

Algoritmer (datastrukturer) och komplexitet, våren 2010

Uppgifter till övning 6

Algoritmkonstruktion

Inuti eller utanför? Låt P vara en konvex n -hörnig polygon beskriven som en array av hörnen p_1, p_2, \dots, p_n i cyklisk ordning. Konstruera en algoritm som talar om ifall en given punkt q är inuti P . Algoritmen ska gå i tid $O(\log n)$ i värsta fallet.

Mobil Konstruera en algoritm som balanserar en mobil! Mobilen beskrivs som ett binärträd. Löven motsvarar kulor som hänger underst i mobilen. Inre noder motsvarar tunna raka pinnar i vars ändrar är fastknutet trådar som sönerna hänger i. Indata är ett binärträd med vikter i noderna. Vikten i varje löv är motsvarande kulas massa. Vikten i varje inre nod är motsvarande pinnes längd i cm. Resultatet av algoritmen ska vara samma binärträd men där det i varje inre nod också står hur många centimeter från vänstra ändpunkten tråden som fäster pinnen i ovanförliggande pinne ska sitta för att mobilen ska bli balanserad.

Sortering av små heltal Konstruera en algoritm som sorterar n heltal som alla ligger i intervallet $[1..n^3]$ i tid $O(n)$ med enhetskostnad.
Tips: tänk på räknearter och radixsortering.

Hitta det saknade talet På en datafil ligger 999 999 999 tal, nämligen alla heltal mellan 1 och 1 000 000 000 utom ett. Vilket tal är det som saknas? Konstruera en algoritm som löser detta problem i konstant minne och linjär tid i en beräkningsmodell som har 32-bitsaritmetik (dvs kan räkna med tal av storlek mindre än $2^{31} = 2147483648$).

Tvådimensionell Fouriertransform Anta att du har tillgång till en FFT-implementation för det endimensionella fallet, men att du behöver transformera tvådimensionella data, till exempel transformera en bild given som en matris $(a_{ij})_{i,j \in \{0, \dots, N-1\}}$ till ett tvådimensionellt frekvensspektrum. Hur bör du utnyttja FFT för detta? Vad blir tiden?

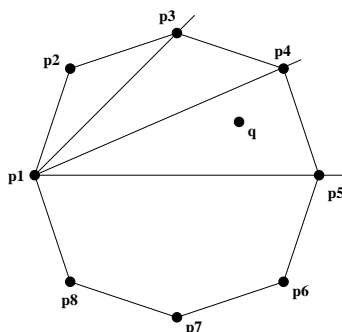
Binärträd med speglad struktur Två binära träd sägs ha *speglad struktur* om det ena är en spegelbild av det andra, det vill säga att om man byter vänster och höger överallt i det ena trädet så blir träden strukturekvivalenta. Konstruera och analysera tidskomplexiteten för en effektiv dekompositionsalgoritm som avgör ifall två binärträd har speglad struktur.

Partyproblemet Indata är en lista med n personer, ett heltal k och en lista med vilka personer som känner varandra. Du vill bjuda så många av personerna som möjligt på ditt party, men för att alla ska trivas vill du att varje inbjuden gäst ska känna minst k av dom övriga gästerna. Konstruera och analysera en algoritm som löser detta problem i linjär tid i indatas storlek.

Lösningar

Lösning till Inuti eller utanför?

Om polygonen inte hade varit konvex hade vi räknat antalet skärningar polygonen har med en linje från q till en punkt utanför P , vilket tar tid $O(n)$, men det har vi inte tid med här. Istället kommer vi att använda en intervallhalveringsliknande sökning för att utesluta hälften av (den återstående) polygonen i taget ända tills bara en triangel återstår. Därefter kan vi lätt (med ett konstant antal jämförelser) avgöra ifall q ligger i P . Se figur 1.



Figur 1: En konvex polygon och linjerna som algoritmen använder för att halvera den.

```
InsideConvex(P, q, l, u) =  
  if u = l + 1 then /* en triangel */  
    välj en punkt q' utanför triangeln p1-p_l-p_u  
    if linjen q-q' skär exakt en av kanterna i triangeln then  
      return inuti  
    else  
      return utanför  
  else  
    mid ← ⌈(l+u)/2⌉  
    if q är på samma sida om linjen p1-p_mid som p_{mid+1} then  
      return InsideConvex(P, q, mid, u)  
    else  
      return InsideConvex(P, q, l, mid)
```

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

Om vi antar att $InsideConvex(P, q, 2, n)$ tar tid $T(n)$ så får vi rekursionsekvationen

$$T(n) = T\left(\frac{n}{2}\right) + c$$

som har lösningen $c \log n$. Tidskomplexiteten blir alltså $T(n) \in O(\log n)$. □

Lösning till Mobil

Betrakta en pinne av längd l i mobilen. Anta att vikten v hänger i vänstra änden och vikten w i den högra. Låt x vara avståndet från pinnens vänstra ände till tråden som fäster pinnen i ovanförliggande pinne. För att momenten ska ta ut varandra krävs enligt mekaniken att $xv = (l - x)w$, det vill säga att $x = lw/(v + w)$. Vi kan nu beräkna x för varje pinne rekursivt nerifrån och upp i trädet. Låt den rekursiva funktionen returnera vikten av mobilen som hänger i den aktuella pinnen. Algoritmen tar linjär tid.

```

Balance(p)=
  if p.left=NIL then return p.num
  left ← Balance(p.left)
  right ← Balance(p.right)
  p.x ← p.num · r/(left + right)
  return left + right

```

□

Lösning till Sortering av små heltal

Betrakta heltalen skrivna i bas n . Alla tal har då (högst) tre siffror, utom n^3 som skrivs som 1000 och kan hanteras separat. (Gå igenom alla tal, plocka ut alla förekomster av n^3 och lägg dom sist i den sorterade följd.) Lägg till inledande nollor på tal som är mindre än n^2 . Gör nu radixsortering på talen. Eftersom det är tresiffriga tal blir det tre omgångar i radixsorteringen, där man i varje omgång ska sortera talen efter en siffra i intervallet $[0..n - 1]$. En sån sortering görs i linjär tid med räknasortering (som i implementationen från föreläsning 9 är stabil).

Analys: Tre omgångar görs och varje omgång tar tid $O(n)$. Specialhanteringen av n^3 tar inte heller mer än $O(n)$. Totalt blir tiden alltså $O(n)$. □

Ledning till Hitta det saknade talet

Summera alla tal modulo till exempel 1 000 000 000.

Lösning till Tvådimensionell Fouriertransform

Vi utnyttjar att Fouriertransformen är separabel. Definitionen av 2D-Fouriertransformen:

$$\hat{a}_{kl} = \frac{1}{N^2} \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} a_{ij} w^{ik+jl}$$

w är en lämplig enhetsrot ($w^N = 1$). Faktorn w^{ik} kan nu flyttas ur den innersta summan. Om vi därefter ser den innersta summan som en endimensionell Fouriertransform \hat{b}_{il} så är \hat{a}_{kl} en endimensionell transform av \hat{b}_{il} .

$$\hat{b}_{il} = \frac{1}{N} \sum_{j=0}^{N-1} a_{ij} w^{jl}, \quad \hat{a}_{kl} = \frac{1}{N} \sum_{i=0}^{N-1} \hat{b}_{il} w^{ik}$$

Dessa ekvationer säger att vi kan Fouriertransformera tvådimensionella data genom att först transformera varje kolumn i matrisen (a_{ij}) och sedan transformera raderna i den resulterande matrisen (b_{il}) (eller tvärtom om man vill det). Algoritmen blir då

```

2D-FFT(a0,0, a0,1, ..., aN-1,N-1, N) =
  for i ← 0 to N - 1
    bi,0, ..., bi,N-1 ← FFT(ai,0, ai,1, ..., ai,N-1, N)
  for i ← 0 to N - 1
    c0,i, ..., cN-1,i ← FFT(b0,i, b1,i, ..., bN-1,i, N)
  return c0,0, c0,1, ..., cN-1,N-1

```

FFT anropas $2N$ gånger. Eftersom varje anrop tar $O(N \log N)$ blir totaltiden $O(N^2 \log N)$. □

Lösning till Binärträd med speglad struktur

Vi använder dekomposition för att stega oss ner i båda träden samtidigt. För att träden ska vara varandras spegelbilder måste vänstra delträdet till första trädet vara spegelbilden av högra

delträdet till andra trädet och högra delträdet till första trädet vara spegelbilden av vänstra delträdet till andra trädet. Basfallet blir att minst ett av träden är tomt. I så fall är träden spegelbilder av varandra bara om båda träden är tomma.

```

SpegladeTräd( $T_1, T_2$ )=
  if  $T_1 = \text{NIL}$  or  $T_2 = \text{NIL}$  then
    return  $T_1 = \text{NIL}$  and  $T_2 = \text{NIL}$ 
  return SpegladeTräd( $T_1.\text{left}, T_2.\text{right}$ ) and SpegladeTräd( $T_1.\text{right}, T_2.\text{left}$ )

```

Eftersom funktionen kommer att anropas (högst) en gång för varje delträd (rot) och arbetet i ett funktionsanrop förutom rekursionen är konstant så blir komplexiteten linjär i trädets storlek.

Det är lätt att inse att algoritmen är korrekt med hjälp av så kallad strukturell induktion, det vill säga induktion över trädets struktur. Vi kollar först att basfallet stämmer (om ena trädet är tomt så måste också det andra trädet vara tomt för att dom ska vara spegelbilder) och kollar sedan att induktionssteget stämmer (om första trädets vänstra delträd är spegelbilden av andra trädets högra delträd och första trädets högra delträd är spegelbilden av andra trädets vänstra delträd så måste träden vara varandras spegelbilder). \square

Lösning till Partyproblemet

Anta att listan med vilka gäster som känner varandra består av m stycken rader. Det är enkelt att representera indata som en graf $G = (V, E)$ där varje person motsvarar ett hörn och varje kant (x, y) motsvarar att person x och y känner varandra. Representera grafen med kantlistor.

Den sökta lösningen är den största (inducerade) delgraf där varje hörn har gradtal minst k . Vi kan hitta denna delgraf genom att plocka bort varje hörn som har gradtal mindre än k . När ett hörn plockas bort tas samtidigt alla kanter till hörnet bort, varför andra hörn får lägre gradtal och kan behöva plockas bort. Detta implementeras enklast med en variabel d_x i varje hörn x som håller reda på det aktuella gradtalet och som uppdateras varje gång en kant till det hörnet plockas bort. Låt oss också ha en kö Q där vi lägger alla hörn som har gradtal mindre än k i väntan på att deras kanter ska plockas bort.

```

foreach  $x \in V$  do
  if  $d_x < k$  then Q.Put( $x$ )
while not Q.Empty() do
   $x \leftarrow$  Q.Get()
  foreach  $(x, v) \in x.\text{kantlista}$  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  $x$ 

```

Eftersom det finns n hörn och varje hörn behandlas ett konstant antal gånger (initiering av Q , köinsättning, köutplockning, utskrift), och eftersom det finns m kanter och varje kant behandlas högst två gånger (den förekommer i två kantlistor) så blir totala komplexiteten $O(n + m)$, alltså linjärt i indata storlek.

Det är enkelt att se att algoritmen är korrekt eftersom varje hörn som skrivs ut i lösningen fortfarande har minst k grannar kvar i grafen, och inget hörn som inte har färre än k grannar kvar i grafen har plockats bort under algoritmens gång. \square