Universiteit Twente 2009-2010/2
Afdeling Informatica, Faculteit EWI Tentamen
dinsdag 19 januari 2010, 8.45-12.15

Algoritmen, Datastructuren en Complexiteit

(214020 en 214025)

Uitwerkingen

Bij dit tentamen mag het boek van Baase en Van Gelder worden gebruikt, evenals een uitdraai van de hoorcollegesheets (dit alles zonder eigen aantekeningen).

Bij de opgaven waar om een algoritme wordt gevraagd, geeft u de pseudocode van uw oplossing en een beknopte maar duidelijke uitleg van de werking. Algoritmes zonder duidelijke uitleg worden niet in beschouwing genomen.

Uitspraken die u doet in antwoord op gestelde vragen moeten nauwkeurig worden beargumenteerd.

Er zijn 5 opgaven, waarmee 90 punten behaald kunnen worden. Het tentamenresultaat is (het aantal behaalde punten gedeeld door 10) plus 1.

Vermeld uw naam en de afkorting ADC op ieder los blad. Vermeld ook de werkcollegeleider waar u dit jaar bij was ingedeeld en geef expliciet aan of u beide huiswerkopgaven gemaakt hebt.

Veel succes!

Opgave 1

15 pt

1. Gegeven een integer array a ter lengte $n = 2^k$ voor een $k > 0$.

   (a) Geef een algoritme dat het minimum en maximum van a bepaalt door het array één keer te doorlopen. Bepaal het aantal vergelijkingen dat dit algoritme uitvoert.

   Uitwerking:

```
(int, int) minmax(int[]a)
{ int mn = a[1], mx = a[1];
  for (i=2; i<=n; i++) {
      mn = min(mn,a[i]);
      mx = max(mx,a[i]); }
  return (mn, mx);
}
```

   De for-loop wordt $n-1$ keer doorlopen; per keer worden 2 vergelijkingen uitgevoerd. Totaal dus $2(n-1)$ vergelijkingen. Reken ook goed als de vergelijking $i <= n$ wordt meegenomen (wat leidt tot $2(n-1)+n$ vergelijkingen).

(b) Beschouw het volgende algoritme voor het bepalen van het minimum en maximum:

```
(int, int) minmax( int a[1], ..., a[n])
{ if (n==2)
   { if (a[1]<=a[2]) return (a[1],a[2]); else return (a[2],a[1])
   }
  else { (mn1,mx1) = minmax(a[1], ..., a[n/2]);
         (mn2,mx2) = minmax(a[n/2+1], ..., a[n]);
         return(min(mn1,mn2), max(mx1,mx2));
       }
}
```

Geef een recurrente betrekking voor het aantal vergelijkingen van dit algoritme en bepaal een exacte uitdrukking voor dit aantal.

Uitwerking:

Inspectie van de code leidt tot de volgende recurrente betrekking:

$$\begin{array}{rcl} T(2) & = & 1 \\ T(n) & = & 2 \cdot T(\frac{n}{2}) + 2 \text{ voor alle } n \geqslant 2 \end{array}$$

Merk op dat de constante kosten 2 de twee vergelijkingen zijn die in het return-statement gemaakt worden.
Voor een exacte uitdrukking analyseren we de recursieboom van $T(n)$ (merk op dat het Master theorema alleen een asymptotische complexiteitsklasse geeft, geen exacte uitdrukking!). De boom is binair, elke knoop heeft kosten 2, bladeren hebben kosten 1. De diepte van de boom is $(\log n) - 1$ en er zijn $\frac{n}{2}$ bladeren (want de bladeren zijn knopen met label $T(2)$). Dus de totale kosten zijn:

$$\sum_{i=0}^{\log n - 2} 2 \cdot 2^i + \frac{n}{2} = 2 \cdot \frac{1 - 2^{\log n - 1}}{1 - 2} + \frac{n}{2} = \frac{3 \cdot n}{2} - 2$$

Reken ook goed als de vergelijking $n == 2$ wordt meegenomen (leidt tot kosten $2n - 3$).

2. Vind de asymptotische orde van de oplossing van de volgende recurrente betrekking:

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

Uitwerking:

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

Opgave 2

20 pt

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 $postorder(-1, N)$.

Opgave 3

20 pt

1. Geef een depth-first search algoritme die kijkt of een gerichte graaf een cykel bevat. De graaf is gegeven als een adjacency list. Wat is de complexiteit van de algoritme?

   Uitwerking:

   We passen het standaard DFS skelet (sheets 26 en 27 van hoorcollege 5) aan door een boolean *cyclefound* waar te maken op het moment dat we ergens in de DFS een edge vinden naar een gele (dus reeds eerder in het pad bezochte) vertex. De DFS wordt gestopt op het moment dat een cykel gevonden is. De worst case complexiteit is (zoals gewoonlijk bij DFS) van de orde $\Theta(|V|+|E|)$, met $|V|$ het aantal vertices en $|E|$ het aantal edges.

   ```
   bool cyclefound;

   bool detectcycle(intlist[] adjV, int n)
   { int[1..n] color;
     cyclefound = false;
     for (int v=1; v<=n; v++) color[v] = blue;
     while v<=n and not cyclefound
     { if color[v]==blue
   ```

```
        cycleDFS(adjV, color, v)
  }
  return cyclefound
}

void cycleDFS(intlist[] ajdV, int[] color, int v)
{ int w; intlist remAdj = adjV[v];
  color[v] = yellow;
  while not remAdj.isEmpty() and not cyclefound
  {w = remAdj.first();
   remAdj = remAdj.rest();
   if color[w]==blue
      then cycleDFS(adjV, color, w)
      else if color[w]== yellow cyclefound = true
  }
  color[v] = green
}
```

2. Een graaf heet 40 vertices en 70 edges. De edges kunnen gewichten hebben ter waarde 1, 2 of 3. Beschouw de som van de edges van een minimal spanning subtree van deze graaf. Welk minimum en welk maximum zou de waarde van deze som mogelijk kunnen aannemen?

   Uitwerking:

   In de opgave had natuurlijk moeten staan: een *connected* graaf. Verder had er moeten staan: spanning tree, in plaats van spanning subtree. Excuses!

   Een connected graaf met 40 vertices heeft een spanning tree van 39 edges. De minimal spanning tree heeft dus minimaal gewicht 39, en maximaal 117.

Opgave 4

25 pt

Beschouw het volgende spel. Het spel wordt gespeeld op een bord met n bij n vierkante vakjes. Je mag een damsteen op een willekeurig vakje op de onderste rij zetten. De damsteen mag je vervolgens steeds diagonaal linksomhoog of rechtsomhoog schuiven, mits je op het bord blijft. Voor elke zet kun je een bepaald positief aantal punten krijgen, die in een tabel gegeven zijn: vanuit vakje (i, j) rechtsomhoog levert $p(i, j, R)$ punten op, linksomhoog levert $p(i, j, L)$ punten op. Uiteindelijk mag je op een willekeurig vakje op de bovenste rij eindigen. Neem aan dat het vakje linksonder de coordinaten $(1, 1)$ heeft.

1. Geef een recurrente betrekking voor het maximaal aantal punten bij aankomst in vakje met coordinaten (i, j).

   Uitwerking:

   We kiezen ervoor dat (i, j) het vakje op kolom i en rij j aangeeft. Stel het maximaal aantal punten in vakje (i, j) is $R(i, j)$. Stel dat $1 \leqslant j \leqslant n$ en $0 \leqslant i \leqslant n+1$, met $R(0, j) = 0$ en $R(n+1, j) = 0$, en alle zetten van en naar vakjes met $i = 0$ of $i = n+1$ leveren 0 punten op. Dan geldt de volgende recurrente betrekking:

   - $R(i, j) =$
     $max\{R(i-1, j-1)+p(i-1, j-1, R), R(i+1, j-1)+p(i+1, j-1, L)\}$
     voor $1 \leqslant i \leqslant n$, $1 < j \leqslant n$
   - $R(i, j) = 0$ voor $i = 0$ of $i = n+1$ of $j = 1$

2. Geef een algoritme om te bepalen hoeveel punten je maximaal kunt winnen. De complexiteit mag niet slechter zijn dan kwadratisch in n.

   Uitwerking:

   We passen dynamisch programmeren toe. Uiteindelijk moeten we het bovenste vakje vinden met het meeste aantal punten.

```
int maxpoints(int[] c, int n)
{ int[0..n+1,1..n] R;
  for(j=1; j<=n; j++) {R[0,j] = 0;
                       R[n+1,j] = 0};
  for(i=1; i<=n; i++) R[i,1] = 0;

  for(j=2; j<=n; j++)
   for(i=1; i<=n; i++)
          R[i,j] = max(R[i-1,j-1]+p[i-1,j-1,R],
                       R[i+1,j-1]+p[i+1,j-1,L]);

  mx = R[1,n];
  for (i=2; i<=n; i++) max = max(mx, R[i,n]);

  return mx
}
```

   De complexiteit van dit algoritme is $\Theta(n^2)$.

3. Geef aan hoe het algoritme zo aangepast kan worden dat ook het beginvak en de winnende zettenreeks geproduceerd kunnen worden.

   Uitwerking:

Hou in een array `move` bij hoe je in vakje (i, j) gekomen bent: is dat vanuit $(i-1, j-1)$, dan wordt `move[i,j]` gelijk aan `right`, is dat vanuit $(i+1, j-1)$, dan `move[i,j] = left`. Uiteindelijk kun je dus met `move` een pad volgen vanuit het maximale vak in de bovenste rij naar de onderste rij; in omgekeerde richting is dat het optimale pad.

Opgave 5

10 pt

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

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

   Uitwerking:
   Onwaar. $b = 2$, $c = 2$, dus $E = 1$, nu geldt $log\ n \in O(n^{1-\epsilon})$ (geval 1) dus $T(n) \in \Theta(n)$.

2. Stel je gebruikt gesloten hashing. Als de gemiddelde lijstlengte 5 is, en er 5000 elementen zijn, is de loadfactor kleiner dan 1.

   Uitwerking:
   Ik bedoelde: hashing met gesloten adressering, en dan is het antwoord onwaar: de loadfactor is gelijk aan de gemiddelde lijstlengte (het aantal elementen doet er niet toe), dus 5.
   Als de vraag is opgevat als open adressering (de echte betekenis van gesloten hashing), is er geen goed antwoord mogelijk; reken de vraag in dat geval goed.

3. Als je het travelling salesman probleem polynomiaal zou kunnen reduceren tot een ander probleem, heb je bewezen dat P = NP.

   Uitwerking:
   Onwaar. Dat zou alleen gelden als je TSP polynomiaal zou kunnen reduceren tot een probleem in P. Maar als je TSP polynomiaal reduceert tot bv. een probleem in NPC geldt niet dat je kunt concluderen dat P = NP.

4. Als we zouden kunnen bewijzen dat P ongelijk is aan NP, dan hadden we tevens bewezen dat problemen in NP niet efficient opgelost kunnen worden.

   Uitwerking:
   De vraag is niet duidelijk. Als hij opgevat wordt als "dat alle problemen in NP niet efficient opgelost kunnen worden" is het antwoord onwaar: de problemen in P zitten ook in NP, en deze problemen kunnen nog steeds efficient opgelost worden. Als hij opgevat wordt als "dat sommige problemen in NP niet efficient opgelost kunnen worden" is het antwoord waar: problemen in NPC kunnen niet efficient opgelost worden.