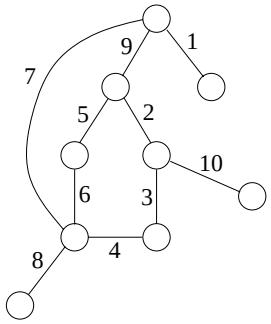
Algoritmer, datastrukturer och komplexitet, hösten 2013

Uppgifter till övning 5

Grafalgoritmer och undre gränser

Spännande träd Visa hur ett minimalt spännande träd 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.

Undre gräns för sökning i sorterad array Binärsökning i en sorterad array med n tal tar som bekant tiden $O(\log n)$. Bevisa att $\Omega(\log n)$ också är en undre gräns för antalet jämförelser som krävs för detta problem (i värsta fallet).

Laga julgransbelysning Jonas julgransbelysning med n seriellt kopplade lampor lyser inte. Det räcker att en lampa är trasig för att belysningen inte ska lysa. Han har skaffat n nya lampor som säkert fungerar. Han vill inte ersätta några hela lampor med nya, för det är dumt att slösa på nya lampor. Han vill naturligtvis göra så få lampprovningar som möjligt.

Konstruera och analysera en effektiv algoritm för hur Jonas ska få belysningen att fungera med så få lampprovningar (ur- och iskruvningar) som möjligt. Analysera värstafalletkomplexiteten för algoritmen exakt (inte bara ordoklass). En urskruvning följt av en iskruvning av en lampa tar tiden 1. Ju snabbare algoritm desto bättre.

Ge också en undre gräns för värstafalletkomplexiteten som matchar din algoritm.

Lösningar och ledningar

Lösning till Spännande träd – egen övning!

Lösning till Förändrat flöde

a) Anta att kanten från u till v får sin kapacitet ökad med ett. Eftersom tidskomplexiteten ska vara linjär kan vi inte beräkna en helt ny lösning utan måste utnyttja lösningen på det tidigare flödesproblemet. Låt Φ vara denna lösning (Vad består Φ av?) 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 Φ fortfarande det maximala flödet.

b) Anta att kanten från u till v får sin kapacitet minskad med ett. Om det tidigare maximala flödet Φ 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 Φ 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 Φ 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 $O(n^2m)$. □

Ledning till Undre gräns för sökning i sorterad array

Konstruera ett beslutsträd för en godtycklig algoritm som söker i en sorterad array. Undersök hur många löv trädet måste ha och kom fram till hur högt det då måste vara.

Lösning till Laga julgransbelysning
Numrera julgransbelysningens lampplatser från 1 till n.

byt ut alla lampor utom den på plats 1 mot nya
lägg dom gamla lamporna i en säck
$i \leftarrow 1$
while (lampor kvar i säcken) **do**
    if (belysningen lyser) **then**
        $i \leftarrow i + 1$
        skruva ur den nya lampan på plats i
    else
        skruva ur den gamla trasiga lampan på plats i
    ta en lampa från säcken och skruva i den på plats i
if not (belysningen lyser) **then**
    skruva ur den gamla trasiga lampan på plats i
    skruva i en ny lampa på plats i

Det är lätt att se att algoritmen är korrekt med hjälp av följande invariant för slingan (invarianten är sann i början av varje varv i slingan): Alla lampor på plats $< i$ är gamla och fungerar, lampan på plats i är gammal, alla lampor på plats $> i$ är nya.

I värsta fall är alla lampor trasiga i belysningen, och då gör algoritmen $(n-1)+(n-1)+1 = 2n-1$ lampbyten.

Att $2n-1$ också är en undre gräns kan man inse så här: Enda sättet att avgöra ifall en viss lampa i belysningen är hel eller trasig är att försäkra sig om att alla andra lampor är hela och då se om belysningen lyser. En tuff motståndare ser till att belysningen fortfarande inte lyser efter att $n-1$ nya lampor skruvats i helt enkelt genom att låta den enda återstående originallampan vara trasig. Alla $n-1$ bortskruvade originallampor måste sedan provas igen, för den tuffa motståndaren har ännu inte tillåtit att någon som helst information om dessa lampor har getts, och det kräver $n-1$ lampbyten till. Slutligen ser motståndaren till att just den sist provade originallampan är trasig, och då krävs det ytterligare ett lampbyte för att fixa den.

Den tuffa motståndaren behöver alltså bara två trasiga lampor för att få algoritmen att ta $2n-1$ lampbyten! □