Algoritmer (datastrukturer) och komplexitet, våren 2010

Uppgifter till övning 7

Reduktioner

Övningens ena timme ägnas åt genomgång av mästarprov 1.

Reduktion som ger negativt resultat På övning 4 beskrevs en algoritm som hittar en approximativ lösning till lådpackningsproblemet. Algoritmen fungerar så att den placerar varje pryl i den första låda som rymmer den. Uppgiften på övning 4 var att implementera algoritmen i tid $O(n \log n)$. Visa att $\Omega(n \log n)$ är en undre gräns för algoritmens tidskomplexitet.

Reduktion som ger positivt resultat Ett ofta användbart sätt att lösa problem på är att hitta en reduktion till ett problem som man redan vet hur man ska lösa. Du ska nu använda denna metod för att lösa kantsammanhängandegradsproblemet som definieras på följande sätt.

> INMATNING: En sammanhängande oriktad graf $G = (V, E)$ och ett positivt heltal K mellan 1 och $|V|$.
>
> PROBLEM: Är det tillräckligt att ta bort K kanter från grafen G för att göra den osammanhängande (dvs uppdelad i flera sammanhängande komponenter)?

Reduktioner mellan besluts-, optimerings- och konstruktionsproblem Anta att algoritmen GraphColouring(G,k) på tid $T(n)$ (där n är antalet hörn i G) svarar 1 omm hörnen i G kan färgas med k färger utan att någon kant har likfärgade ändpunkter.

a) Konstruera en algoritm som givet en graf G med n hörn bestämmer det minimala antalet färger som behövs för att färga G. Tiden ska vara $O(\log n \cdot T(n))$.

b) Konstruera en algoritm som givet en graf G med n hörn färgar hörnen med minimalt antal färger i tid $O(P(n)T(n))$ där $P(n)$ är ett polynom.

Lösningar

Lösning till Reduktion som ger negativt resultat
Som vanligt för undre gränser som är $\Omega(n \log n)$ så konstruerar vi en reduktion av problemet att sortera n tal till det aktuella problemet. Vi vet att det är omöjligt att med hjälp av jämförelser sortera n tal snabbare än $\Omega(n \log n)$. Detta gäller även om dom n talen är heltalen 1 till n permuterade, och även om vi tillåter oss att göra en inledande linjär omskalning av talen. Låt oss visa hur vi med hjälp av lådpackningsalgoritmen kan sortera dessa tal.

Idé: Vi skalar om talen som ska sorteras med faktorn $1/(2n)$ så att dom ligger mellan $1/(2n)$ och $1/2$. Sedan konstruerar vi hjälpprylar som ska fylla lådorna så att man sedan får plats med exakt talen $1/(2n)$ till $1/2$ (i ordning från låda 1 till låda n). Om man först stoppar i dessa hjälpprylar och därefter dom omskalade talen enligt algoritmen så kommer talen att bli sorterade.

Anta att $v[1..n]$ är posterna som ska sorteras och att nyckelfältet heter *key*. Anta vidare att algoritmen för varje låda returnerar en lista med indexen för dom prylar som den innehåller.

```
Sort(v[1..n]) =
    p ← 1/(2n)
    for i ← 1  to n do x[i] ← 1 − i · p
```

for $i \leftarrow n+1$ **to** $2n$ **do** $x[i] \leftarrow v[i-n] \cdot p$
$L[1..n] \leftarrow$FirstFit$(x[1..2n])$
for $i \leftarrow 1$ **to** n **do** $res[i] \leftarrow v[L[i][2]-n]$ // ta andra prylen ur varje låda
return $res[1..n]$

Funktionen Sort reducerar alltså sorteringsproblemet till FirstFit. Reduktionen (oräknat anropet till FirstFit) tar tid $O(n)$, så om man kunde implementera FirstFit så den tar tid mindre än $\Omega(n \log n)$ så skulle det gå att sortera n tal snabbare än $\Omega(n \log n)$, vilket är omöjligt. □

Lösning till Reduktion som ger positivt resultat

Först bör vi försöka förstå problemet. Anta att X är en minimal kantmängd vars borttagande gör G osammanhängande. (Detta ska tolkas som att ingen strikt delmängd av X uppfyller detsamma.) Då gäller att G uppdelas i två komponenter, och alla kanterna i X går mellan dessa två. (Annars skulle ju X inte vara minimalt.) X svarar alltså mot ett snitt i grafen (en uppdelning av hörnmängden i två delar). Antalet kanter i X kan sägas vara snittets storlek. Detta innebär att det minsta antalet kanter som vi måste ta bort för att G ska bli osammanhängande — kalla detta $\lambda(G)$ — är lika med storleken på ett minsta snitt (V_1, V_2) i G.

Minimalt snitt är ju samma som maximalt flöde. Vi ska därför försöka reducera kantsammanhängandegradsproblemet till maximalt flöde mellan två hörn s och t i en graf. Om vi utökar G till en flödesgraf G' genom att ge varje kant dubbelriktad kapacitet 1, så kan vi alltså finna $\lambda(G)$ genom att beräkna maxflödet från s till t för olika s och t. Vi behöver dock inte variera båda dessa. Om vi väljer s godtyckligt så måste det höra till någon av mängderna V_1 och V_2 ovan. Genom att variera t över alla hörn utöver s kommer så vi garanterat att träffa något hörn i den andra mängden. Slutsatsen blir alltså att för ett godtyckligt hörn s gäller

$$\lambda(G) = \min_{t \in V - \{s\}} \{\text{MaxFlow}(G', s, t)\}.$$

Om vi ska vara noggranna så var nu uppgiften att avgöra om $K \geq \lambda(G)$. Vi kan alltså svara NEJ om $K <$ MaxFlow(G', s, t) för alla $t \in V - \{s\}$ och JA annars.

Flödesalgoritmen anropas $|V| - 1$ gånger. Varje flödesberäkning kan göras i tid $O(|V|^3)$ (vilket man inte behöver kunna utantill). Alltså blir komplexiteten för vår algoritm $O(|V|^4)$. □

Lösning till Reduktioner mellan besluts-, optimerings- och konstruktionsproblem

a) Vi vet att antalet färger är mellan 1 och n. Använd binärsökning i detta intervall för att med hjälp av algoritmen GraphColouring hitta ett k så att GraphColouring$(G, k) = 1$ och GraphColouring$(G, k-1) = 0$. Detta innebär nämligen att minimala färgningen har k färger. Det behövs $\log n$ halveringar av n för att komma ner till 1. Tidskomplexiteten blir därför $O(\log n \cdot T(n))$.

b) Hitta först det minimala antalet färger k med metoden ovan. Vi vill färga hörnen i G med färger med nummer 1 till k. Följande algoritm gör det:

CreateColouring$(G = (V, E),\ k)$=
$u \leftarrow$ första hörnet i V
$C \leftarrow \{u\}$; $u.colour \leftarrow k$
foreach $v \in V - \{u\}$ **do**
  if $(u, v) \notin E$ **then**
    if GraphColouring$((V, E \cup \{(u, v)\}), k) = 1$ **then** $E \leftarrow E \cup \{(u, v)\}$
    else $C \leftarrow C \cup \{v\}$; $v.colour \leftarrow k$
if $k > 0$ **then** *CreateColouring*$((V - C, E), k - 1)$

GraphColouring anropas högst en gång för varje par av hörn i grafen. Tidskomplexiteten för hela algoritmen blir därför $O(\log n \cdot T(n) + n^2 \cdot T(n)) = O(n^2 \cdot T(n))$. □