Universiteit Twente
Afdeling Informatica
6 december 2010

Kwartiel 2010/2
1^e huiswerkserie

Algoritmen, Datastructuren en Complexiteit (192140200)

Uitwerkingen

De **deadline** voor het inleveren van deze huiswerkserie is woensdag 22 december. U levert het werk in in het postvak van uw werkcollegedocent. Bij de opgaven waar om een algoritme wordt gevraagd, geeft u de pseudo-code van uw oplossing en een beknopte maar duidelijke uitleg van de werking. Plagiaat zal streng worden bestraft.

Deze opgave zal worden beoordeeld met twee minnen, een min, een voldoende, of een plus. Deze huiswerkserie kan maximaal beloond worden met 0.3 punt extra bij het tentamencijfer, onder de voorwaarden die bij de spelregels worden uitgelegd. Veel succes!

1. Beschouw het volgende sorteeralgoritme dat van een rij A van integers het segment $A[i, \ldots, j]$ sorteert waarbij $1 \leqslant i \leqslant j$:

```
void sort (int [] A, int i, j) {
  if (A[i] > A[j]) swap (A[i], A[j]); // wissel A[i] en A[j] om
  if (i+1 >= j) return;
  int k = floor ((j-i+1) / 3); // floor = naar beneden afronden
  sort (A, i, j-k);
  sort (A, i+k, j);
  sort (A, i, j-k);
}
```

   Gevraagd:

   (a) Bepaal de asymptotische ordegrootte van de worst-case tijdscomplexiteit van *sort* om $n > 0$ getallen te sorteren. Neem als basisoperatie een vergelijking tussen elementen in A.

   Uitwerking:

   Het sorteren van een array ter lengte 1 of 2 kost 1 vergelijking. Voor arrays met meer elementen vinden er 3 recursieve aanroepen plaats, ieder met een array ter lengte $\frac{2}{3}$ van de originele lengte. De niet-recursieve kosten in dit geval zijn 1, daar er 1 vergelijking van A-elementen plaatsvindt. Dit leidt tot de volgende recurrente betrekking voor $W(n)$:

$$\begin{array}{lcll} W(n) & = & 1 & \text{voor } n < 3 \\ W(n) & = & 3 \cdot W(\frac{2n}{3}) + 1 & \text{voor } n \geqslant 3 \end{array}$$

Voor het bepalen van de worst-case tijdscomplexiteit passen we het Master theorema toe. Er volgt dat $a = 3$, $b = \frac{3}{2}$, $f(n) = 1$, en $E = \log a / \log b = \log 3 / \log \frac{3}{2}$. Aangezien $f(n) \in O(n^{E-\epsilon})$ voor bijvoorbeeld $\epsilon = \frac{1}{2}$ is de eerste case van het Master theorema van toepassing, en geldt dat $W(n) \in \Theta(n^E)$.

(b) Onder welke omstandigheden zou u *sort* prefereren als sorteeralgoritme boven quicksort, insertion sort, mergesort en heapsort?

Uitwerking:

Merk op dat $E = \log 3 / \log \frac{3}{2} = {}^{1.5}\log 3 > 2$ omdat $1.5^2 = 2.25$. Dus de worst-case tijdscomplexiteit van *sort* is significant slechter is dan die van de andere sorteer-algoritmen; we geven dus nooit de voorkeur aan *sort*.

2. Geef voor de volgende paren functies zo precies mogelijk weer wat hun relatie is v.w.b. hun asymptotische orde (bv. $f \in O(g)$ etc.).

(a) $log\ n^2$, $\sqrt{n}$

Uitwerking:

$\lim_{n\to\infty} \frac{\log n^2}{\sqrt{n}} = \lim_{n\to\infty} \frac{2 \cdot \log n}{\sqrt{n}} = 0$ aangezien $\lim_{n\to\infty} \frac{\log n}{n^p} = 0$ voor alle $p > 0$. Dus $\log n^2 \in O(\sqrt{n})$.

(b) n^k, e^n (hint: herhaaldelijk L'Hopital toepassen)

Uitwerking:

$\lim_{n\to\infty} \frac{n^k}{e^n} = \lim_{n\to\infty} \frac{k \cdot n^{k-1}}{e^n} = \lim_{n\to\infty} \frac{k \cdot (k-1) \cdot n^{k-2}}{e^n} = \ldots = \lim_{n\to\infty} \frac{k!}{e^n} = 0$.

Dus $n^k \in O(e^n)$.

(c) ${}^3\log n$, ${}^2\log n$

Uitwerking:

$\lim_{n\to\infty} \frac{{}^3\log n}{{}^2\log n} = \lim_{n\to\infty} \frac{{}^3\log 2 \cdot {}^2\log n}{{}^2\log n} = {}^3\log 2$.

Dus ${}^3\log n \in \Theta({}^2\log n)$.

3. Vind de asymptotische orde van de oplossingen van de volgende twee recurrente betrekkingen:

(a) $T(n) = T(\frac{n}{2}) + 2 \cdot log\ n$, $T(1) = 1$

Uitwerking:

$a = 1$, $b = 2$, dus $E = \frac{\log 1}{\log 2} = 0$. Het blijkt dat het Master theorema niet kan worden toegepast, dus we analyseren de recursieboom. De diepte van de recursieboom is $\log n$. Er is één enkel blad met kosten 1. De kosten op level i zijn $2 \cdot \log \frac{n}{2^i}$. Dus de totale kosten:

$$
\begin{aligned}
1 + \sum_{i=0}^{\log n - 1} 2 \cdot \log \frac{n}{2^i} &= 1 + \sum_{i=0}^{\log n - 1} 2 \cdot (\log n - \log 2^i) \\
&= 1 + \left(2 \sum_{i=0}^{\log n - 1} \log n \right) - \left(2 \sum_{i=0}^{\log n - 1} \log 2^i \right) \\
&= 1 + 2 \cdot (\log n)^2 - \left(2 \sum_{i=0}^{\log n - 1} i \cdot \log 2 \right) \\
&= 1 + 2 \cdot (\log n)^2 - \left(2 \cdot \log 2 \sum_{i=0}^{\log n - 1} i \right) \\
&= 1 + 2 \cdot (\log n)^2 - 2 \cdot \frac{(\log n - 1)(\log n)}{2} \\
&= 1 + 2 \cdot (\log n)^2 - \log n \cdot (\log n - 1)
\end{aligned}
$$

Dus $T(n) \in \Theta((\log n)^2)$.

(b) $T(n) = 2 \cdot T(\frac{n}{4}) + 8$, $T(n) = 0$ voor $n < 4$

Uitwerking:

Master theorema toepassen: $a = 2$, $b = 4$, $E = \frac{\log 2}{\log 4} = \frac{1}{2}$. $f(n) = 8 \in O(n^{0.5-\epsilon})$ voor bijvoorbeeld $\epsilon = 0.1$. Dus we hebben te maken met geval 1, dus $T(n) \in \Theta(\sqrt{n})$.

4. Geef een algoritme die voor een postorder gesorteerde binaire boom met positieve elementen bepaalt of deze een paar elementen bevat die 1 verschillen.

Uitwerking:

```
(int, bool) postorder (int val, node root)
{ if root == null return (val, false)
  else
  { (int max, bool found)  = postorder(val, root.left)
    if found return (max, true)
    else
     { (max, found) = postorder(max, root.right)
       if found  return (max, true)
       else return (root.key, (root.key==max+1))
     }
  }
}
```

Omdat de keys positief zijn, kunnen we voor een boom N de functie aanroepen met $postorder(-1, N)$.

5. (a) Stel je verandert in een heap een willekeurig element (stel, met index i). Geef een algoritme *ExtHeapify(E,i)* die er voor zorgt dat de heapeigenschap zonodig hersteld wordt.

   Uitwerking:

   Stel de heap is gegeven in array E. Er zijn drie mogelijkheden:

   i. E is nog steeds een heap, dan ben je klaar.

   ii. De nieuwe waarde k van $E[i]$ is groter dan zijn ouder. Swap dan $E[i]$ met zijn ouder. Herhaal dit netzolang tot k op een plek staat waar ie kleiner is dan zijn ouder; je hebt nu een heap.

   iii. De nieuwe waarde k van $E[i]$ is kleiner dan zijn ouder, maar ook kleiner dan minstens een van zijn kinderen. Roep nu *Heapify(E,i)* aan.

   (b) Geef een algoritme die een element met index i uit een heap verwijdert, en ervoor zorgt dat het resultaat weer een heap is (hint: gebruik *ExtHeapify*.

   Uitwerking:

   Swap $E[i]$ met $E[E.heapsize]$, en verlaag de heapsize met 1. Roep nu *ExtHeapify(E,i)* aan.

6. Neem de totaal gebalanceerde BST met de elementen $10, 20, \ldots, 70$. Laat zien hoe deze BST met rotaties tot een lijst getransformeerd kan worden.

   Uitwerking:

   Er zijn veel mogelijke oplossingen. Een gestructureerde:

   ```
   Leftrotate(20)
   Rightrotate(60)
   Rightrotate(40)
   Rightrotate(30)
   Rightrotate(20)
   ```

   Een korte:

   ```
   Rightrotate(40)
   Rightrotate(20)
   Rightrotate(40)
   Rightrotate(60)
   ```

7. Geef van de volgende beweringen aan of ze waar of onwaar zijn, en motiveer je antwoord.

   (a) Beschouw de recurrente betrekking $T(n) = T(\frac{n}{2}) + 2 \cdot log\ n$, $T(1) = 1$. Volgens Masters theorema geldt $T(n) \in \Theta(log\ n)$.

   Uitwerking:

   Onwaar. $a = 1$, $b = 2$, dus $E = 0$, nu geldt niet $log\ n \in O(n^{-\epsilon})$ (geval 1) of $log\ n \in \Theta(1)$ (geval 2) of $log\ n \in \Omega(n^{\epsilon})$ (geval 3), dus het Master theorema is niet toepasbaar.

(b) Stel je gebruikt voor open adressering een hastabel met 400 locaties. Stel dat de kans, dat een item op een locatie wordt afgebeeld, voor alle items en alle locaties even groot is. De kans dat het vijfde toegevoegde item to een hashcollision leidt is 0,01.

Uitwerking:

Waar. Met 4 bezette locaties is de kans dat het vijfde item een hash collision veroorzaakt gelijk aan $4/400 = 0,01$.

(c) Het kleinste element van een maxheap bevindt zich altijd in een blad.

Uitwerking:

Waar. Als een element geen blad is, heeft ie een kind dat kleiner is.