnektivitet.

2-sammenhengende grafer/bikonnektive grafer

Definisjon: En graf “G” er 2-sammenhengende/bikonnektive, hvis “G” forblir sammenhengende ved fjerning av en vilkårlig node. En annen måte å formulere dette på: Hvis alle par av noder “u” og “v” har to distinkte stier(deler ingen kanter eller noder) mellom dem.



Denne grafen er ikke 2-sammenhengende ettersom om vi fjerner den “rød” noden så blir ikke grafen sammenhengende. Hensikten ved bikonnektive grafer, er å unngå at vi har slike kritiske feilpunkter som den “rød” noden over. Men noder som ved fjerning, fører til at grafen blir ikke-sammenhengende kalles for “separasjonsnoder”.

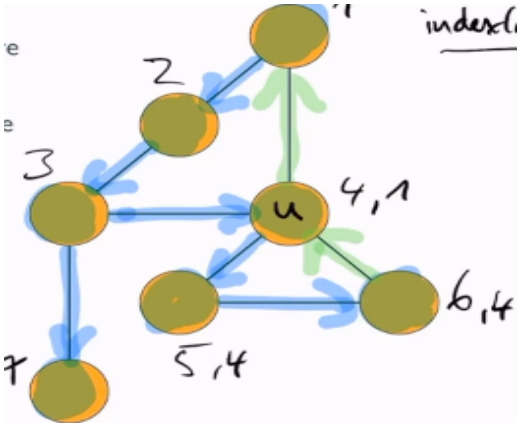
Separasjonsnoder

Definisjon: Noder ved fjerning som fører til at grafen blir ikke-sammenhengende kalles for “separasjonsnoder”. Disse nodene tilsvarer da “kritiske feilpunkter”.

Hvordan kan vi finne separasjonsnoder i 2-sammenhengende grafer?

Det vi gjør, er å foreta et DFS til å lage et spenntre. Deretter gjør vi følgende:

- Vi indekserer rekkefølgen nodene blir besøkt.
- Angir at “Discovery edges” (de blå) i treet er kanter som fører til nye kanter.
- “Back edges” (de grønne) er kanter som fører tilbake til allerede besøkte noder.



Gjennom dette bildet så er det mulig å identifisere hvilke noder som er separasjonsnoder.

En mer presis formulering:

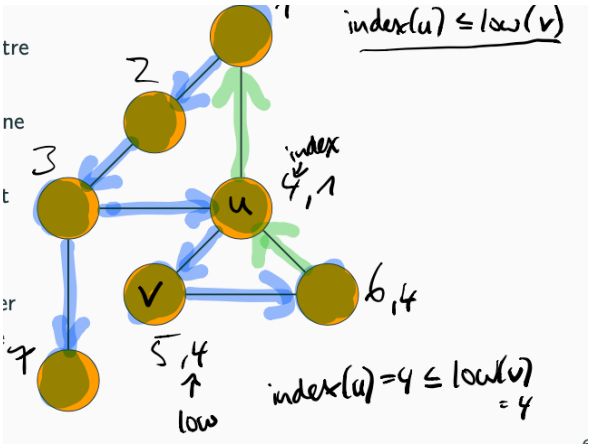
En node “u” er en separasjonsnode hvis:

- “u” er rot i “T” og har mer enn ett barn, eller
- “u” ikke er roten og har et barn “v” slik at ingen etterkommere “v” (inkludert “v” selv) har en back-egde til en forgjenger av u.

Det siste punktet kan vi sjekke ved hjelp av low-nummer. Dette “low-nummeret” til en node er definert som indeksen til den noden med lavest indeks som kan nås ved å følge:

1. 0 eller flere kanter i T (discovery edges) etterfulgt av
2. 0 eller 1 back-edge

For å gi en mer simplistisk eller enklere forklaring av siste punktet er å se dette tilfellet:
“u” er en separasjonsnode hvis det finnes en kan <u,v> i T, slik at inde(u) <= low(v)



Her ser vi et eksempel hvor node med indeks “4”, er en separasjonsnode. Dette kan man se også, men hvis vi følger regelen vi har, så blir vi istand til å finne separasjonsnoden ved å følge index(u) <= low(v).

Algoritmen for å finne separasjonsnoder kalles “Hopcroft-Tarjan” og er vist undet:

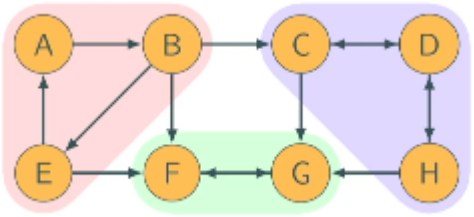
Input: En graf G, en startnode u og dybden

Output: Alle separasjonsnoder i inputgrafen “G”

```
Procedure HopcroftTarjan(G,u,depth)
        visited[u] = true
        low[u] = index[u] = depth
        childCount = 0
        for each edge (u,v) in G do
                if visited[v] = false then
                        childCount = childCount +1
                        HopcroftTarjan(G,u,depth + 1)
                        low[u] = min(low[u], low[v])
                        if index[u] <= low[v] then
                                ç
                else
                        low[u] = min(low[u], index[v])
        if index[u] = 1 then
                if childCount > 1 then
                        low[u] = index[u] = depth
        return sep_vertices
```

Sterkt sammenhengende komponenter

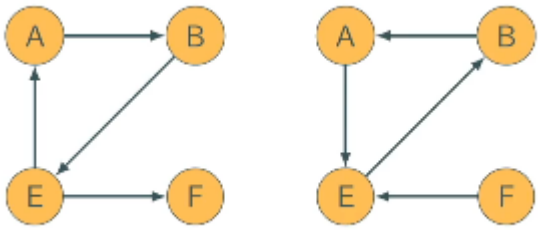
Definisjon: En rettet graf er sterkt sammenhengende hvis det finnes en sti mellom alle par av noder.



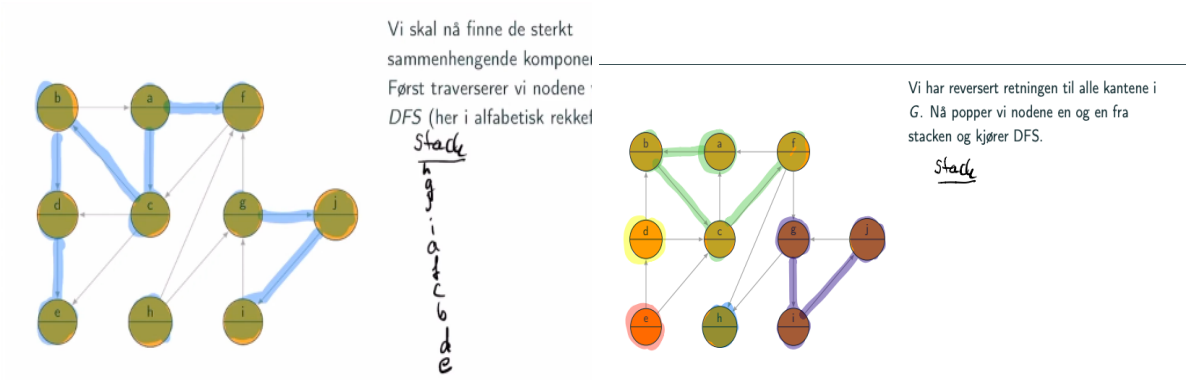
Bildet over er et eksempel på en sterkt sammenhengende graf med slike komponenter hvor {A,B,E} utgjør et komponent, {C,D,H} utgjør et annet komponent og tilslutt {F, G}. Vi kan også lage en graf hvor disse sterkt sammenhengende komponenter sees som noder hvor vi har kanter mellom hver komponent hvor disse grafene vil tilsvare en rettet asyklisk graf(DAG).

Hvordan finne sterkt sammenhengende komponenter til G?

For å finne de sterkt sammenhengende komponenter til “G”, skal vi bruke DFS ettersom vi må foreta en form for traversering i grafen. Ideen bak dette, er at de sterkt sammenhengende komponentene til “G” skal forbli uendret hvis vi reverserer alle kantene.



F.eks, så ser vi at {A,B,E} er en sterkt sammenhengende komponent. Om vi reverserer grafen så forblir denne komponenten allikevel som understreker at den er sterkt sammenhengende. For algoritmen så vil vi da kjøre to DFS søk. Den første DFS´en kjøres på en “G” hvor vi puser noder på en stack når de er ferdig behandlet(gått gjennom alle noder vi kan traversere igjennom). Imens den andre så kjører vi DFS på den reverserte grafen av “G” hvor reverseringen vil si den samme grafen bare med reverserte kanter. For denne DFS´en så begynner vi fra toppen av stacken og popper fra stacken. Slik fungerer algoritmen.



Den til venstre utgjør DFS hvor vi pusher nodene i stacken. Den andre utgjør DFS hvor vi har reversert grafen og søker ved at vi popper fra stacken og traverserer fra øverste element mot nedover.