Algoritmer, datastrukturer och komplexitet

Övning 6

Anton Grensjö
grensjo@csc.kth.se

9 oktober 2015

Kursplanering

Ö5: Grafalgoritmer och undre gränser

F16: Sortering i linjär tid

F17: Textsökning

F18: Polynom och FFT

Ö6: Algoritmkonstruktion

F19: Probabilistiska algoritmer

Deadline Mästarprov 1

F20: Reduktioner

Ö7: Probabilistiska algoritmer, reduktioner

Idag

- Algoritmkonstruktion (lite blandat)
- Redovisning och inlämning av labbteori 3

Uppgift 1: Inuti eller utanför?

- P är en konvex n-hörnig polygon.
- Beskrivs som en array av hörnen $p_1, p_2, \ldots, p_n$ i medurs ordning.
- Konstruera en algoritm som talar om ifall en given punkt q ligger inuti P.
- Algoritmen ska gå i tid $\mathcal{O}(\log n)$ i värsta fallet.

Uppgift 1: Inuti eller utanför?

Lösning

1. En idé:
   - Dra en stråle från punkten q. Räkna antalet gånger som den skär polygonen.
   - Udda antal $\implies$ q ligger inuti P.
   - Jämnt antal $\implies$ q ligger utanför P.
   - Fungerar även för icke-konvexa polygoner!
   - Finns ett problem. Vad? Tidskomplexitet $\mathcal{O}(n)$.
2. En annan idé:
   - Genom att utnyttja att P är konvex kan vi hitta en snabbare algoritm.
   - Konvexa polygoner kan delas in i trianglar.
   - Kanske kan vi använda dekomposition?

Uppgift 1: Inuti eller utanför?

Lösning

- Idé: dela polygonen på mitten, kolla på vilken sida av snittet q ligger, rekursera. (Binärsökning/intervallhalvering.)
- Hur tar vi reda på vilken sida av en linje som en punkt ligger på?
  - Linjär algebra.
  - Låt $\vec{v}$ vara en vektor längs med linjen, och $\vec{u}$ vara en vektor från linjen till punkten.
  - Betrakta kryssprodukten $\vec{v} \times \vec{u}$.
  - Både $\vec{v}$ och $\vec{u}$ ligger i tavlans plan (säg xy-planet), så produkten måste peka antingen in i eller ut ur tavlan (den har endast z-komponent).
  - Tecknet på z-komponenten avgör vilken sida av linjen som q ligger på.
  - Hur beräknar vi kryssprodukten?

Uppgift 1: Inuti eller utanför?

Lösning

- Algoritmen:
  - Välj ett hörn v_1.
  - Dra en linje från v_1 till motsatt hörn, låt den dela polygonen i två delar.
  - Kolla på vilken sida av snittet q ligger.
  - Rekursera på den delen av polygonen.
  - Fungerar ungefär som binärsökning.
- Basfall? Polygonen är endast en triangel! Lös med vår första idé på konstant tid.
- Tidskomplexitet:
  - Varje steg går på konstant tid.
  - Delen av polygonen vi tittar på halveras vid varje rekursivt anrop.

$$T(n) = T\left(\frac{n}{2}\right) + \mathcal{O}(1) \implies T(n) \in \mathcal{O}(\log n)$$

Uppgift 1: Inuti eller utanför?

Pseudokod

$InsideConvex(P, q, l, u) =$
  if $u = l + 1$ **then** /* en triangel */
    välj en punkt q' utanför triangeln p_1–p_l–p_u
    if linjen q–q' skär exakt en av kanterna i triangeln **then**
      return inuti
    else
      return utanför
  else
    $mid \leftarrow \lceil \frac{l+u}{2} \rceil$
    if q är på samma sida om linjen p_1–p_{mid} som p_{mid+1} **then**
      return $InsideConvex(P, q, mid, u)$
    else
      return $InsideConvex(P, q, l, mid)$

Algoritmen anropas med $InsideConvex(P, q, 2, n)$.

Uppgift 2: Sortering av små heltal

Uppgift Konstruera en algoritm som sorterar n heltal som alla ligger i intervallet $[1..n^3]$ i tid $\mathcal{O}(n)$ med enhetskostnad.

- Tid $\mathcal{O}(n)$? Hur är det möjligt? Är inte $\Omega(n \log n)$ en undre gräns till sorteringsproblemet?
  - Undre gränsen $\Omega(n \log n)$ gäller enbart för jämförelsebaserade sorteringsalgoritmer.
  - Här vet vi dels att det är tal vi ska sortera, samt något om hur stora de är.
  - Detta kan utnyttjas för att lösa problemet på linjär tid.
- Hint: tänk på räknesortering och radixsortering.

Uppgift 2: Sortering av små heltal

Lösning

- Representera talen som strängar i basen n.
- Varje tal kan skrivas med högst 3 siffror, och det finns n olika siffror.
- Lägg till inledande nollor så alla får längd 3.
- (Notera att specialfallet n^3 har fyra siffror (skrivs 1000). Behandla detta specialfall separat.)
- Tresiffriga tal $\implies$ tre omgångar i radixsorteringen. Varje omgång sorterar talen efter en av dess tre siffror.
- Varje steg utförs med hjälp av räknesortering (stabil, se föreläsning 16).
- Börja med minst signifikanta siffran och fortsätt åt vänster.

Se exempel på tavlan.

Uppgift 2: Sortering av små heltal

Tidskomplexitet

Varje omgång räknesortering går på tiden $\mathcal{O}(n)$. Vi gör tre omgångar.

$$\Longrightarrow \quad \text{tidskomplexiteten är } \mathcal{O}(n)$$

Uppgift 3: Hitta det saknade talet

Uppgift På en fil ligger $999\,999\,999$ tal, nämligen alla tal mellan 1 och $1\,000\,000\,000$, förutom ett av dem.
Konstruera en algoritm som tar reda på vilket tal som saknas.

- Krav (låt n vara antalet tal):
  - Minnesanvändning: $\mathcal{O}(1)$.
  - Tidskomplexitet: $\mathcal{O}(n)$.
  - Beräkningsmodell med 32-bitarsaritmetik.

Uppgift 3: Hitta det saknade talet

Vi betraktar först några enklare varianter av problemet.

- Om vi tillåter linjär minnesanvändning:
  - Skapa en bitarray med ett element för varje tal mellan 1 och n.
  - Iterera över alla tal i filen och sätt dess bit till 1.
  - Vilket tal har ej fått sin bit satt?
- Om vi tillåter godtyckligt stora tal:
  - Beräkna summan S av alla tal mellan 1 och n:

$$S = \frac{n(n+1)}{2}$$

  - Beräkna summan S' av alla tal i filen.
  - Det saknade talet är $x = S - S'$.
  - Notera: Detta behöver analyseras med bitkostnad. Kräver mycket minne.

Uppgift 3: Hitta det saknade talet

Lösning

Så hur löser vi problemet med alla givna krav (linjär tid, konstant minne, 32-bitarsaritmetik)?

- Ledning: Vi kan inspireras av den senast nämnda lösningen.
- Beräkna summan S av talen $1, \ldots, n$, modulo n (vi antar n jämnt).

$$S = \frac{n(n+1)}{2} = \frac{n^2}{2} + \frac{n}{2} \equiv \frac{n}{2} \pmod{n}$$

- Beräkna summan S' av alla tal i filen, modulo n.
- Låt x vara det saknade talet. Vi måste då ha:

$$S \equiv S' + x \pmod{n} \iff x \equiv S - S' \equiv \frac{n}{2} - S' \pmod{n}$$

- Vi kan alltså nu enkelt beräkna x med hjälp av formeln ovan. I fallet $n = 10^9$:

$$x \leftarrow \left(\frac{3n}{2} - S'\right) \%10^9$$

Uppgift 3: Hitta det saknade talet

Alternativ variant

Alternativ variant av problemet:

- Vi betraktar nu alla möjliga n-bitarsheltal, dvs $0, 1, \ldots, 2^n - 1$.
- Alla dessa tal utom ett står nedskrivet i en fil. Lista ut vilket med linjär tid och konstant minne.

Lösning:

- Vad händer om vi tar xor av alla tal?
  - Svar: 0. Varför?
- Vad händer om vi tar xor av alla tal utom ett?
  - Svar: Vi får det saknade talet! Varför?

Testa med $n = 3$!

Uppgift 4: Komplex multiplikation

Hur multiplicerar man vanligen komplexa tal?

$$z = (a + bi)(c + di) = ab + adi + bci - bd = (ac - bd) + i(ad + bc)$$

Dvs: $\mathrm{Re}(z) = ac - bd, \quad \mathrm{Im}(z) = ad + bc$.
Fyra multiplikationer och två additioner.

Uppgift Multiplikationer är dyrare än additioner, så det lönar sig att minimera antalet multiplikationer. Hitta en algoritm som bara använder tre multiplikationer för att multiplicera två komplexa tal.

Uppgift 4: Komplex multiplikation

$$z = (a + bi)(c + di) = (ac - bd) + i(ad + bc)$$

Knep: definiera och beräkna följande:

$$A_1 = (a + b)(c - d) = ac - ad + bc - bd$$

$$A_2 = ad, \quad A_3 = bc$$

Vi får att

$$\mathrm{Re}(z) = A_1 + A_2 - A_3$$

$$\mathrm{Im}(z) = A_2 + A_3$$

Totalt: 3 multiplikationer och 5 additioner.

Uppgift 5: Binärträd med speglad struktur

- Två binärträd har speglad struktur om det ena är en spegelbild av det andra. Med andra ord, om man byter höger mot vänster i det ena trädet så blir de strukturekvivalenta.
- Konstruera en dekompositionsalgoritm som avgör ifall två binärträd har speglad struktur.

Uppgift 5: Binärträd med speglad struktur

Lösning

- Rekursiv idé: Subträden rotade i u resp. v är speglade omm det vänstra subträdet till u är en spegelbild av det högra subträdet till v, och vice versa.
- Basfall: antingen u eller v är det tomma subträdet. Då är de spegelbilder om och endast om båda är tomma.
- Vi anropar alltså vår rekursiva algoritm på roten i respektive träd, och den kommer traversera båda träden samtidigt (fast spegelvänt) och jämföra strukturen.

SpegladeTräd(T_1, T_2)=
    if T_1 =NIL or T_2 =NIL then
        return T_1 =NIL and T_2 =NIL
    return SpegladeTräd(T_1.left,T_2.right) and SpegladeTräd(T_1.right,T_2.left)

- Tidskomplexitet: $\mathcal{O}(n)$.

Uppgift 6: Partyproblemet!

- Du vill hålla party. Till din hjälp har du en lista med n personer, ett heltal k och en lista över vilka av personerna på listan som känner varandra.
- Du vill bjuda så många av personerna som möjligt, men för att alla ska trivas så bestämmer du att varje inbjuden gäst måste känna minst k av de övriga gästerna.
- Vilka personer ska du bjuda?

Konstruera och analysera en algoritm som löser detta problem i linjär tid i indatas storlek.

Uppgift 6: Partyproblemet!

Lösning

- Vad är en lämplig representation av problemet?
  - Se det som en graf!
  - Personerna är hörnen, och det går kanter mellan de personer som känner varandra.
  - Lösningen är den största delgrafen där varje hörn har minst gradtal k.
- Se exempel.
- Allmän strategi:
  - Om det finns något hörn med gradtal lägre än k så kan det ej vara med i lösningen. Ta bort det hörnet. (Detta kan leda till att ett annat hörn nu har för lågt gradtal).
  - Upprepa tills det bara finns hörn med lägst grad k kvar.

Uppgift 5: Partyproblemet!

Pseudokod

foreach $x \in V$ **do**
    if $d_x < k$ **then Q.Put(**x**)**
while not Q.Empty() do
    $x \leftarrow$ **Q.Get()**
    foreach $(x, v) \in x$**.kantlista do**
        if $d_v \geq k$ **then**
            $d_v \leftarrow d_v - 1$
            if $d_v < k$ **then Q.Put(**v**)**
write ’**Lösningen består av:**’
foreach $x \in V$ **do**
    if $d_x \geq k$ **then write** x

- Tidskomplexitet: $\mathcal{O}(n + m)$.
- Korrekthet
  - Varje hörn som skrivs ut har minst k grannar kvar i grafen.
  - Inget hörn som inte har färre än k grannar kvar i grafen har plockats bort $\implies$ lösningen måste vara maximal.

Nästa vecka

- Probabilistiska algoritmer
- Reduktioner