Vakcode
Datastructuren 2

Paul Sohier
0806122

Kees Jan Simon
0791114

16 januari 2012

Inhoudsopgave

Hoofdstuk 1

De efficiëntie van programmatuur

1.1 Opdracht 1

Bepaal de herhalingsfrequentie $T(n)$ van:

- `for(i=n-1; i<n; i++){}`
- `for(i=n-1; i <n2̂; i++){}`
- `for(i=n; i<n; i++){}`
- `i=0; while(i < n) {a[i] = 0}`

`Antwoord:`

- $T(1)$
- $T(n^2 - (n-1))$
- $T(0)$
- $T(\infty)v0keer$

1.2 Opdracht 2

Bepaal de O-notatie van:

- $T(n) = 17n^3 - 13n^2 + 10n + 2000$
- $T(N) = 3^n - 13n$
- $T(n) = 20log_2 n + n^2$

`antwoord:`

- $T(n) = 0(n^3)$

- $T(n) = 0(n^2)$
- $T(n) = 0(n^2)$

1.3 Opdracht 3

In het sorteeralgoritme *SelectionSort* worden de elementen in een lijst $a[1 \ldots n]$ verwisseld van plaats afhankelijk van het onderlinge verschil in waarde. Het algoritme begint bij het eerste element te verwisselen met het kleinste element in de rest van de lijst. Vervolgens wordt het tweede elelment verwisseld met het ena kleinste element in de lijst, etc.:

```
1  void SelectionSort(lijst a){
2  int i,j,k;
3  i = 0;
4  while(i < n){
5    j = ++i;
6    k = j;
7    while (j <= n){
8      if (a[j]<a[k]) k = j;
9      ++j;
10   }
11   verwissel(a; i; k);
12 }
13 }
```

Het soorteeralgoritme *InsertionSort* werkt de lijst $a[1 \ldots n]$ door, het eerste stuk $(1 \ldots i)$ is gesorteerd, het tweede stuk (i+1 $\ldots$ n is nog ongesorteerd. Elk element i+1 uit het ongesorteerde lijstgedeelte wordt in het gesorteerde gedeelte tussengevoegd, waarna het gesorteerde gedeelte van de lijst met één element is toegenomen, ten koste vand e ongesorteerde deellijst:

```
1  void InsertSort(lijst a){
2  int i, j, ready;
3  i = 1;
4  while(i<n){
5    j = i++;
6    ready = 0;
7    while((j >=1)&&(ready==0)){
8      if (a[j+1]<a[j]){
9        verwissel(a;j+1;j);
10       --j;
11     }
12     ready=1;
13   }
14 }
```

Bepaal van *InsertionSort* en *SelectionSort* de efficiëntie in P-notatie.

`Antwoord:`

SelectionSort: $O(n^2)$ InsertionSort: $O(n^2)$

1.4 Opdracht 4

BubbleSort maakt gebruik van herhaald verwisselen van buurelementen in een lijst. Een element wordt naar voren verplaatst indien het kleiner is dan het buurelement. Geef de tijdcomplexiteit

in O-notatie van het *BubbleSort* algoritme. Is dit slechter dan de complexiteit van *SelectionSort* en *InsertionSort*?

`antwoord:`

BubbleSort: $O(n^2)$

1.5 Opdracht 5

Bepaal de tijdcomplexitijd in de O-Notatie van een algoritme om alle permutaties van n verschillende voorwerpen te genereren.

`antwoord:`

$O(n!)$

1.6 Opdracht 6

Bepaal de tijdcomplexiteit in O-Notatie om de determinant van een nxn matrix te bepalen.

`antwoord`

$O(n^2)$, de formule voor het berekenen is: $\det(a) = \sum_{\sigma \in S_n} \prod_{i=1}^{n} A_{i,\sigma(i)}$, wat dus een dubbel geneste forloop is.

1.7 Opdracht 7

Een *polynoom* $f(x) \sum_{n}^{i=0} a_i x^j$ $(a0 \ldots a_n)$ zijn coëfficiënten) wordt meestal uitgeschreven als:

$$f(x) = a_n x^n + a_{n-1} x^{n-1} + a_{n-2} x^{n-2} + \ldots + a_1 x^1 + a0$$

Wij kunnen de polynoon $f(x)$ herschrijven met de *regel van Horner*:

$$f(x) = ((\ldots((a_n x + a_{n-a})x + a_{n-2})x \ldots)x + a1)x + a0$$

`antwoord:`

De orginele notatie is $n!$ vermenigvuldigingen, n optellingen. Bij de verbetering is het n vermenigvuldigingen en n optellingen.

1.8 Opdracht 9

Het is bekend dat twee priemgetallen redelijk snel tot een product vermenigvuldigd kunnen worden. Het omgekeerde, het ontbinden van zo'n product in twee priemfactoren, kost meer moeite. In verband met een redelijk werkverdeling tussen docent en studenten zijn docenten dol op dit soort problemen. Wij nemen problemen, die door de docent snel geconstrueerd kunnen worden en voor de studenten lastig op te lossen zijn, in de klasse Ideale-problemen op.

Geef enkele voorbeelden uit de klasse Ideale-problemen. Is er, naast het onderwijs, nog een een gebeid waar Ideale-problemen kunnen worden toegepast?

`antwoord:`

- Een lastige willekeurge formule opschrijven -¿ ga maar differentieren of intergreren.
- Los software probleem X op, bijvoorbeeld het handelsreizigersprobleem -¿ algoritmiek bedenken. Dinning philosophers).

Voor het berekenen van een unieke code om de echtheid van bijvoorbeeld een bepaald bestand te controleren. Het ICT-gebied dus.

Hoofdstuk 2

Recursie

2.1 Opdracht 1

Leidt de recurrente betrekking af voor het aantal verbindingslijnstukken tussen n punten.

`antwoord:`

Iedere $n$de stap levert $n-1$ extra lijnen op bij het geheel, dus: $T(n) = T(n-1) + n - 1$.

2.2 Opdracht 2

De volgende C-functie berekent de faculteit van een natuurlijk getal $n \geq 0$:

```
1  long faculteit (int n)
2  {
3    if (n == 0)
4    {
5      return 1;
6    }
7    else
8    {
9      return n*faculteit(n-1);
10   }
11 }
```

Maak een iteratieve versie van deze functie.

`antwoord:`

```
1  long int faculteit(int n)
2  {
3     if (n == 0)
4     {
5        return 1;
6     }
7     else if (n > 0)
8     {
9        int i;
10       for(i = n-1;  i > 1; i--)
11       {
```

```
12            n = n*i;
13        }
14        return n;
15    }
16    else
17    {
18        return -1;
19    }
20 }
```

2.3 Opdracht 3

Maak een recursief algortime voor de torens van Hanoi.

`antwoord:`

```
1  int hanoi()
2  {
3     if(n > 0)
4     {
5       return hanoi((n-1) + hanoi(n-1) + 1);
6     }
7     else
8     {
9       return n;
10    }
11 }
```

2.4 Opdracht 4

Bepaal het aantal stappen T_n voor het verplaatsen van n schijven bij de torens van Hanoi, indien rechtstreekse verplaatsingen van toren A naar toren C verboden zijn. Elke schijf moet langs toren B.

`antwoord:`

```
1  int hanoi(int n)
2  {
3     if(n == 1) {
4       return 2;
5     }
6     else
7     {
8       return hanoi((n-1) + 2*pow(2,n) - 1);
9     }
10 }
```

2.5 Opdracht 5

Bereken $alg\_a(n)$ en $alg\_b(n)$ voor $n = 1 \ldots 5$. Bereken de efficiëntie van algoritme alg_a en van algoritme alg_b in O-notatie:

(a)

```
1 alg_a(n):resultaat
2 if n > 1  then
3 return (alg_a(n-1)+alg_a(n-1))
4 else
5 return (1)
```

```
1 alg_b(n): resultaat
2 if n > 1 then
3 return 2 * alg_b(n-1))
4 else
5 return 1
```

antwoord:

$n = 1 \ldots 5 -> \{1, 2, 4, 8, 16\}$ voor beide algoritmes. Is het niet gewoon simpeler, dit: $O(n)$ per functieaanroep. twee keer per aanroep, recursief dus $O(2^n)$, één keer per aanroep, resultaat vermenigvuldigen, dus $2 \cdot O(n) = O(n)$

2.6 Opdracht 6

Leidt een recurrente betrekking af voor een vermenigvuldiging van twee n-bit getallen x en y. Maak hiervan een recursief algoritme. Bepaal tevens de tijdcomplexiteit van dit algoritme. Wij mogen veronderstellen dat voor het berokken proccessortype, het vermenigvuldigen en delen $O(n^2)$ instructies zijn. Daarentegen zijn optellen, aftrekken en 1-bits schuifacties $O(n)$ instructies. Het testen of een natuurlijk getal even-of oneneven is, zoals alle overige instructies atomair $O(1)$

antwoord:

```
1  int mul(int x, int y)
2  {
3      if (y == 1)
4      {
5          return x;
6      }
7      else if ((y % 2) == 0)
8      {
9          x = x << 1;
10         return mul(x, (y >> 1));
11     }
12     else
13     {
14         return x + mul(x, (y - 1));
15     }
16 }
```

Geen vermenigvuldigingen, alleen optellingen, dus $O(n)$

2.7 Opdracht 7

Leidt een recurrente betrekking af voor de berekening van een x^p, waarbij x een reël getal en p een natuurlijk getal van n bits. Maak hiervan een recursief algoritme. Bepaal de tijdcomplexiteit voor dit algoritme op dezelfde processortype als die uit de vorige opgave.

antwoord:

```
1  int pow(int x, int y)
2  {
3      if(y == 0)
4      {
5          return 1;
6      }
7      else if (y == 1)
8      {
9          return x;
10     }
11     else if ((y%2) == 0)
12     { //even getal
13         int t = pow(x, (y >> 1));
14         return mul(t,t);
15     }
16     else
17     { //oneven
18         return mul(x, pow(x, (y-1)));
19     }
20 }
```

$O(n)$? We hebben alleen optellen/aftrekken/bitshifts zijn $O(n)$, compares zijn $O(1)$

Hoofdstuk 3

De grafentheorie

3.1 Opdracht 1

Een probleem, dat voor het eerst geformuleerd werd door een Chinese wiskundige, luidt: Een *Chinese postbode* moet lopend de post bezorgen. Langs elke weg (een tak met een positieve afstandswaarde) staan brievenbussen. De optimale route heeft de minimale afstand. In wele type grafen is een optimale oplossing aanwezig?

`antwoord:`

Euler: Een zogeheten Euler-wnandeling of zelfs een dergelijke wandeling te maken zodat deze begint en eindigt in dezelfde knoop (Euler-cykel).

3.2 Opdracht 2

Een handelsreiziger moet vanaf een basis een aantal steden bezoeken met de kortste reisafstand en daarna terugkeren op zijn thuisbasis. In welk type grafen is een optimale oplossing aanwezig?

`antwoord:`

Ongericht gewogen grafen.

3.3 Opdracht 3

Kan de volgende graaf zonder snijdende lijnen getekend worden op een plat vlak?

`antwoord:`

Ja.

3.4 Opdracht 4

Wat is het minimaal aantal kleuren dat nodig is om de knopen van de graaf uit vraagstruk 3 te kleuren zodanig dat adjacente knopen niet dezelfde kleur hebben?

`antwoord:`

4 kleuren.

3.5 Opdracht 5

Verzin een praktische voorbeeld van een knopgewogen graaf.

`antwoord:`

Steden met het aantal inwoners, een stad met meer inwoners is van meer betekenis voor een bedrijf om daar een vestiging te openen.

3.6 Opdracht 6

In een graaf zijn vaak meer dan ëën opspannende boom te vinden. Bepaal van de graaf uit vraagstuk 4 het aantal opspannende bomen.

`antwoord:`

16, 5 mogelijkheden voor het pad van een boom, met steeds vier start mogelijkheden. Daarna nog 4 eraf, omdat twee bomen maar 2 mogelijkheden hebben in plaats van 4. In totaal dus 16.

3.7 Opdracht 7

Bewijs dat een opspannende boom in een samenhangende graaf met n knopen en m takken uit $n - 1$ takken bestaat.

`antwoord:` 2 startpunten betekend 1 pad. Drie knooppunten is 1 pad meer. Kortom iedere extra node is een extra pad.