INF 4130
10. november 2011

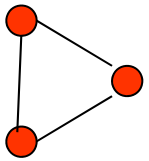
Stein Krogdahl

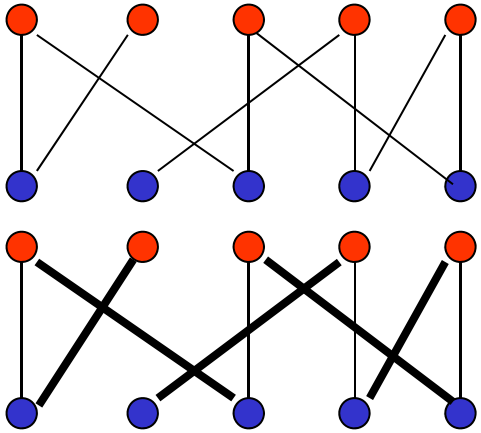
- Oblig 3 ligger ute, frist 25/11. Har oppgave fra dagens stoff
- Det er mye (og dermed liten) tekst på disse foilene.
  - Ment å være greit for repetisjon
- Dagens tema: Kapittel 14:
  - Matchinger i (urettede) grafer (matching = pardannelse)
  - Flyt i nettverk (nettverk = rettede grafer med kapasiteter etc.)
- Dagens tema er kraftig forbundet med:
  - Konveksitet, polyedre med heltallige hjørner etc., og dette ser man nærmere på i mer matematisk rettede kurs (bl.a i INF-MAT 3370: Lineær optimering). Vi går ikke inn på det her.

”Matchinger” i urettede bipartite grafer, kap. 14.1

Bipartit graf = Node-mengden kan deles i to (X og Y), slik at alle kanter går mellom en node i X og en i Y.

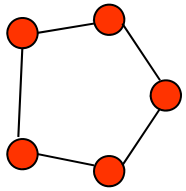
Samme som: To-fargbar graf eller: Graf uten (slike) ”odde løkker”:





Nodemengden X, f.eks. håndverkere

Nodemengden Y, f.eks. dagens jobber



Kantene: Hvem har kompetanse til de forskjellige jobber?

Vi klarte her å finne en ”perfekt matching”, som altså gjør at vi kan få utført alle jobbene denne dagen

Ofte er |X| = |Y|, men de er også ofte ulike i størrelse. Da snakker man ikke om *perfekte* matchinger

Masse anvendelser, f.eks.:

(1) Gruppelærere (X) som har ønsker til grupper (Y). Kan alle få hver sin gruppe?
(2) En klasse skal danne ”lag” med en gutt og en jente, og læreren vet hvem som jobber godt sammen.
(3) …

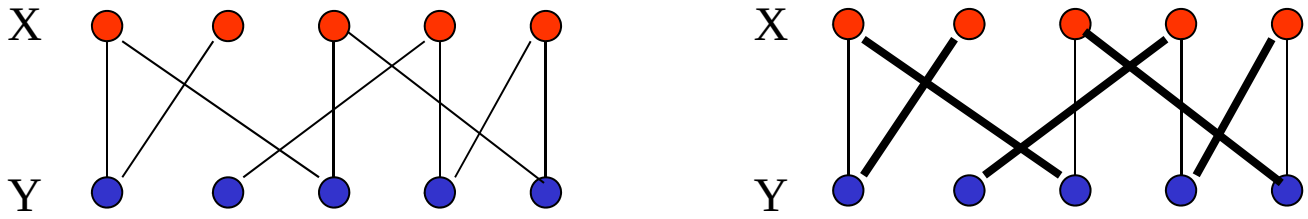
Noen varianter av problemet:

- Selv om vi ikke kan finne en *perfekt* matching, kan vi være interessert i å finne en *størst mulig* matching (spesielt om |X| ikke er lik |Y|).
- Det kan være ”vekter” på kantene, og vi kan være interessert i å finne en ”tyngst mulig” matching (eller tyngst mulig av de perfekte).

Halls Teorem:

Når kan vi finne en perfekt matching?

En bipartit graf med en perfekt matching (krever altså |X| = |Y|):

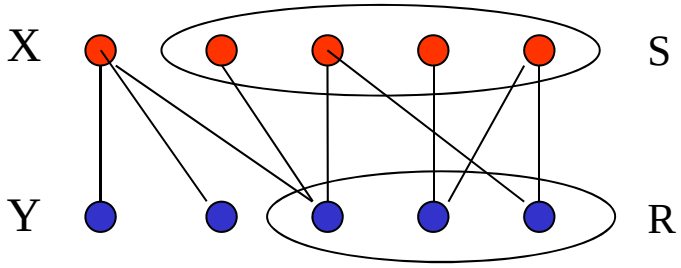


Under: En undermengde S av X er forbundet (bare) med nodemengden R i Y, og R har færre noder enn S. Da finnes opplagt ingen perfekt matching. Men dette gjelder også andre veien:

Halls Teorem: Det finnes en perfekt matching **hvis og bare hvis** det ikke finnes noe utplukk S av X slik at R har færre noder enn S.

Bevis den "lette" veien, som antydet over: Om det *finnes* en slik S (slik at R er mindre enn S) så finnes det opplagt ingen perfekt matching. Vi kan ikke få matchet alle i S med en egen i R.

Bevis den "vanskelige" veien: Den "ungarske" algoritme vil enten gi en perfekt matching, eller den vil vise oss (når den stopper før vi har en perfekt marching) en slik S.



Boka: R = Gamma(S)

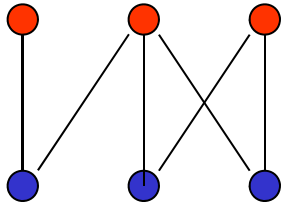
Den naive ”grådighets-algoritemen” virker ikke

Problem: Gitt en bipartit urettet graf. Finn, om mulig, en ”perfekt” matching.

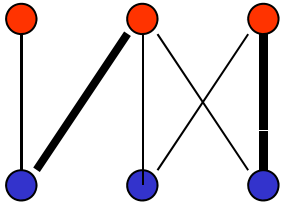
Grådighets-tilnærmelse (virker for noen få problemer, men ikke her):

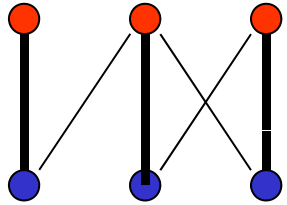
> ”Se på kantene i en eller annen, og ta en kant inn i matchingen om den ikke har en felles node med noen av kantene som allerede er med i matchingen”

Eksempel på at grådighets-tilnærmelsen ikke virker her:



Gitt den øverste grafen. Grådighet kan, etter to steg, gi matchingen under til venstre. Det finnes opplagt en matching med tre kanter (under til høyre), men den til venstre kan ikke utvides ved enkel grådighet!





Kommentar: I forbidelse med grådighet har elementene oftest også en vekt, og man vil f.eks. minimere vekten. Da ser man på elemnetene i sortert rekkef.

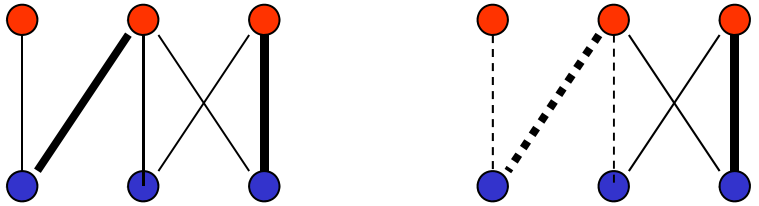
Et sted der slik grådighet faktisk virker er når en vil finne det letteste (eller tyngste) spenntreet i en sammenhengende urettet graf med vektede kanter:

> ”Se på kantene i rekkefølge av stigende vekt, og ta med de som ikke danner en løkke med de nodene som allerede er plukket ut” (Kruskals algoritme).

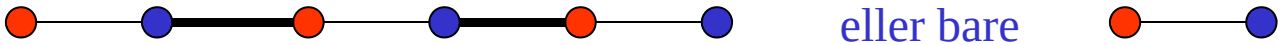
Den ungarske algoritme for å finne en perfekt matching

Antar altså at |X| = |Y|

- Det viser seg imidlertid at om vi, i stedet for å lete etter helt ”ledige” kanter, leter etter ”forbedringsveier”, så vil vi helt sikkert finne en slik vei, dersom en større matching i det hele tatt eksisterer.
  - Dette kan lett vises direkte, men vi gjør det slik at vi samtidig viser Halls teorem
- En slik forbedringsvei for den venstre matchingen M er stiplet i den høyre:

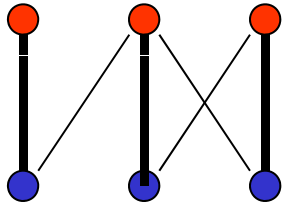


- Forbedringsvei P:
  - En ”alternerende vei” (annenhver kant er med og ikke med i matchingen)
  - Begge endenodene er ”umatchet” (er ikke ende-node i noen kant i matchingen)



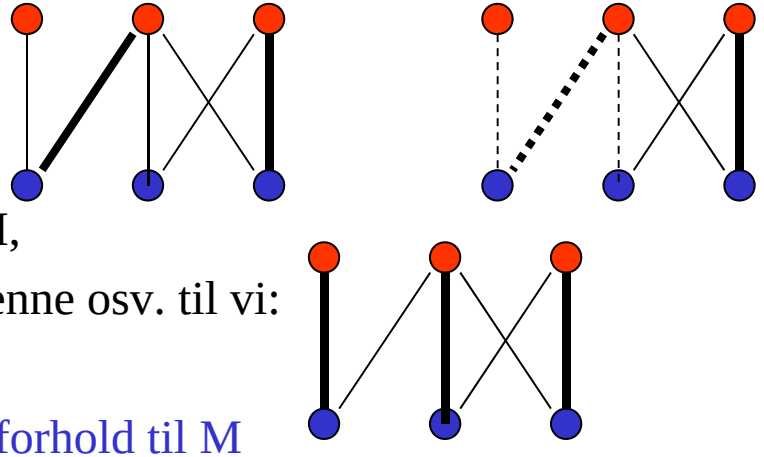
- Vi ”bruker” en forbedringsvei ved å bytte kanter i matchingen langs forbedringsveien:

  - Dette må opplagt føre til en ny matching, som er én større.
  - I tilfellet over får vi den til høyre (som skrives $M \oplus P$):



Hvordan finne mulige forbedringsveier?

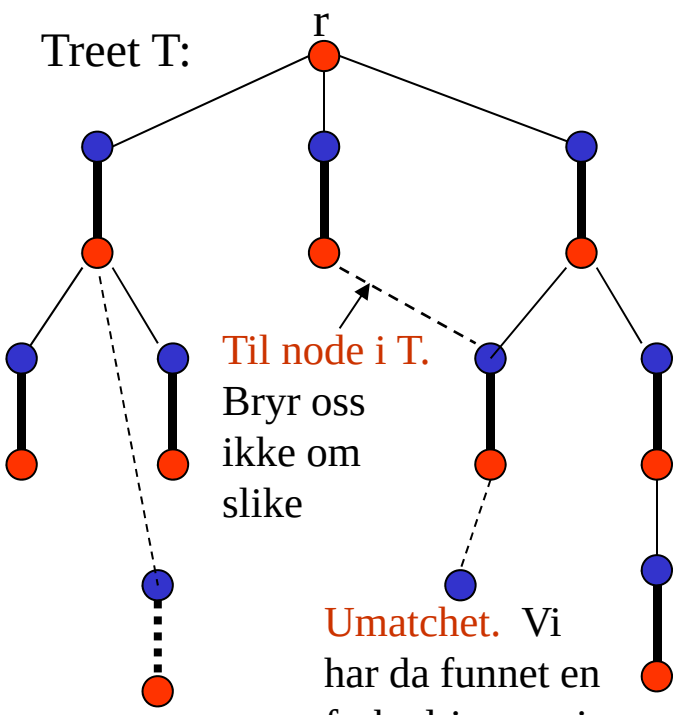
- Den ungarske algoritme går ut på å:
  - starte med en tom matching,
  - så lete etter en forbedringsvei,
  - så ”bruke” denne til å få en større matching M,
  - så finne ny forbedringsvei i.f.t. M og bruke denne osv. til vi:
  - enten har en perfekt matching
  - eller til vi ikke finner noen forbedringsvei i forhold til M
    - I siste tilfelle vil situasjonen forhåpentligvis vise oss en undermengde S i X som er forbundet med mengden R= Gamma(S) i Y, slik at R er mindre enn S.
    - Og dermed vise oss at det ikke finnes noen perfekt matching!



- Det generelle steget i den ungarske algoritme går altså ut på å:
  - ha en foreløpig (ikke-perfekt) matching M
  - så prøve å finne en forbedringsvei som vi kan bruke til å lage en matching som har én kant mer. (Om det er en helt ”ledig” kant kan vi velge den som forb.vei)
- Søket etter en slik forbedringsvei gjøres generelt slik (figur neste foil):
  - Velg en umatchet node ”r” i X. Den skal bli roten av det nye treet T, der alle veier ut fra roten skal være alternerende veier (det blir de ”automatisk” i en bipartit graf).
  - Vi har da funnet en forbedringsvei dersom en forgrening av treet kan få kontakt med en umatchet node i Y.

Steget i den ytre løkka i den ungarske algoritme

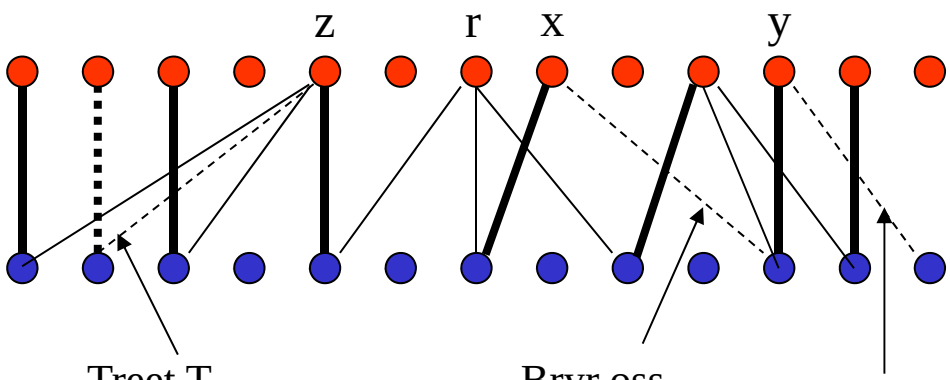
- Vi antar altså at vi står med en ikke-perfekt matching M, og at vi vil lete etter en forbedringsvei.
- Dette steget skal bygge et ”alternerende tre” T, og ved starten består treet T bare av en rotnode ”r”, som er en umatchet node i X (og en slik kan alltid finnes når M ikke er perfekt og |X| = |Y|)
- Dette treet bygges slik: Ut fra den valgte roten bygger vi et sammenhengende alternerende tre T fra roten r. Dette gjøres i en løkke med følgende steg:
- Steg i trebyggingen: Vi leter opp en kant som går ut fra en *rød* node i T (altså en node i X) og som går til en node (i Y!) som *ikke er med i T*.
- Om vi finner en slik kant er det to tilfeller:
  1. Kanten går til en umatchet node i Y, da har vi *funnet en forbedringsvei*, og vi kan bruke denne til å øke M, og (om vi da ikke har en perfekt matching) starte på en ny trebygging.
  2. Kanten går til en matchet node i Y, og da tar vi også med i T den tilhørende kanten i M. Treet blir altså utvidet med to kanter/noder.

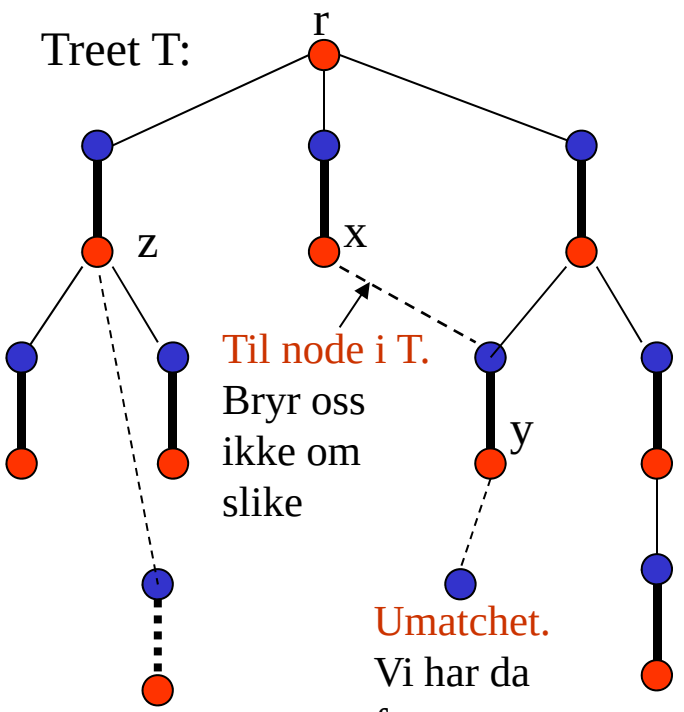


Steget i den ungarske algoritme - 2

Treet ser greit og ryddig ut til høyre. Merk at der er bare treets nye og potensielle kanter er tegnet. Det kan være mange flere kanter og noder!

Men treet kan selvfølgelig også tegnes inn i den bipartite grafen. Da tar det seg slik ut (der er *alle* noder, men ikke alle kanter, tegnet inn):





Til node i T. Bryr oss ikke om slike

Den blå er matchet. Vi tar da også med i T den tilhørende M-kanten

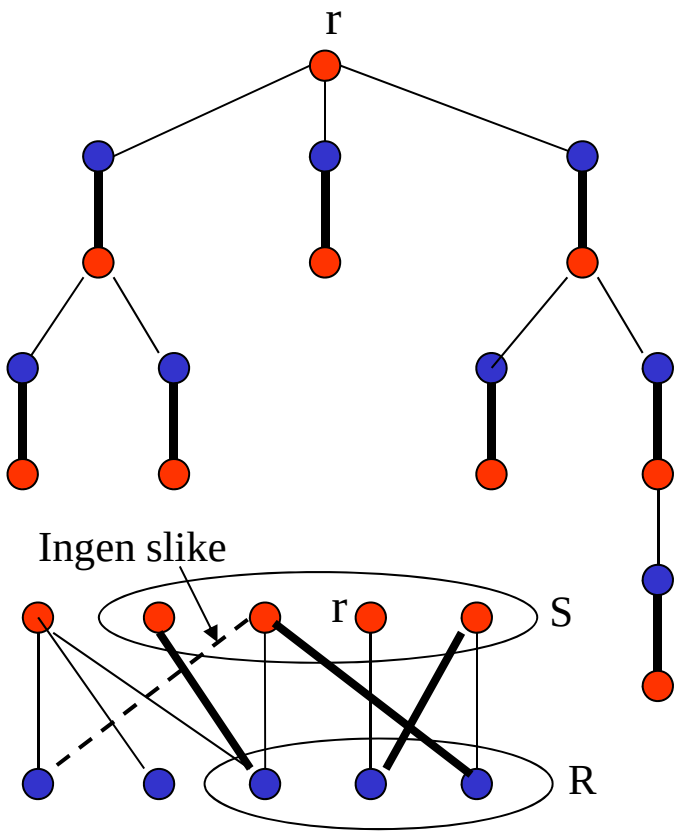
Umatchet. Vi har da funnet en forbedrings-vei. Vi "bruker" denne, og får en større matching

Avslutning av den ungarske algoritme

uten å finne noen perfekt matching

- Anta at algoritmen stopper fordi vi ikke finner en kant fra en rød node i T til en (blå) node utenfor T. Da finner vi altså ingen forbedringsvei, og vi ønsker da å finne et bevis på at *ingen* perfekt matching finnes:
  - Ønske: En delmengde S av nodene i X (røde) som er slik at de nodene R den er forbundet med i Y er færre enn i S.
  - Som S velger vi da rett og slett de røde nodene i T. Av dem er det én mer enn antall M-kanter i treet.
  - Vi lar så R være de blå nodene i T. R har opplagt én node færre enn S (like mange som kanter fra M i treet)
  - Vi påstår nå at nodene i S ikke har kanter til noen andre blå noder enn de i R, altså at R=Gamma(S). Begrunnelse:

    Algoritmen har stoppet nettopp fordi det ikke finnes kanter fra røde noder i treet til blå noder utenfor treet.



NB: Det over er to forskjellige trær!

Dermed er også Halls Teorem vist:
Denne algoritmen kan kjøres på enhver bipartit graf med |X|=|Y|, og den vil gi enten en perfekt matching eller en S slik at R=Gamma(S) er mindre enn S.

Varianter over problemstillingen

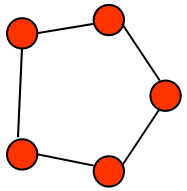
- Sett på til nå:
  - Finn en perfekt matching i en bipartit graf (eller vis at en slik ikke finnes)
  - Et programskisse av denne algoritmen er gitt på side 422/423

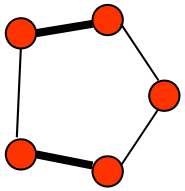
- Andre spørsmål (som også kan løses greit):
  - Finn en matching med flest mulig kanter (og da behøver ikke nodemengdene X og Y være like store)
    - Skal dere se på som gruppeoppgave neste uke
  - Gitt vekter på kantene: Finn en perfekt matching med størst mulig vekt
    - Står i boka, men vi tar ikke den med i pensum (kap 14.1.3)
  - Flyt i ”nettverk” (der matching-problemet for bipartite grafer fremkommer som et spesialtilfelle)
    - Den sammenhengen skal dere også se på på gruppene neste uke
    - Flyt i nettverk skal vi se på straks.
  - Generalisering til generelle grafer (ikke bare bipartite)
    - Se neste foiler

Matching i grafer som *ikke* er bipartite

Disse tre foilene er til en viss grad pensum, se under.

Har ”odde” løkker:





”Vrange” for matching

Generaliseringer av det bipartite matching-problemet:

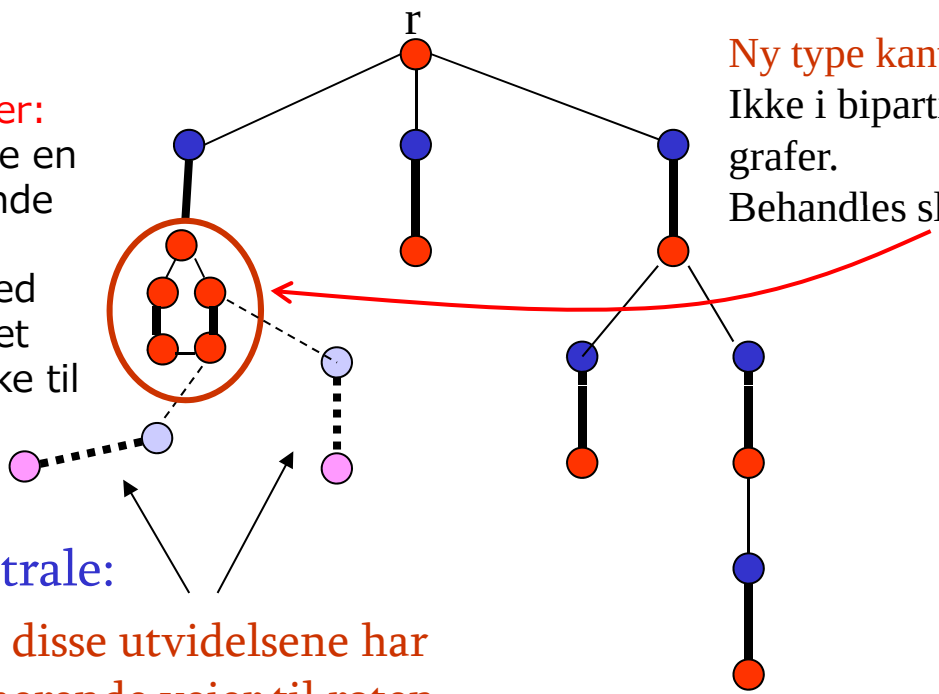
- Gå over til generelle grafer, *ikke* bare bipartite grafer
- Still de tilsvarende spørsmål angående matchinger:
  - Finn en perfekt matching (eller vis at en slik ikke finnes)
  - Finn en matching med flest mulig kanter
  - Med vekter på kantene: Finn en perfekt matching med størst mulig vekt
- Alle disse kan også løses i polynomisk tid
- Algoritme for det generelle matching-problemet:
  - Her finnes også greie algoritmer (skisse neste to foiler), men de er mer krunglete å bevise
  - Pensum er bare å *vite* at en slike algoritmer finnes, og at matching-problemet i generelle grafer derved også kan løses i polynomisk tid.

Steget i den ”utvidede ungarske algoritme”

Nye elemter i algoritmen:

- Vi fjerner all farging mellom tre-byggingene
- Starter på tilfeldig umatchet node, farges rød!
- Når grafen ikke er bipartit kan det også opptre kanter i treet fra røde til røde noder, slik som kanten (u,v) i figuren. Denne danner da en odde løkke med resten av treet.
- Den behandles rett og slett ved at vi ”snurper sammen” den odde løkka (inklusive indre kanter) til en ny ”stor” rød node.

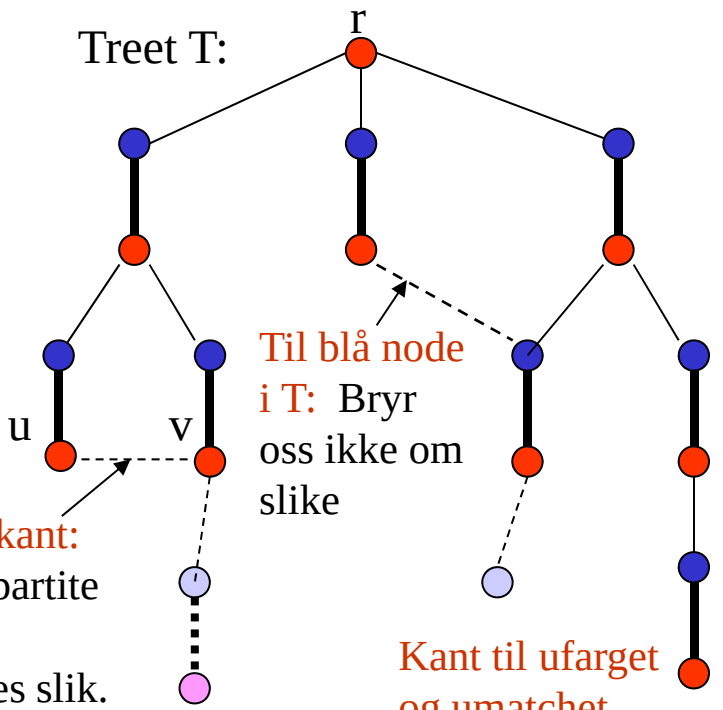
  (Eller: Farg *alle* nodene i den odde løkka røde)



Røde noder:
De har alle en alternerende vei som starter med en matchet kant tilbake til roten

Det sentrale:

Begge disse utvidelsene har alternerende veier til roten



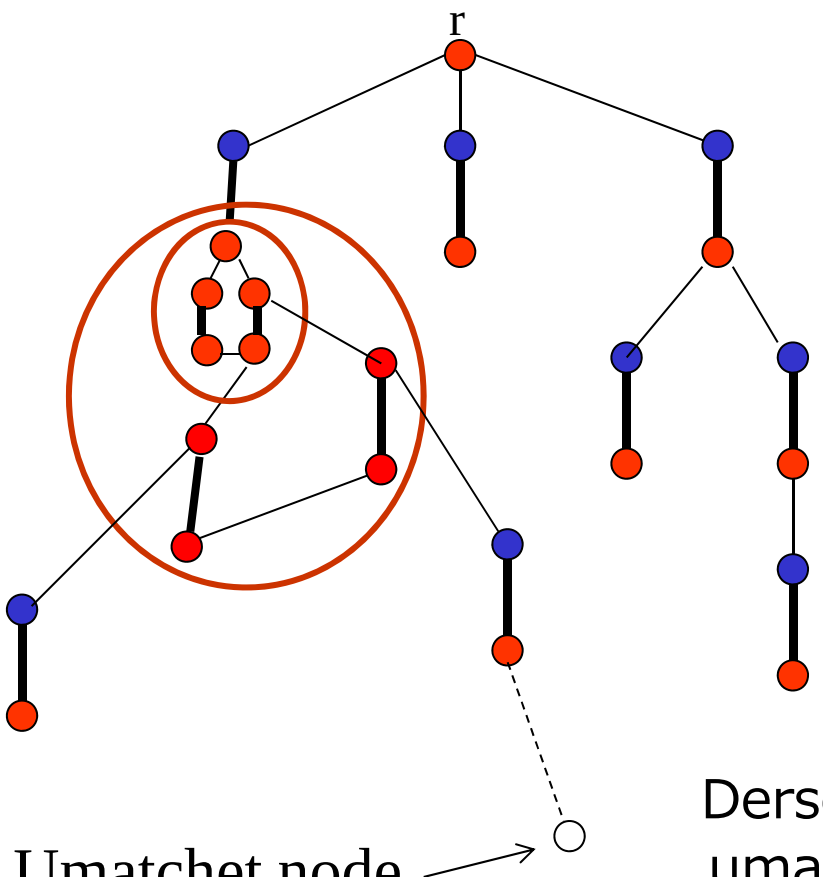
Til blå node i T: Bryr oss ikke om slike

Ny type kant: Ikke i bipartite grafer. Behandles slik.

Kant til matchet ufarget node: Farger noden blå, og den tilhørende matchede node rød. Tar disse med i treet.

Kant til ufarget og umatchet node: Vi har da funnet en forbedrings-vei. Vi ”bruker” denne, og får en større matching

Avslutningen av steget i den ”utvidede ungarske algoritme”



Umatchet node
HURRA!

Om vi finner kant fra rød node til umatchet node:

Da går vi bakover langs den alternerende veien, pakker opp de sammensnurpede nodene, og finner den alternerende veien tilbake gjennom dem.

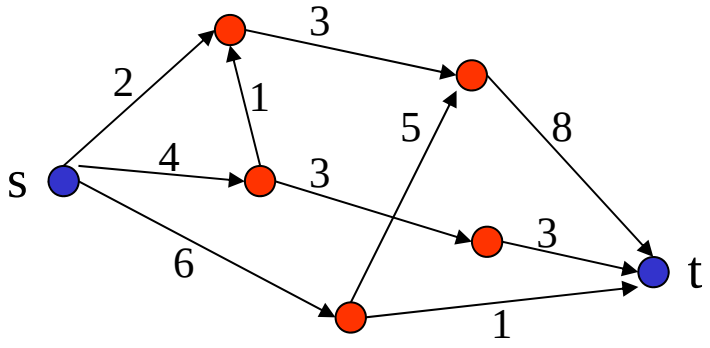
Dermed får vi en alternerende vei i den opprinnelige grafen tilbake til roten. Denne kan vi bruke til å øke matching med en kant.

Dersom vi *ikke* finner kant fra rød til umatchet (= ufarget) node:

- Da finnes *ingen større matching!*
- *Men dette er noe krunglete å vise!*

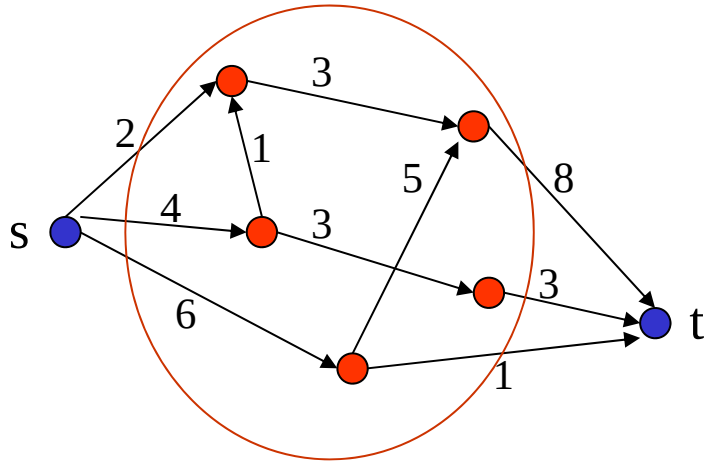
Flyt i nettverk, kap. 14.2

- Dette stoffet er også noe dekket i Weiss-boka, så man kan også lese der.
- At man her bruker ordet ”nettverk” (og ikke noe med ”grafer”) er bare ren tradisjon. Grovt sett er nettverk rettede grafer med forskjellige kapasiteter, vekter etc. på kantene (og ofte også på nodene)
- Svært mange praktiske problemer faller inn under nettverksproblemer, og spesielt flyt i nettverk:
  - Datanett med flyt av datapakker, og kapasitet (”båndbredde”)
  - Forskjellige typer rør-nettverk, der væsker flyter, og rørene har kapasitet
  - Vei-nettverk, der biler flyter, og med forskjellige kapasitet på veiene
  - …
- De nettverk vi skal studere her har
  - Kapasiteter på kantene
  - Én ”kilde-node” *s* og én ”sluk-node” *t*
  - Og oppgaven er generelt å presse så mye ”flyt” fra *s* til *t* som mulig.



Flyt i nettverk, kap. 14.2

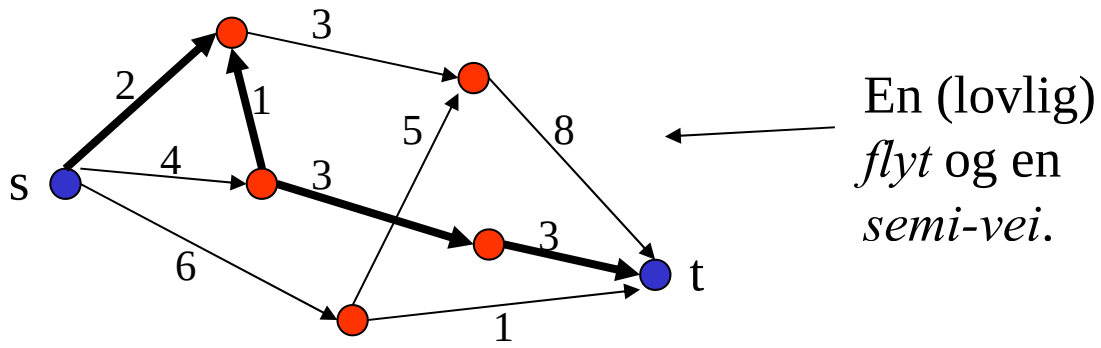
- En flyt *f* i et slikt nettverk er sammensatt av en flyt *f(e)* på hver kant *e*, som er slik at:
  - **Flytkonserverings-prinsippet:** I hver node, bortsett fra i *s* og *t*, er summen av flyt *inn* til noden lik sum av flyt *ut* av noden (definert i forhold til kantenes retning).
  - **I nettverk med kapasiteter:** Hver kant *e* har en viss kapasitet $c(e) \geq 0$, og flyten *f(e)* må da ligge mellom *0* og *c(e).*
- Forutsetter i denne fremstilling: Det går ikke kanter inn i s eller ut av t.
  - *val(f)* er summen av flyten som går ut av s.
  - Lemma: Summen av flyten som går inn i t er også *val(f)*
    - Vises grovt sett ved summering av flyten inn/ut over alle noder



Begreper brukt i boka, men som vi ikke bruker direkte i foilene

Disse detaljene er derfor ikke pensum!

- En *semi-vei* gjennom grafen = en vei fra s til t i den underliggende *urettede* grafen
- *Enhets-flyt:* Definert av en semi-vei der det er flyt lik +1 på de kantene som følger veiens retning, og lik –1 på de kantene som går *mot* veiretningen
- Lemma for nettverk uten kapasiteter: To flyter som summeres kant for kant gir en ny lovlig flyt
- Lemma for nettverk uten kapasiteter: Om hver kant-flyt multipliseres med en gitt konstant får vi en ny lovlig flyt.

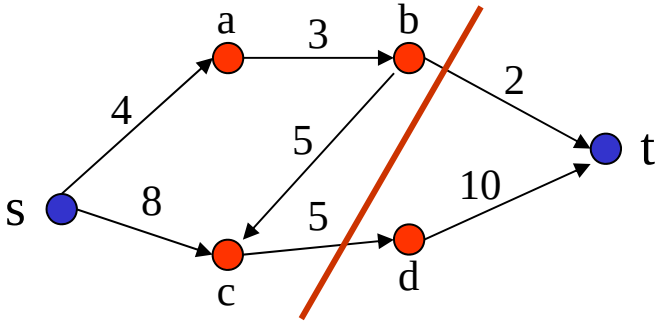


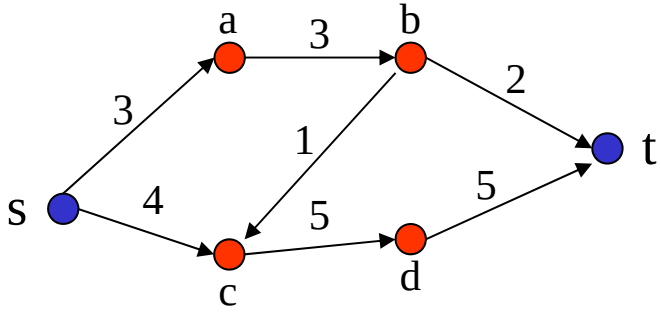
En (lovlig) *flyt* og en *semi-vei*.

Flyt i nettverk, med kapasiteter

- Hver kant *e* har en viss kapasitet *c(e),* og flyten *f(e)* gjennom kanten *e* må ligge mellom 0 og *c(e)*.
- Ønske:
  - Gitt et nettverk med kapasiteter.
  - Vi ønsker å finne kantflyter *f(e)* som
    - holder seg innenfor kapasitetene
    - utgjør en maksimal flyt, altså en som gir en så stor *val(f)* som mulig
- Eksempelet under til venstre, er et nettverk med gitte kapasiteter.
  - Vi ser intuitivt: Maksimal flyt er her 7, og en slik flyt er gitt til høyre.

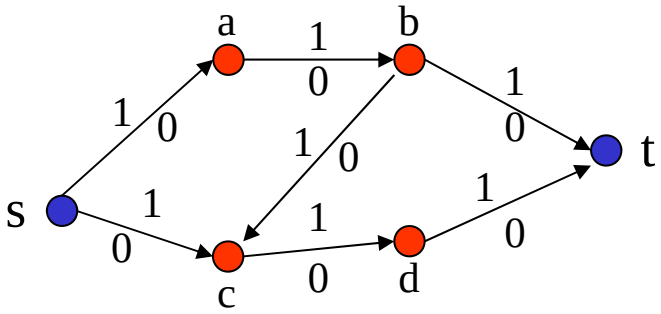
Et ”kutt” med kapasitet 7.
Mer om det siden

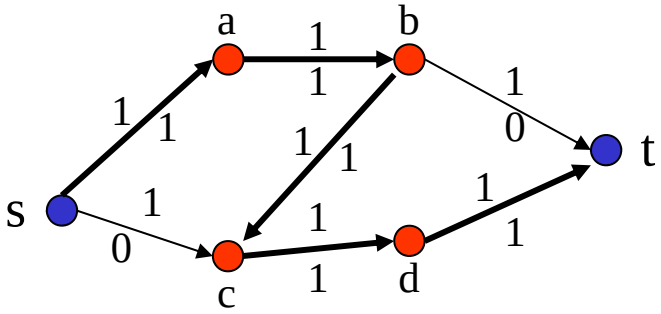




Ren grådighet virker ikke her heller

- Den naive grådighets-algoritmen (som ikke virker!):
  - Steget: Finn ”enkel flytøkningsvei”:
    - Finn en rettet vei fra s til t som er slik at alle flyter *f(e)* er lavere enn *c(e)* langs veien
    - Øk flyten langs denne så mye som mulig (gitt av den kanten som har minst *c(e) – f(e)* langs veien)
  - Gjenta dette til ingen slike veier finnes.
- På figuren under er kapasitetene angitt *over* kantene (alle 1) og flyten angitt *under* kanten (initielt er den 0 overalt).
  - Vi finner først en tilfeldig slik *enkel flytøkningsvei*, f.eks. s-a-b-c-d-t. Langs denne kan vi øke flyten med 1, og vi får neste situasjon under.
  - *val(f)* er nå 1, men det er opplagt at vi kan oppnå *val(f)*=2
  - MEN, det finnes *ingen enkel flytøkningsvei* av typen definert over som kan bringe oss til en situasjon med *val(f)=2.*

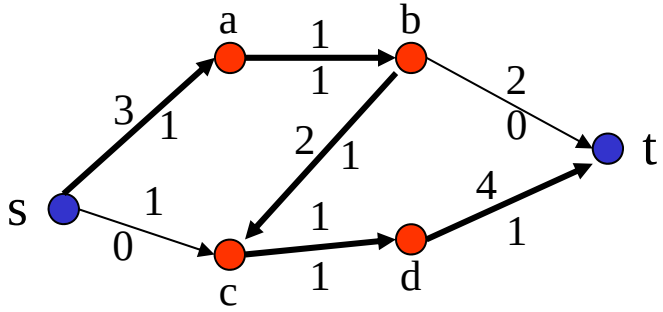




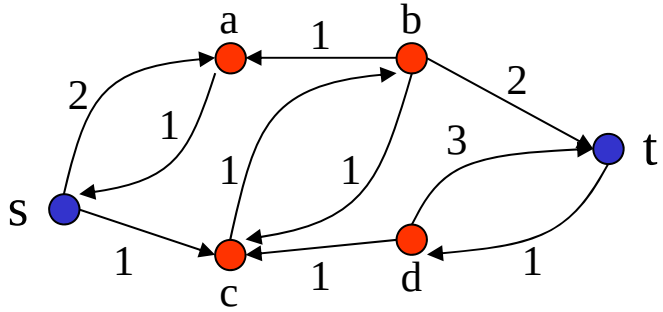
Det *f*-avledede nettverket *N(f)*

- Det vi tydeligvis ikke har tatt hensyn til i den enkle betraktningen med flytøkende veier, er at vi også kan *minke* flyten i noen kanter når vi vil gjøre en forandring – og ved å ta hensyn til det får vi faktisk en fullgod algoritme
- For å få oversikt over forandrings-mulighetene på den enkelte kant kan vi, ut fra et gitt nettverk med kapsiteter *c(e)* og en gitt lovlig flyt *f(e),* tegne det *f-avledede nettverket* betegnet N_f, N*f* eller *N(f)*. Vi bruker her *N(f)*.

  (Merk her: Nye kapasiteter i forhold til forrige foil):



Nettverk med kapasiteter (over)
og flyt (under)

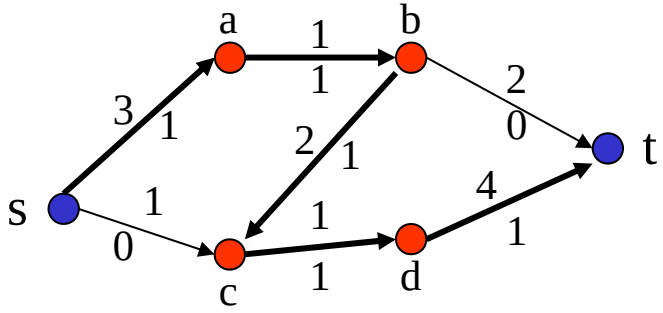


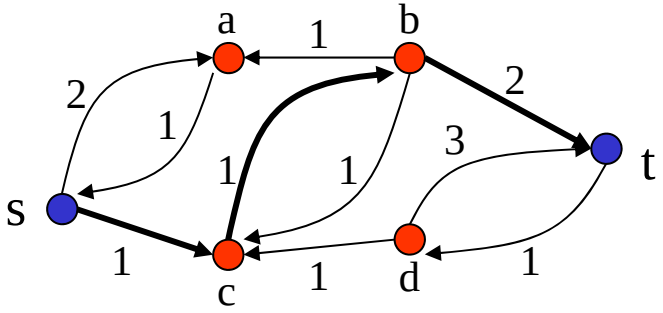
Det *f*-avledede nettverket *N(f)*
(angir mulige flytforandringer)

Se også figur 14.8 i boka (side 435)

f-forbedringsveier

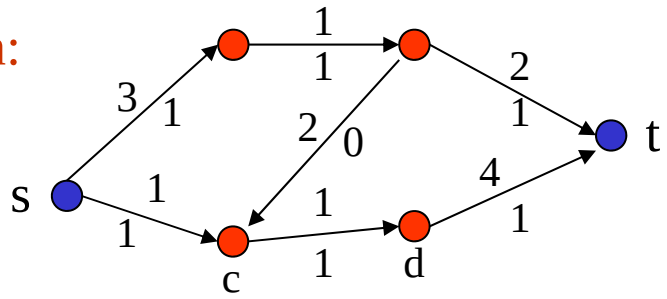
(Samme figurer som på forrige foil, det opprinnelige nettverket N til venstre:)





- Vi leter så etter veier fra *s* til *t* i det *f*-avledede nettverket *N(f)*
  - Slike veier kalles "*f*-forbedringsveier" ("*f*-augumenting semipaths")
  - Søket kan gjøres f.eks. bredde-først eller dybde-først i *N(f)* fra *s*.
  - Vi kan for eksempel velge s-c-b-t. Den maksimale flytforandringen langs denne er her 1 (anta generelt h).
- Vi gjør så den tilsvarende flytforandringen, ved å
  - øke flyten med h i de kantene i N der *f*-forbedringsveien går samme vei som i N
  - minke flyten med h der kanten i *f*-forbedringsveien går motsatt vei av i N

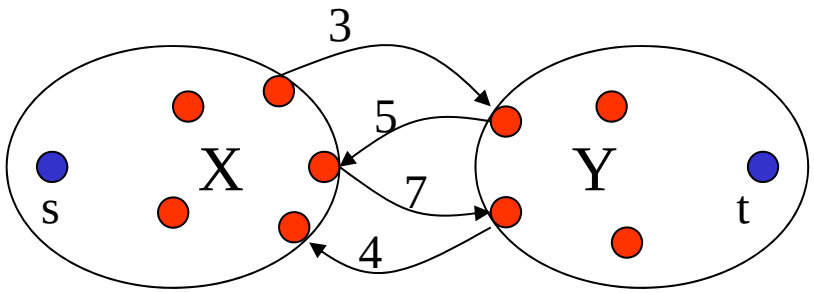
Dette gir den nye flyten:



- Ut fra denne må vi så lage et helt nytt *f*-avledet nettverk *N(f),* osv

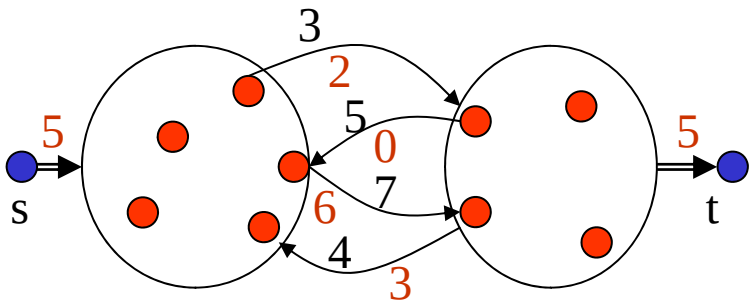
Kutt i nettverk

- Et ”kutt” (Cut) i et nettverk er rett og slett en todeling av nodemengden i mengdene X og Y. Her skal vi bare se på kutt der *s* er i X og *t* er i Y.



- *Kapasiteten* av et kutt K=X,Y (skrives *cap(K)*) er summen av kapasitetene på de kantene som går fra X til Y (og er altså *uavhengig* av kapasiteten på de som går motsatt) .
- I figuren over blir kuttets kapasitet altså 3+7=10

Mer om kutt i nettverk



- Lemma: Gitt en lovlig flyt *f* og et kutt K=X,Y. Da er *val(f)* ≤ *cap(K)*.

  Vises grovt sett slik:

  – Ved summering av flyten inn/ut over alle noder i X' = X–s finner vi at flyt inn i X' må være lik flyt ut av X'. Derved må

  (flyt ut av s) + (flyt bakover over K) = (flyt fremover over K) altså

  (flyt ut av s, altså: *val(f)*) = (flyt fremover over K) – (flyt bakover over K)

  – Høyresiden over kalles "flyten over *K*", og den er opplagt ikke større enn *cap(K)*

  – Derved vet vi: *val(f)* ≤ *cap(K)*.

  – I figuren over: *5 = 2 + 6 – 0 – 3 ≤ 3 + 7*

- Dette gir oss en mulighet til å vise at vi har en maksimal flyt:

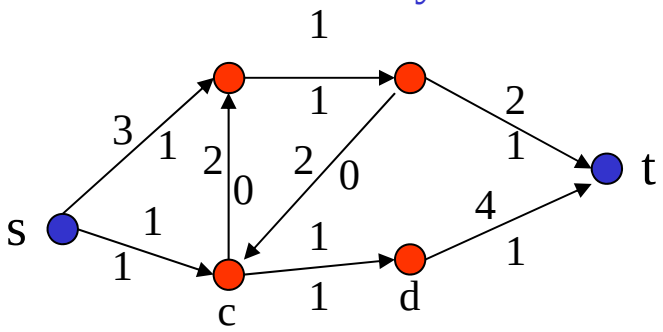
  Om vi har en flyt *f* og et kutt *K* slik at *val(f) = cap(K)*

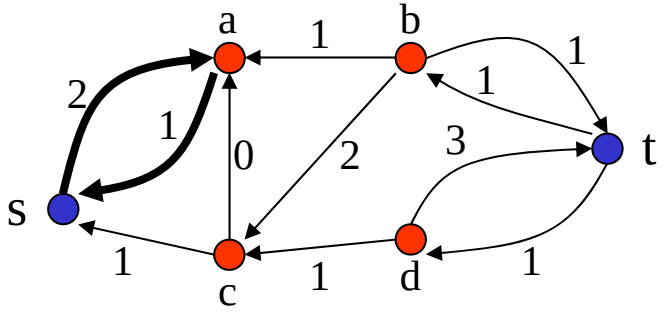
  så er *flyten maksimal*, og *intet kutt er mindre*!

FordFulkerson-algoritmen

FordFulkerson-algoritmen går slik:

- Start med null flyt
- Steget (og ved starten av dette har vi generelt en eller annen lovlig flyt *f*):
  - Lag det *f*-avledede nettverket *N(f)* (som angir alle forandringsmuligheter)
  - Finn en *f*-forbedringsvei gjennom dette nettverket, og finn maksimal økning for denne (bestemt av den kanten langs veien med minst forandringsmulighet)
  - Gjør den forandringen i flyten som denne angir
- Gjenta steget til vi ikke lenger kan finne en *f*-forbedingsvei fra s til t i *N(f)*
  - Algoritmen slutter når det ikke er *noen* rettet vei fra s til t i *N(f)*. Bevis for at vi da har en maks flyt er at vi da *kan vise et kutt med denne kapasitet* (neste foil).



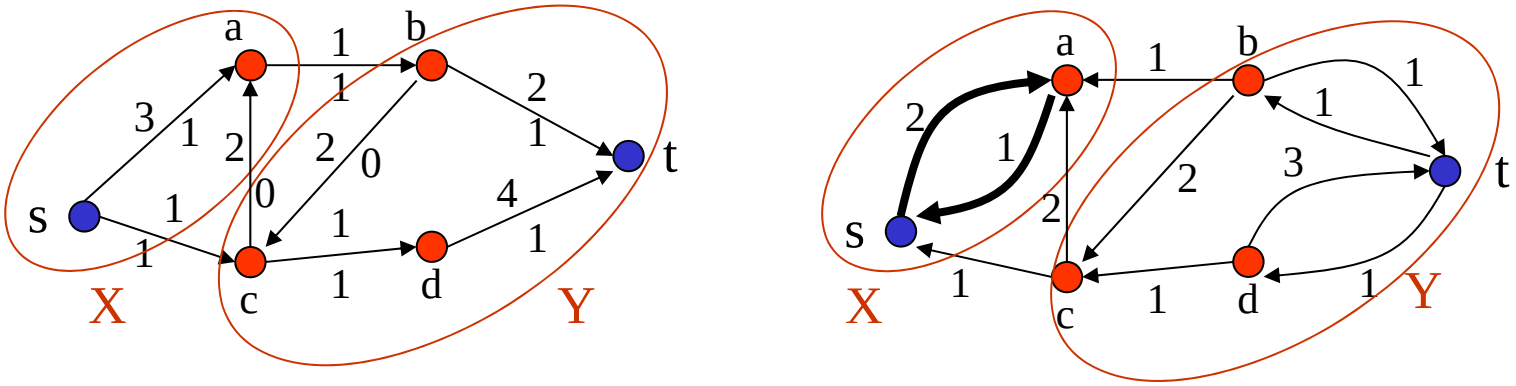


Se også programmet på side 438

Dere skal ”håndgå” denne algoritmen ut fra figur 14.9 på gruppene neste uke

Avslutning av FordFulkerson-algoritmen

Den slutter altså med at det ikke er noen forbindelse fra s til t i *N(f)*.



- For å vise at vi faktisk har en maksimal flyt ønsker vi da å finne et kutt *K* som har nøyaktig samme kapasitet som flyten *f,* altså: *cap(K)=val(f).*
- Det viser seg at et slikt er lett å finne: La X være de nodene som kan nåes i *N(f)* fra *s*, og la Y være resten av nodene.
- Noden *t* er da i Y, se figuren over.
- Siden ingen kanter i *N(f)* går fra X til Y er det lett å se at
  - Alle kanter i N som går fra X til Y bruker hele sin kapasitet (er ”mettet”)
  - Alle kanter i N som går fra Y til X har flyt *f = 0.*
- Ut fra defenisjonen av *cap(K)* ser vi da at den er lik flyten over *K = val(f)*
  - Demed vet vi at flyten er maksimal, og vi har vi bevist følgende teorem:

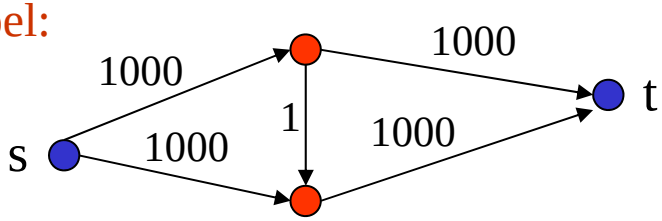
Teorem (Max-flyt min-kutt): I et nettverk med kapasiteter kan vi finne en flyt *f* og et kutt *K* slik at *val(f)=cap(K).* Da vet vi at flyten er maksimal, og at intet kutt *K* har mindre kapasitet.

Varianter av FordFulkerson-algoritmen

- **FordFulkerson-algoritmen** sier bare at man skal velge *en eller annen* forbedringsvei i forhold til kapasitetene og den nåværende flyten, deretter finne den maksimale flytøkningen vi kan gjøre langs denne, og så legge til denne flyten.

- Når vi ikke lenger kan finne noen slik forbedringsvei har vi en maksimal flyt (bevist ved at algoritmen gir oss et kutt med kapasitet = flyten)

- Om vi ikke legger på ytterligere styring for valg av forbedringsvei, gjelder:
  - Om kapasitetene er heltall, så kan antall steg bli den maksimale flyten for nettverket. Eksempel:



n = antall noder
m = antall kanter

  - Om kapasitetene er reelle tall kan algoritmen teoretisk sett gå i evig løkke (!?)

- **Forbedring 1:** Man kan hele tiden velge den forbedringsveien som gir størst forbedring (kan lett finnes med en algoritme analog til en korteste-vei-algoritme)
  - Dette gir worst-case-tid: $O(m \log(n) \log(\text{maks-flyt}))$

- **Forbedring 2:** (Edmonds og Karp) Man kan også hele tiden velge den veien som er kortest i antall kanter (kan finnes ved bredde først søk)
  - Dette gir worst-case-tid: $O(n m^2)$ (altså *uavhengig* av maksimal flyt!)

Varianter av problemet med maksimal flyt

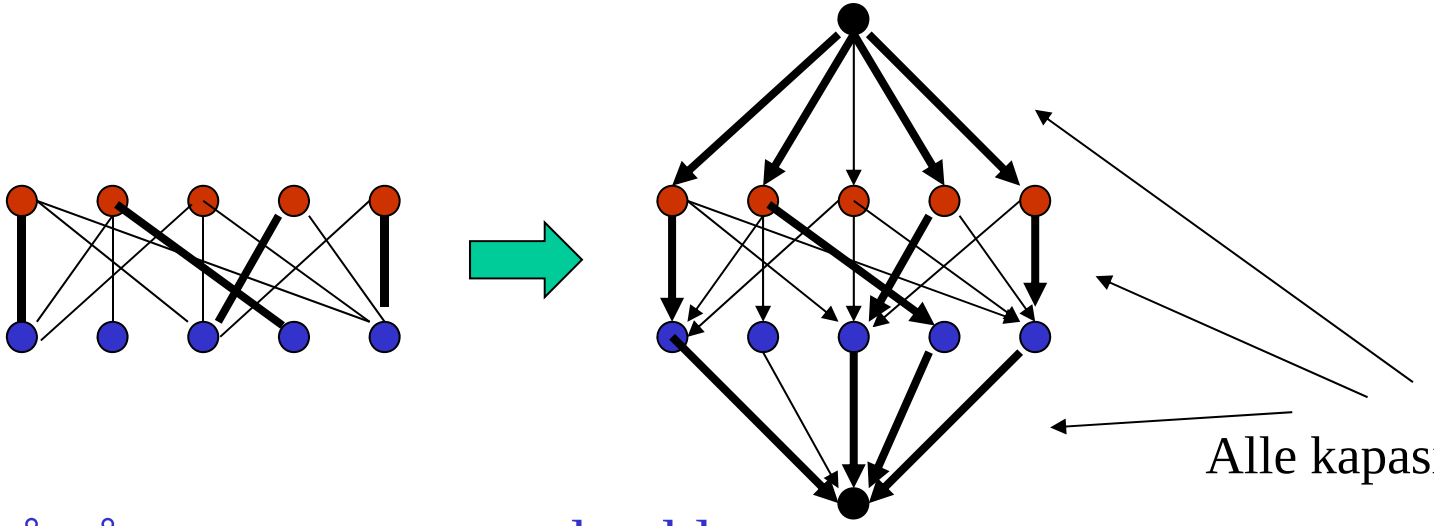
- For det første finnes alternativer til FordFulkerson
  - *Dinac* har designet en algoritme
  - *Goldberg and Tarjan* (preflow push algorithm)
- Vi kan også ha angitt også en *minimal flyt* på hver kant.
  - Da er det et eget problem bare å *finne en mulig flyt*
  - Men etter det kan man fortsette som for FordFulkerson
- Man kan ha en *pris* på hver kant, for å sende flyt på denne kanten.
  - Her finnes en kjent optimaliserings-algoritme: Out-of-kilter-algoritmen
- Man kan ha flere kilder og flere sluk, med forskjellige krav til flyten inn og ut av disse.
- Og det kan være flere forskjellige ting (”commodities”) som skal flyte (busser, personbiler, …) og kantene kan ha forskjellige kapasitet for hver av disse (eventuelt være sperret for noen av dem)
  - Dette er et aktivt forskningsområde, for trafikkplanlegging, ruting i kommunikasjonsnettverk etc.

Kap. 14.2.7: En sammenheng mellom flyt i grafer og matchinger i bipartite grafer

Enkelt men viktig lemma:

1. Ved heltallige kapasiteter kan man alltid finne en maksimal flyt som er heltallig.
2. Når alle kapasitetene er 1 kan vi altså finne en maksimal flyt der hver kant har flyt 0 eller 1 (og FordFulkerson vil alltid finne en slik!)

En slik flyt kan dermed tolkes som et *utplukk av kanter* (de som har flyt)



Se på på gruppene neste uke, bl.a.:

- Leting etter forbedringsvei i flytnettverket tilsvarer helt leting etter forbedringsvei ut fra en matching i grafen
- Et utplukk av noder som dekker alle kanter tilsvarer et kutt i flytnettverket