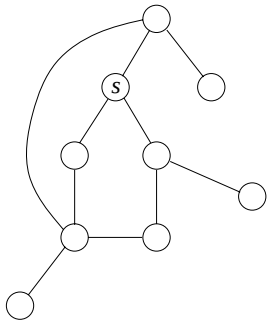
**Algoritmer (datastrukturer) och komplexitet, våren 2010**

Uppgifter till övning 3

# Grafalgoritmer

**Grafsökning** Gör en DFS- och en BFS-genomgång av följande graf. Starta i hörnet $s$ och numrera hörnen i den ordning dom besöks.




---

**Topologisk sortering av DAG** En DAG är en riktad acyklisk graf. En topologisk sortering av en sån graf är en numrering av hörnen så att alla kanter går från ett hörn med lägre nummer till ett hörn med högre nummer.

Modifiera den vanliga djupetförstsökningsalgoritmen så att den konstruerar en topologisk sortering av grafen i linjär tid.

---

**Klick** En *klick* (eng. clique) i en oriktad graf är en mängd hörn där varje par av hörn har en kant mellan sig. Ett viktigt problem i datalogin är problemet att hitta den största klicken i en graf.

Ett annat mycket omtalat problem är problemet att hitta den största *oberoende mängden* i en graf. En oberoende mängd (eng. independent set) är en mängd hörn som inte har någon kant alls mellan sig.

Dessa båda problem är så lika varandra att om man har en algoritm som givet en graf hittar den största klicken så kan man utnyttja den algoritmen i en för övrigt mycket enkel algoritm som givet en graf hittar den största oberoende mängden. Visa detta!

---

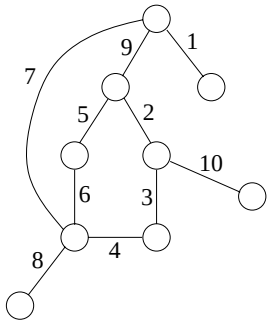
**Kantmatrisprodukt** På föreläsningen beskrevs hur en riktad graf $G = \langle V, E \rangle$ representeras som en kantmatris $B$ (eng. incidence matrix) av storlek $|V| \times |E|$. Beskriv vad elementen i matrisen $BB^T$ representerar (där $B^T$ är $B$ transponerad).

---

**Bipartithet** Beskriv och analysera en algoritm som avgör om en graf är bipartit. Tidskomplexiteten för algoritmen ska vara linjär i antalet kanter och antalet hörn i grafen.

---

**Spännande träd** Visa hur det minsta spännande trädet hittas i följande graf med både Prims och Kruskals algoritmer.




---

**Förändrat flöde**

a) Beskriv en effektiv algoritm som hittar ett nytt maximalt flöde om kapaciteten längs en viss kant *ökar* med en enhet.

   Algoritmens tidskomplexitet ska vara linjär, alltså $O(|V| + |E|)$.

b) Beskriv en effektiv algoritm som hittar ett nytt maximalt flöde om kapaciteten längs en viss kant *minskar* med en enhet.

   Algoritmens tidskomplexitet ska vara linjär, alltså $O(|V| + |E|)$.

---

**Eulercykel** Givet en oriktad sammanhängande graf $G = \langle V, E \rangle$ där alla hörn har jämnt gradtal. Hitta en Eulercykel, dvs en sluten stig som passerar varje kant i $E$ exakt en gång. Algoritmen ska gå i linjär tid.

---

**Julklappsfördelning** En pappa ska ge sina $n$ barn var sin julklapp. Varje barn har skrivit en önskelista. Pappan vill ge varje barn en julklapp från barnets önskelista, men han vill inte ge samma julklapp till flera barn.

Konstruera och analysera en effektiv algoritm för detta problem. Du kan anta att det finns högst $m$ saker på varje önskelista.

---

# Lösningar

**Lösning till Grafsökning** – egen övning!

---

**Lösning till Topologisk sortering av DAG**
Om man tittar på i vilken ordning djupetförstsökningen passerar hörnen på tillbakavägen i grafen (det vill säga i vilken ordning hörnen färdigbehandlas av sökningsproceduren) så ser man att det är precis omvänd topologisk ordning. Om vi vill sortera hörnen topologiskt behöver vi alltså bara lägga till en sats `Push(u)` sist i proceduren `DFSVISIT(u)`. När djupetförstsökningen är genomförd kan vi poppa hörnen ett i taget från stacken, och då kommer dom topologiskt sorterade. □

---

**Lösning till Klick**
Ledning: studera komplementgrafen (som har kanter bara där ursprungsgrafen inte har kanter). □

---

**Lösning till Kantmatrisprodukt**
Diagonalelementet $(i, i)$ anger hur många kanter som har sin ändpunkt i hörn $i$. Ickediagonalelementet $(i, j)$ är det negerade antalet kanter som går mellan hörnen $i$ och $j$. □

---

**Lösning till Bipartithet**
Använd djupetförstsökning men färga hörnen alternerande med rött och grönt. Om en kant mellan två hörn av samma färg upptäcks är grafen inte bipartit. □

---

**Lösning till Spännande träd** – egen övning!

---

**Lösning till Förändrat flöde**

a) Anta att kanten från $u$ till $v$ får sin kapacitet ökad med ett. Utgå från det tidigare maximala flödet $\Phi$ och gör bara en ny iteration i Ford-Fulkersons algoritm med den ändrade grafen: I restflödesgrafen ökas kapaciteten i kanten $(u, v)$ med ett. Gör en grafsökning (i tid $O(|V| + |E|)$) för att se om det nu finns någon stig i restflödesgrafen längs vilken flödet kan öka. Om det finns det måste det vara ett flöde av storleken 1 (eftersom alla flöden är heltalsflöden). Om det inte finns något flöde i restflödesgrafen är $\Phi$ fortfarande det maximala flödet.

b) Anta att kanten från $u$ till $v$ får sin kapacitet minskad med ett. Om det tidigare maximala flödet $\Phi$ inte utnyttjade hela kapaciteten i $(u, v)$ så förändras inte flödet alls. I annat fall måste vi uppdatera flödet på följande sätt:

   Eftersom det kommer in en enhet större flöde till $u$ än som går ut och det kommer in en enhet mindre flöde till $v$ än det går ut så måste vi försöka leda en enhets flöde från $u$ till $v$ någon annan väg. Sök alltså i restflödesgrafen efter en stig från $u$ till $v$ längs vilken flödet kan öka med ett. Detta görs med en grafsökning i tid $O(|V| + |E|)$. Om det finns en sån stig uppdaterar vi $\Phi$ med det flödet.

   Om det inte finns en sån stig måste vi minska flödet från $s$ till $u$ och från $v$ till $t$ med en enhet. Det gör vi genom att hitta en stig från $u$ till $s$ i restflödesgrafen längs vilken flödet kan öka med ett och en stig från $t$ till $v$ i restflödesgrafen längs vilken flödet kan öka med ett. (Det måste finnas såna stigar eftersom vi hade ett flöde från $s$ till $t$ via $(u, v)$.) Uppdatera sedan $\Phi$ med dessa två flöden.

□

---

**Lösning till Eulercykel**
Idén är att börja i ett hörn $v$ och söka sig igenom grafen tills $v$ nås igen. Sökstigen $P$ kommer då antingen att ha besökt alla kanter eller så återstår det, om vi tar bort $P$, en eller flera sammanhängande komponenter $G_1, G_2, \ldots, G_n$. I det senare fallet kan vi göra om samma sorts sökning för varje komponent så vi får en Eulercykel för varje $G_i$, och sedan sätter vi in alla dessa Eulercykler i $P$ så att vi får en total Eulercykel.

För att kunna genomföra detta i linjär tid så hittar vi på en algoritm som använder två spelare $P_1$ och $P_2$ som tillsammans går igenom grafen. Båda spelarna startar i $v$. Spelare $P_1$ skapar stigen $P$ genom att gå längs kanter hur som helst. Han märker varje kant han passerar och stannar när han kommer tillbaka till $v$ igen. Sedan följer $P_2$ efter längs $P$, men varje gång han kommer till ett nytt hörn $u$ så kollar han om alla kanter från $u$ är märkta. I så fall fortsätter han längs $P$. Om det finns omärkta kanter (det finns alltid ett jämnt antal) så skickar han ut $P_1$ för att hitta en ny stig som börjar och slutar i $u$. $P_2$ går sedan denna nya stig innan han fortsätter från $u$. Efter att ha upprepat detta kommer $P_2$ till slut att ha gått längs varje kant exakt en gång och därför skapat en Eulercykel. $P_1$ har inte heller gått längs samma kant mer än en gång så totala körtiden blir linjär.

I algoritmen nedan kallas $P_1$ för *PathFinder* och $P_2$ för *Straggler*.

```
EulerCycle(G) =
    cycle ← {1}
    choose edge (1, t)
    mark (1, t)
    path ← PathFinder(G, 1, t)
    Straggler(path)
    return cycle

PathFinder(G, start, cur) =
    append(cur, path)
    while cur ≠ start do
        choose unmarked edge (cur, v)
        mark (cur, v)
        append(v, path)
        cur ← v
    return path

Straggler(path) =
    while path ≠ ∅ do
        u ← next(path)
        append(u, cycle)
        for all edges (u, v) do
            if unmarked (u, v) then
                mark (u, v)
                p ← PathFinder(G, u, v)
                Straggler(p)
```

□

---

**Lösning till Julklappsfördelning**
Problemet är ett bipartit matchningsproblem. Det gäller att hitta en matchning i en graf där vänsterhörnen motsvaras av barn ($n$ stycken hörn) och högerhörnen av saker. Det går en kant mellan ett barn och en sak om saken står med på barnets önskelista. Bipartit matchning kan lösas som ett flödesproblem i en graf med $O(nm)$ kanter. Eftersom flödet ökar med ett i varje varv i Ford-Fulkersons algoritm och matchningen (flödet) är högst $n$ så går slingan högst $n$ varv och komplexiteten blir $O(n^2m)$. □

---