Svarforslag til ukeoppgaver til INF 4130
15. november 2011

Oppgave 1: Løs 14.4 (hvori innbakt svaret på oppgave 14.5)

Vi skal altså vise at Hungarian-algoritmen kan implementeres i tid $O(n^3)$, der n er antall noder i X, som er lik antall noder i Y.

Om vi bare tenker på algoritmen (uten å studere koden på side 422/423) så består den altså av en ytre løkke som går rundt hver gang vi finner en forbedringsvei, og dermed får øket antall kanter i matchingen. Denne løkka kan selvfølgelig ikke gå flere enn n (og dermed $O(n)$) ganger.

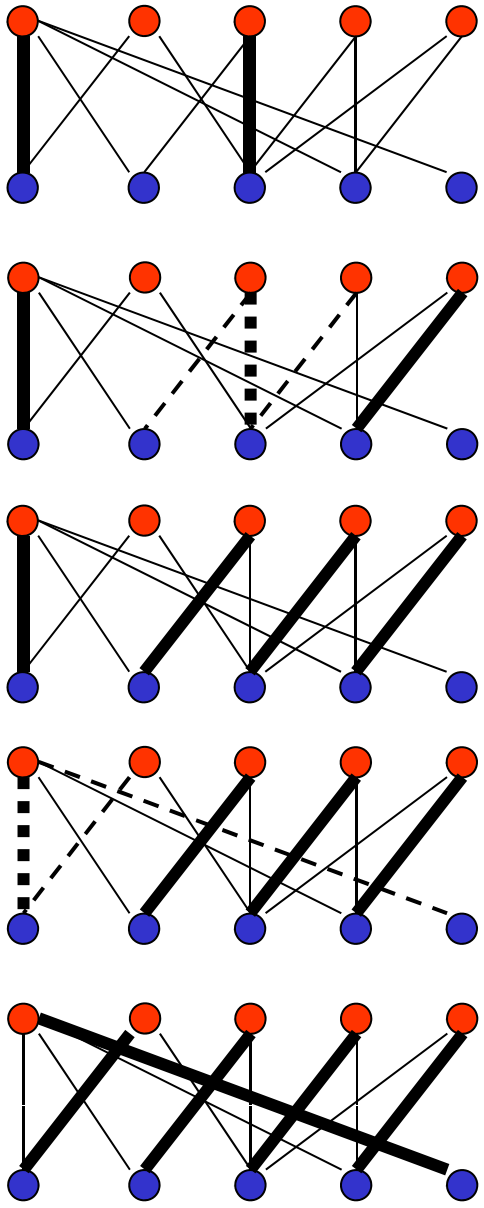
Den indre løkka vil bestå i å bygge opp treet kant for kant (eller det blir snarere bygget med to og to kanter av gangen, en umatchet og en matchet, om vi da ikke finner en forbedringsvei). Rekkefølgen vi legger (par av) kanter til i treet er vilkårlig, og det viktige er at vi kan få lagt til et nytt par i tid $O(1)$. Det går greit om vi (1) har et merke i nodene som angir om de er med i treet eller ikke, og (2) om nodene har en peker som er "null" om noden ikke er med i en matching, ellers peker den til den noden den er matchet med. Dessuten må vi kunne holde mengder av noder, der det tar tid $O(1)$ å sette noder inn og å få en (tilfeldig) node ut (f.eks. ved at nodene har en neste-peker og holdes i en liste).

En slik mengde R brukes til å holde de nodene man har sett at skal være med i treet, men som man ennå ikke har fulgt kantene ut fra. Denne mengden inneholder fra starten bare den umatchede noden fra X som man velger å starte fra. Steget blir å ta en node ut av R, og følge alle kanter fra denne. Hver slik kant kan enten

- gå til en node som allerede er i treet. Da gjør ingenting
- gå til en node som er matchet men ikke er i treet. Da utvider vi treet med et kant-par, og legger den nye bladnoden inn i R (altså *ikke* den som lå på midten av paret).
- gå til en umatchet node som ikke er i treet. Da har man funnet en forbedringsvei (og trebyggingen avsluttes)

Alle disse operasjonene kan nå gjøres i tid $O(1)$. Siden det ikke er flere enn n^2 kanter og man aldri ser på kanter mer enn to ganger (fra den ene og den andre endenoden), kan det å finne en mulig forbedringsvei ikke ta mer tid enn $O(n^2)$. (Det viser seg, om man ser på saken i detalj, at man faktisk bare ser på hver kant én gang, men det er ikke viktig her). Dermed har vi vist at hele algoritmen kan utføres i tid $O(n^3)$.

Oppgave2. Løs oppgave 14.6



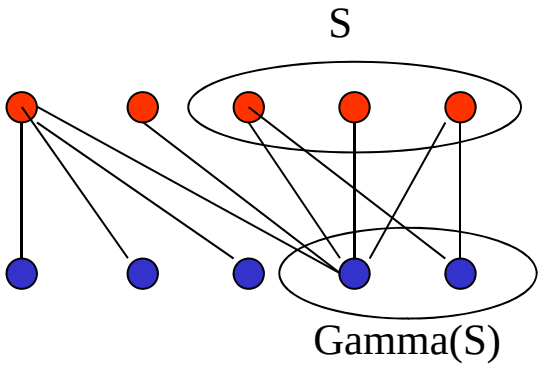
Vi starter altså med grafen angitt i oppgaven, som er vist øverst på figurene til høyre Vi nummererer nodene fra venstre x1 – x5 og y1 – y5. Vi kan tenke oss at vi så fortsetter med å lage et tre fra x5. Dette gir umiddelbart en forbedringsvei (med én kant) om vi ser på x5-y4. Merk altså at en kant med umatchede noder i begge ender er en (enklest mulig) forbedringsvei.

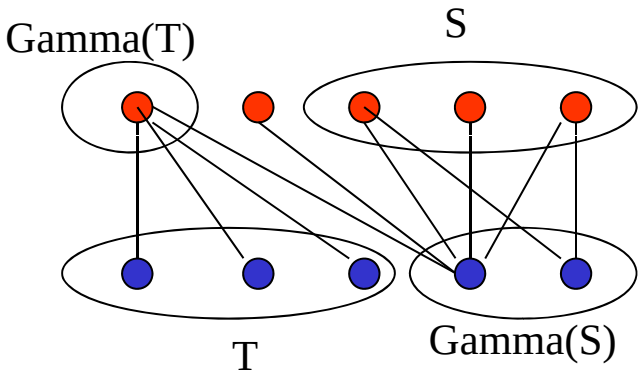
Etter at denne forbedringen er gjort, kan vi så f.eks. bygge et tre fra x4, og en mulighet er da at men finner forbedringsveien x4-y3-x3-y2 (stiplet i andre figur).

Om vi ”bruker” denne forbedringsveien får vi tredje figur, og om vi her bygger et tre fra x2, finner vi før eller siden forbedringsveien som er stiplet i fjerde figur. Bruker man den får man den perfekte matchingen angitt nederst.

Oppgave 3

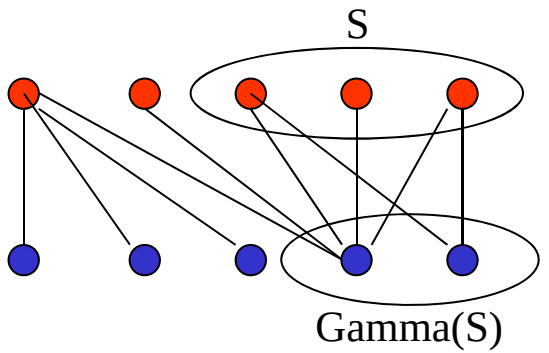
Dette er faktisk veldig lett å vise. Anta at vi har et utplukk *S* av *X* slik at *Gamma(S)* har færre noder enn *S*, som på venstre figur under. Vi kan da konstatere, ut fra definisjonen av *Gamma(S)*, at det ikke går kanter mellom *S* og *T = (Y - Gamma(S))*. Dermed må *T* være slik at *Gamma(T)* er inneholdt i *X – S*, og derved ha færre noder enn *T*.



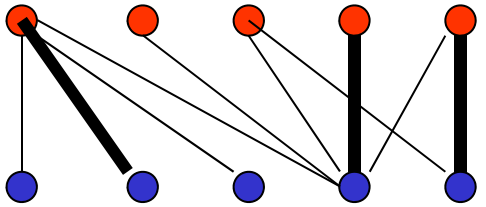


Oppgave 4

La oss først begrunne at et nodeutplukk som dekker alle kanter ikke kan ha færre noder enn en matching i grafen kan ha kanter. Det kan man se ved at hver kant i en matching må ”bruke opp” minst én node i utplukket og kanskje to, nemlig den/de som ”dekker” denne matchingkanten. Ingen andre kanter i matchingen kan berøre disse nodene.



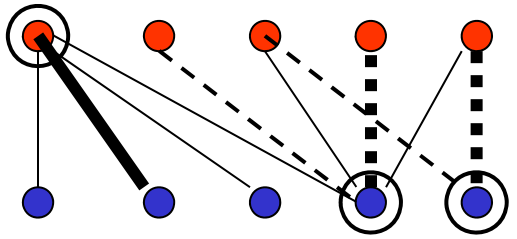
Vi kan bruke den øverste grafen til høyre som et eksempel på hvordan et slikt utplukk kan finnes, siden den opplagt ikke har noen komplett matching. Vi benevner nodene hhv. x1 – x5 og y1 – y5. Den utvidede ungarske algoritme bygger raskt opp en matching f.eks. som angitt i midterste figur (det kunne også den opprinnelige ungarske algoritmen).



I neste skritt ville så den utvidede ungarske bygge opp to trær med x2 og x3 som røtter. Om vi bruker disse i den rekkefølgen får vi de to trærne angitt ved de stiplede kantene i nederste figur. Hvert tre får bare to kanter, og så stopper det uten at noen forbedringvei er funnet. Matching-kanten x1-y2, samt de umatchede nodene y1 og y3 er ikke med i noe tre.

En overdekning kan nå velges slik:

- Ta med alle Y-noder som er *med* i et tre. Merk at alle disse er matchet.
- Ta med alle X-noder *utenfor* trærne. Merk at alle disse er matchet, ellers ville vi startet et nytt tre.



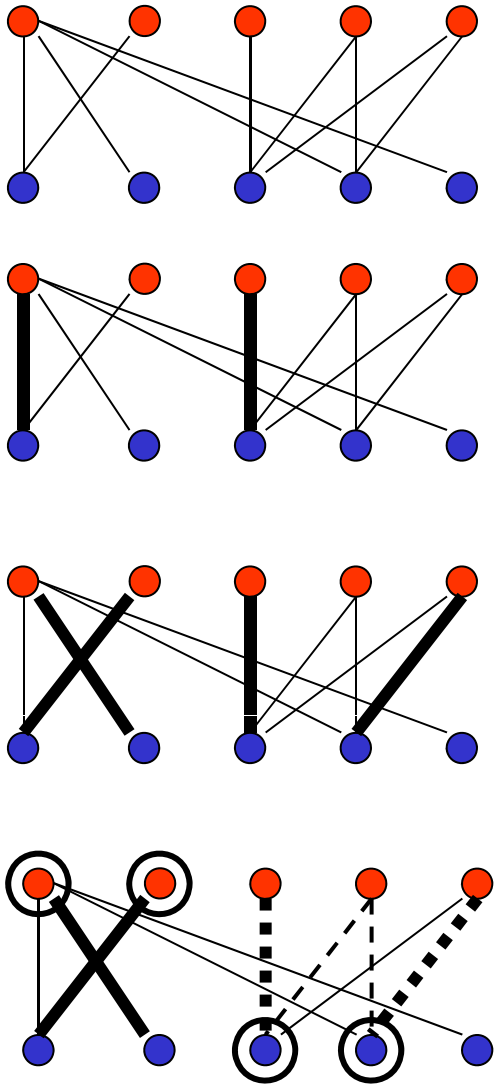
Denne mengden er merket med ringer rundt nodene. Dette utplukket blir for det første like stort som matchingen, siden det er plukket nøyaktig én node fra hver matching-kant (siden en matchet kant enten har begge ende-noder innenfor eller begge utenfor et tre). Videre vil disse nodene dekke alle kanter, siden det ikke kan gå noen kant fra en X-node i et tre til en Y-node utenfor trærne (da ville trebyggingen fortsatt), og selvfølgelig ingen mellom to X-noder eller to Y-noder. Alle andre typer kanter ser man lett må være dekket av den angitte nodemengden.

Ekstraoppgave 4.1

Om vi fjerner x2-y3 og x3-y2 fra grafen i oppgave 14.6, så får vi grafen angitt øverst til høyre. Vi kan så i rekkefølge bygge trær fra x1 og x3, og finner da enkle forbedringsveier som gir oss neste graf.

Vi kan så bygge tre fra x2 og finne forbedringsveien x2-y1-x1-y2 og så bygge tre fra x5 og få forbedringsveien x5-y4. Dette gir tredje graf. Vi bygger så et tre fra den siste umatchede X-noden x4, og får et tre som er stiplet i nederste graf.

Vi velger så en nodemengde slik som angitt i svaret på forrige oppgave, og dette er angitt av sirklene på nodene. Vi ser at den dekker alle kanter, og har størrelse 4, altså lik størrelsen av matchingen.



Ekstraoppgave 4.2

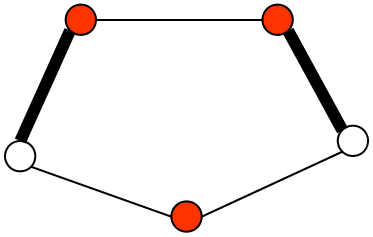
For generelle grafer kan vi altså finne en maksimal matching i polynomisk tid. Dersom det var slik at størrelsen på en maksimal matching alltid var lik størrelsen på et minimalt kantdekkende nodeutplukk også i generelle grafer, så kunne vi jo svare på følgende NP-komplette spørsmål i polynomisk tid:

Gitt: En generell graf og et heltall K.

Spørsmål: Finnes det et kantdekkende nodeutplukk for denne grafen med færre enn eller lik K noder"

Vi kunne jo da besvare denne oppgaven ved rett og slett å finne den maksimale matchingen, og se om størrelsen av den er mindre eller lik K, og dette kan gjøres i polynomisk tid. Men om P=/=NP så kan ikke dette løses i polynomisk tid. Altså må antakelsen om den angitte likhet være gal.

Å finne en graf der en maksimal matching og et minimalt kantdekkende nodeutplukk ikke er like store er veldig lett. Det er bare å se på en odde løkke, f.eks. den under. Her kan vi se at den største matchingen har to kanter (f.eks. de tykke) mens det minste kantdekkende nodeutplukket er på tre noder (f.eks. de røde).



Oppgave 5 og 6

Oppgave 5

Det å snakke om algoritmen, og følge hva som foregår i forhold til algoritmen overlates til gruppa. Listen av kjente trykkfeil i eldste utgave av boka er som følger (men mye (alt?) er rettet i siste utgave):

- Step 1: Kanten 4-7 in N*f* skal være stiplet.
- Step 2-7: Kanten 4-7 skal snus i alle N*f*-ene.
- Step 2: I flyt-grafen skal de to indre kantene fjernes.
- Step 2: Kanten 0-3 i N*f* skal ikke være stiplet.
- Step 7: Noden 5 i N skal ha en dobbeltring omkring seg, og det skal legges til en kant fra node 2 til node 5, med flyt 1.

Oppgave 6

Denne ble delvis gått gjennom på forelesningen, og overlates til studier på gruppeøvelsen. Spørsmålene 6.3 til 6.6 tar det antakeligvis for lang tid å gå gjennom på gruppa, men de skulle være greie å gå gjennom på egenhånd.