Algoritmer, datastrukturer och komplexitet

Övning 7

Anton Grensjö
grensjo@csc.kth.se

14 oktober 2015

Kursplanering

Ö6: Algoritmkonstruktion

F19: Probabilistiska algoritmer

Inlämning Mästarprov 1

F20: Reduktioner

Ö7: Probabilistiska algoritmer, reduktioner

F21: Introduktion till komplexitet

Redovisningar Mästarprov 1

ADK del 2: Komplexitet...

Idag

- Probabilistiska algoritmer
- Reduktioner

Probabilistiska algoritmer

- Probabilistiska algoritmer: använder slump
- Las Vegas-algoritm: svarar alltid rätt, kan ta olika lång tid
- Monte Carlo-algoritm: går alltid snabbt, kan ge olika bra resultat

Uppgift 1: Skipplistor

- Kom ihåg skipplistor från föreläsning 3.
- Sannolikheten att ett element som finns på nivå i också finns på nivå $i + 1$ kallas för p.
- En närmare analys ger att skipplistan för så kort sökstig som möjligt om $p = 1/e \approx 0{,}3679$.

  Uppgift Trots detta resultat kan det vara bättre att välja $p = 1/2$ eller $p = 1/4$. Varför?

  Svar Slump är dyrt.

- För $p = 1/2$ eller $p = 1/4$ räcker det med 1 eller 2 slumpmässiga bitar för att avgöra om en ny nivå ska läggas till eller inte, med rätt sannolikhet.

Uppgift 2: Verifikation av matrismultiplikation

- Snabbaste kända metoden för att multiplicera två $n \times n$ matriser är $\mathcal{O}(n^{2,376})$.
- Visa att det går att probabilistiskt verifiera en matrismultiplikation i kvadratisk tid.
- Närmare bestämt:
  - Konstruera en Monte Carlo-algoritm som verifierar att $AB = C$.
  - Tidskomplexiteten ska vara $\mathcal{O}(kn^2)$ och algoritmen ska ha en felsannolikhet på högst 2^{-k}.
- Hint: multiplikation av $n \times n$-matris med $n \times 1$-vektor har tidskomplexitet $\mathcal{O}(n^2)$.

Uppgift 2: Verifikation av matrismultiplikation

Lösningsidé

- Insikt: Antag att vi har en vektor $\vec{v}$. Då gäller

$$C = AB \implies C\vec{v} = A(B\vec{v})$$

- Om det skulle vara så att $C\vec{v} \neq A(B\vec{v})$ så vet vi att $C \neq AB$.

Algoritmen:

1. Slumpa fram en vektor $\vec{r} \in \{0, 1\}^n$.
2. Beräkna $\vec{a} = C\vec{r}$ och $\vec{b} = A(B\vec{r})$.
3. Kolla om $\vec{a} = \vec{b}$.

Kör steg 1-3 k stycken gånger. Om testet lyckas (dvs $\vec{a} = \vec{b}$) varje gång säger vi att multiplikationen $C = AB$ är korrekt, annars säger vi att den är felaktig.

Uppgift 2: Verifikation av matrismultiplikation

Pseudokod

VerifyMult$(n, A, B, C, k) =$
for $i \leftarrow 1$ **to** k **do**
r $\leftarrow$ slumpvektor$\in \{0,1\}^n$
a $\leftarrow C \circ$ **r**
b $\leftarrow A \circ (B \circ$ **r**$)$
if a $\neq$ **b then return** false
return true

Tidskomplexitet: $\mathcal{O}(kn^2)$

Uppgift 2: Verifikation av matrismultiplikation

Analys

Om $AB = C$ Algoritmen ger alltid korrekt svar.

Om $AB \neq C$ Algoritmen kommer ibland ge “false positive”, dvs säga att multiplikationen är korrekt, fast så inte är fallet. Vi vill analysera hur ofta detta sker.

- Antag att $AB \neq C$.

$$AB \neq C \iff D = AB - C \neq 0$$

- Vår algoritm kommer upptäcka felet om $D\vec{r} \neq 0$ (då har den upptäckt att $D = AB - C \neq 0$), och missa det om $D\vec{r} = 0$.

Uppgift 2: Verifikation av matrismultiplikation

Analys

- Vi vill undersöka hur ofta vår algoritm upptäcker att $AB \neq C$.
- Vi vet att $D = AB - C \neq 0$.
- Algoritmen upptäcker felet om och endast om $D\vec{r} \neq 0$.

- Insikt: det måste finnas en position k i $\vec{r}$, sådan att $D\vec{r}$ ändras om den biten flippas. Varför?
  - Betrakta ett nollskilt element d_{ik} i D (måste finnas, ty $D \neq 0$).
  - Låt $\vec{v} = D\vec{r}$. Utför matris-vektor-multiplikation:

$$v_i = \sum_{j=1}^{n} d_{ij} r_j = \ldots + d_{ik} r_k + \ldots$$

  - Om vi flippar biten r_k så ändras v_i!

Uppgift 2: Verifikation av matrismultiplikation

Analys

- Vi vill undersöka hur ofta vår algoritm upptäcker att $AB \neq C$.
- Vi vet att $D = AB - C \neq 0$.
- Algoritmen upptäcker felet om och endast om $D\vec{r} \neq 0$.
- Produkten $D\vec{r}$ ändras om biten r_k flippas.

- Partitionera in alla vektorer i $\{0,1\}^n$ i två mängder, en där $r_k = 0$ och en där $r_k = 1$.

$$V_0 = \left\{\vec{r} \in \{0,1\}^n \ : \ r_k = 0\right\}$$
$$V_1 = \left\{\vec{r} \in \{0,1\}^n \ : \ r_k = 1\right\}$$

- Låt $f : \ V_0 \to V_1$ vara funktionen som flippar bit k. Detta är en bijektion, dvs funktionen parar ihop varje vektor $\vec{r} \in V_0$ med exakt en vektor $\vec{r}' \in V_1$.
- Låt $\vec{r} \in V_0$, $\vec{r}' = f(\vec{r}) \in V_1$ vara ett sådant par.
  - Om $D\vec{r} = 0$ så är $D\vec{r}' \neq 0$.
  - Om $D\vec{r}' = 0$ så är $D\vec{r} \neq 0$.
- Minst en vektor i varje par måste avslöja bluffen!

Uppgift 2: Verifikation av matrismultiplikation

Analys

- Vi vill undersöka hur ofta vår algoritm upptäcker att $AB \neq C$.
- Vi vet att $D = AB - C \neq 0$.
- Algoritmen upptäcker felet om och endast om $D\vec{r} \neq 0$.
- Vektorerna i $\{0,1\}^n$ kan delas in i par, sådana att $D\vec{r} \neq 0$ åtminstone för en av vektorerna i paret.

- Därför måste åtminstone hälften av alla möjliga vektorer $\vec{r} \in \{0,1\}^n$ avslöja bluffen (dvs leda till att algoritmen upptäcker att $AB \neq C$).
- Sannolikheten att ett fel inte upptäcks vid ett varv av algoritmen är högst $1/2$.
- $\implies$ Efter k körningar är sannolikheten att ett fel missats högst 2^{-k}.

Detta var den felsannolikhet vi var ute efter! (Och det är en bra sådan, minskar exponentiellt med k.)

Reduktioner

- Reduktioner kan användas till att
  - Visa positiva resultat (hitta algoritmer)
  - Visa negativa resultat (undre gränser)
  - Visa att ett problem är svårare än ett annat
  - Visa att två problem är lika svåra
- Om P kan reduceras till Q och P är svårt, så är Q också svårt.
- Turingreduktion av P till Q: en algoritm som löser P genom att anropa Q en eller flera gånger.
- Karpreduktion av P till Q: specialfall av Turingreduktion, där Q anropas exakt en gång och resultatet returneras direkt.

Reduktioner

Olika typer av problem:

- Beslutsproblem (utdata är sant/falskt)
- Optimeringsproblem (maximera/minimera en målfunktion)
- Konstruktionsproblem

Uppgift 3: Reduktion som ger negativt resultat

Att hitta medianelementet (det $\lceil \frac{n}{2} \rceil$:e minsta elementet) av n positiva heltal kräver minst $2n$ jämförelser i värsta fallet.

Utnyttja detta resultat för att bestämma en undre gräns för antalet jämförelser som krävs för att hitta det $\lceil n/2 \rceil + 10$:e minsta elementet.

- Låt oss reducera medianproblemet till det nya problemet.
- Dvs givet en algoritm som löser det nya problemet, lös medianproblemet med hjälp av ett eller flera anrop till denna (Turing-reduktion).
- Eftersom vi har en undre gräns till medianproblemet (ingen algoritm kan göra bättre än så) så kommer detta även ge en undre gräns till vårt nya problem.

Uppgift 3: Reduktion som ger negativt resultat

- Givet: En algoritm MED10 som hittar det $\lceil n/2 \rceil + 10$:e minsta elementet i en array.
- Vi vill hitta medianen med hjälp av denna.
- Idé:
  - Lägg till 20 stycken ettor till vår array.
  - Den ursprungliga medianen ligger nu på plats $\lceil n/2 \rceil + 10$.
  - Anropa den givna algoritmen!

function MEDIAN(L, n)
    Lägg till 20 ettor till L.
    return MED10$(L, n + 20)$

Uppgift 3: Reduktion som ger negativt resultat

function MEDIAN(L, n)
    Lägg till 20 ettor till L.
    return MED10$(L, n + 20)$

- Säg att vår reduktion MEDIAN(L, n) använder $f(n)$ jämförelser, och att MED10(L, m) använder $g(m)$ jämförelser.
- Notera: $f(n) = g(n + 20)$.
- Den givna undre gränsen ger: $f(n) \geq 2n$. Således:

$$g(n + 20) \geq 2n$$

- Substituera $m = n + 20$:

$$g(m) \geq 2m - 40$$

Slutsats: att hitta det $\lceil n/2 \rceil + 10$:e minsta elementet i en array kräver minst $2n - 40$ jämförelser, oavsett algoritm.

Uppgift 4: Reduktion som ger positivt resultat

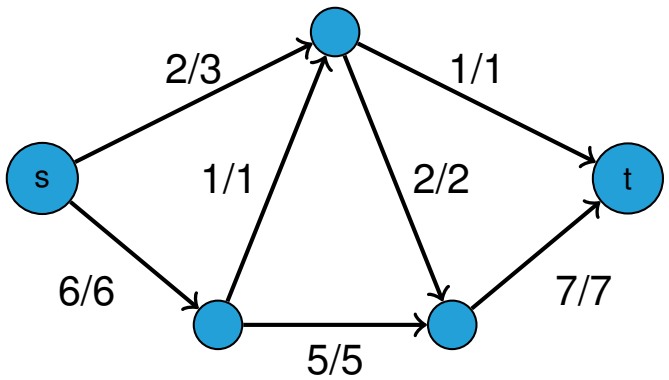
Ett ofta användbart sätt att lösa problem på är att hitta en reduktion till ett problem som man redan vet hur man ska lösa. Du ska nu använda denna metod för att lösa kantsammanhängandegradsproblemet som definieras på följande sätt.

> 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)?*

Uppgift 4: Reduktion som ger positivt resultat

Kom ihåg:

- Ett snitt i en graf är en indelning av den i två delar.
- Snittets storlek är summan av vikterna på de kanter som går mellan delarna.
- Känt resultat:
  - Minsta snittet som placerar noderna s och t i olika delar är lika med det maximala flödet från s till t.
  - Minsta snittet kan tolkas som hur mycket kapacitet vi måste blockera för att helt stoppa flödet från s till t.



- Om alla vikter är 1 kan minsta snittet tolkas som hur många kanter som måste tas bort för att lägga s och t i olika komponenter.

Uppgift 4: Reduktion som ger positivt resultat

Lösning:

- Gör om G till en flödesgraf G', ge dubbelriktad kapacitet 1.
- För varje par av noder s, t, beräkna minsta snittet (=maximala flödet).
- Det minsta av alla dessa resultat är antalet kanter som måste tas bort för att dela upp G i två komponener.
- Måste vi anropa maxflow för varje par av noder ($\mathcal{O}(|V|^2)$ gånger), eller kan vi göra det bättre?
  - Räcker med att fixera s och testa för alla möjliga t. Varför?
  - Låt $\lambda(G)$ vara det minsta resultatet av alla dessa anrop till maxflow.

$$\lambda(G) = \min_{t \in V - \{s\}} \left(\textsc{MaxFlow}(G', s, t) \right)$$

Uppgiften var att avgöra om $K \geq \lambda(G)$. Vi svarar alltså JA om $K \geq \lambda(G)$ och NEJ annars.

Uppgift 4: Reduktion som ger positivt resultat

Tidskomplexitet:

- Vi anropar maxflow $|V| - 1$ gånger.
- Varje flödesberäkning tar tid $\mathcal{O}(|V|^3)$.
- $\implies$ vår algoritm har komplexitet $\mathcal{O}(|V|^4)$.

Uppgift 5: Formulera optimeringsproblem som beslutsproblem

Jobbmaskningsproblemet definieras som ett optimeringsproblem på följande sätt:

Indata: Arbetsdagens längd T och n arbetsuppgifter. Uppgift i tar tiden t_i och kräver arbetsinsatsen w_i. Alla tal är positiva heltal.
Lösning: Ett urval av arbetsuppgifterna $D \subseteq [1..n]$ som tillsammans fyller ut en arbetsdag, det vill säga $\sum_{i \in D} t_i \geq T$.
Målfunktion: Den arbetsinsats som krävs för att utföra dagsverket D, alltså $\sum_{i \in D} w_i$.
Mål: Minimera målfunktionen, alltså hitta det dagsverke som kräver minst arbetsinsats.

a) Formulera optimeringsproblemet som ett beslutsproblem. Utvidga indata med ett målvärde I för arbetsinsatsen och formulera en fråga som har svaret ja eller nej.

b) Turingreducera optimeringsproblemet till beslutsproblemet, det vill säga visa hur man kan lösa optimeringsproblemet med hjälp av en algoritm som löser beslutsproblemet.

Uppgift 5: Formulera optimeringsproblem som beslutsproblem

a)

- Inför ett mål I som också ges som indata.
- Beslutsproblemet svarar på frågan om problemet är lösbart med en total arbetsinsats som är högst I.

Indata: Arbetsdagens längd T, målet I och n arbetsuppgifter. Uppgift i tar tiden t_i och kräver arbetsinsatsen w_i.
Fråga: Finns det en delmängd $D \subseteq [1..n]$ så att $\sum_{i \in D} t_i \geq T$ och $\sum_{i \in D} w_i \leq I$?

b)

- Antag att vi har en algoritm som löser beslutsproblemet.
- Den optimala totala arbetsinsatsen måste vara ett heltal mellan 0 och $\sum_{1 \leq i \leq n} w_i$.
- Binärsök mellan dessa extremvärden för det minsta värde I för vilket den givna algoritmen för beslutsproblemet svarar ja.

Uppgift 6: 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 $\mathcal{O}(\log n \cdot T(n))$.

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

Uppgift 6: Reduktioner mellan besluts-, optimerings och konstruktionsproblem

a)

- Det minimala antalet färger måste vara mellan 1 och n.
- Binärsök i detta intervall efter det minsta k sådant att GraphColouring$(G, k) = 1$.
- Intervallet halveras $\log n$ gånger $\implies$ komplexitet $\mathcal{O}(\log n \cdot T(n))$.

b)

- Använd först algoritmen ovan för att hitta minimala antalet färger k som behövs.
- Vi vill nu färga hörnen i G med färger 1 till k.
- Idé: Vi kan använda algoritmen för beslutsproblemet för att avgöra vilka val vi får/inte får göra.

Uppgift 6: Reduktioner mellan besluts-, optimerings och konstruktionsproblem

Algoritm:

1. Välj ett hörn u. Ge det färg k.
2. Iterera över alla andra noder v, som inte är granne med u.
   - Vi undrar: måste u och v ha samma färg?
   - Testa lägga till kanten (u, v) och anropa beslutsproblemet. Finns det fortfarande en lösning?
     - Om nej: u och v måste vara samma färg. Ta bort kanten och ge v färgen k.
     - Om ja: vi drar slutsatsen att u och v inte behöver ha samma färg. Låt v vara färglös och lämna kvar den nya kanten.
3.
   - Varje nod har nu antingen färg k eller en granne med färg k.
   - Vi vet att det fortfarande finns en lösning.
   - Kvarvarande ofärgade hörn måste ha färg $< k$.
   - De färgade hörnen påverkar inte längre lösningen.
   - Anropa rekursivt utan de färgade hörnen och med en färg mindre.

Uppgift 6: Reduktioner mellan besluts-, optimerings och konstruktionsproblem

$CreateColouring(G = (V, E), k)=$
$u \leftarrow$ 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 $k > 1$ **then** $CreateColouring((V - C, E), k - 1)$

Tidskomplexitet: GraphColoring anropas högst en gång för varje par av hörn i grafen. Således blir komplexiteten

$$\mathcal{O}(\log n \cdot T(n) + n^2 \cdot T(n)) = \mathcal{O}(n^2 \cdot T(n))$$

Vad händer framöver?

- Nu blir det mästarprovsredovisningar och tentaperiodsuppehåll.
- Efter tentaperioden börjar vi om med del 2 av kursen!
- Nästa övning handlar om oavgörbarhet, dvs problem som inte går att lösa!