Vakcode
Datastructuren 2

Sebastiaan Polderman
0820738

Paul Sohier
0806122

18 mei 2011

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 <n2; i++){}`
- `for(i=n; i<n; i++){}`
- `i=0; while(i < n) {a[i] = 0}`

1.2 Opdracht 2

Bepaal de O-notatie van:

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

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.

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*?

1.5 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*:

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

Hoofdstuk 2

Recursie

2.1 Opdracht 1

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

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.

2.3 Opdracht 3

Maak een recursief algortime voor de torens van Hanoi.

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.

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
```

2.6 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.

2.7 Opdracht 11

Een ingewikkelde vorm van recursie is de functie van *Ackermann*:

$$ack(m,n) = \begin{cases} n+1 & alsm = 0enn \geq 0 \\ ack(m-1,1) & alsm < 0enn = 0 \\ Ack(m-1, ack(m,n-1)) & alsm > 0enn > 0 \end{cases}$$

Toon aan dat $Ack(2,3) = 9$.

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?

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?

3.3 Opdracht 3

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

3.4 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.

3.5 Opdracht 7

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

3.6 Opdracht 9

Wat stelt de rij en de kolomsom van een adjaceentiematrix van een gerichte graaf voor?

Hoofdstuk 4

Graafalgoritmen

4.1 Opdracht 1

Een touringcarbedrijf wil een lucratieve toeritische route uitzetten langs een maximale opspannende boom in een netwerk. Geef een algoritme voor een maximum opspannende boom.