De uitvoeringstijd v/e algoritme

Algoritme voor berekenen v/d uitvoeringstijd

1) Lagere orde termen weglaten
2) Constante termen gelijkstellen aan 1

Som v/d eerste n getallen

Kleine Gauss :

$$\begin{array}{rccccccccc} & 1 & + & 2 & + & 3 & + & \dots & + & n \\ + & n & + & n-1 & + & n-2 & + & \dots & + & 1 \\ \hline & n+1 & + & n+1 & + & n+1 & + & \dots & + & n+1 \end{array}$$

$$\Rightarrow \frac{n(n+1)}{2}$$

Som van de eerste n machten van a

$$\begin{array}{rcccccccc} S_n = & 1 & + & a^1 & + & a^2 & + & \dots & + a^n \\ a\,S_n = & a & + & a^2 & + & a^3 & + & \dots & + a^{n+1} \\ \hline (a-1)S_n = & a^{n+1} & - & 1 & & & & & \end{array}$$

$$\Rightarrow \frac{a^{n+1}-1}{a-1}$$

Som van de eerste n kwadraten

$$\sum_{i=1}^{n} i^2 = \frac{1}{6}n(n+1)(2n+1)$$

Recursie

Bewijs door inductie

1) Bewijs de basisstap
2) Bewijs de inductiestap
   1) Stel de inductiehypothese op
   2) Gebruik de definitie
   3) Gebruik de inductiehypothese

Som v/d eerste n machten van a

$$\underbrace{1 + a + a^2 + \ldots + a^n}_{S_n} = \frac{a^{n+1}-1}{a-1}$$

1) Basisstap voor $n = 0$

$LL = 1$

$RL = \frac{a^{0+1}-1}{a-1} = 1$ } ok

2) Inductiestap

IH: Stel dat $S_m = \frac{a^{m+1}-1}{a-1}$ $m \leq 1$

$$S_{n+1} = 1 + a + a^2 + \ldots + a^n + a^{n+1}$$

$$= S_n + a^{n+1}$$

$$\overset{IH}{=} \frac{a^{n+1}-1}{a-1} + \frac{a^{n+1}(a-1)}{a-1}$$

$$S_{n+1} = \frac{a^{(n+1)+1}-1}{a-1}$$

$$= \frac{a^{n+1} - 1 + a^{n+2} - a^{n+1}}{a-1}$$

$$= \frac{a^{n+2}-1}{a-1}$$

■

Som van de eerste n kwadraten

$$\underbrace{1^2 + 2^2 + 3^2 + \ldots + n^2}_{S_n} = \frac{n(n+1)(2n+1)}{6}$$

1) Basisstap voor $n = 1$

LL: $1^2 = 1$

RL: $\frac{1.(1+1)(1+2)}{6} = \frac{6}{6} = 1$ } OK

2) Inductiestap

IH: Stel $S_m = \frac{m(m+1)(2m+1)}{6}$ als $m \leq n$

$$\begin{aligned} S_{n+1} &= 1^2 + 2^2 + 3^2 + \ldots + n^2 + (n+1)^2 \\ &\overset{IH}{=} S_n + (n+1)^2 \\ &= \frac{n(n+1)(2n+1)}{6} + (n+1)^2 \\ &= \frac{1}{6}(n+1)(n(2n+1)+6(n+1)) \\ &= \frac{1}{6}(n+1)(2n^2+7n+6) \\ &= \frac{1}{6}(n+1)(n+2)(2n+3) \end{aligned}$$

$S_{n+1} = \frac{(n+1)(n+2)(2n+3)}{6}$

Zoekalgoritmes

Sequentieel of lineair zoeken

De elementen 1 voor 1 overlopen en kijken of het gezochte element voorkom

Best: $\Theta(1)$ Slechtst: $\Theta(n)$ Gemiddeld: $\Theta(n)$

Binair zoeken

werkt enkel bij gesorteerde array's (telefoonboek zoeken)
De array wordt steeds in 2 gedeeld en dan word op dezelfde manier in de gehalveerde array gezocht.

Gemiddeld: $\Theta(\log n)$

Sorteeralgoritmes

Selection sort

algoritme :
1) zoek het grootste element in het niet gesorteerde deel
2) wissel het grootste element met het laatste element van he niet gesorteerde deel
3) herhaal 1 en 2 tot er enkel een gesorteerd deel over is

Gemiddelde : $\Theta(n^2)$

Insertion of card sort

algoritme :
1) neem het volgende te sorteren element
2) voeg het element op de juiste plaats toe in het gesorteer deel
3) herhaal 1 en 2 tot alle elementen gesorteerd zijn.

Gemiddelde : $\Theta(n^2)$ Best : $\Theta(n)$ Slechtst: $\Theta(n^2$

Insertion sort zal in het geval van een gesorteerde rij sneller zijn dan sele sort omdat niet de gansc array moet overlopen worden.

Mergesort

algoritme :
1) splits de array in 2
2) herhaal 1 op beide nieuwe array's tot er enkel nog array's over zijn met 1 element
3) merge de arrays

Dit algoritme wordt recursief toegepast.

Gemiddelde: $\Theta(n \log(n))$

Mergesort is tijdsefficiënt, maar niet geheugen efficiënt

Quicksort

Keuze v/d spil

Passief: middelste element is de spil
Actief: mediaan van eerste, middelste en laatste element is de spil

Als een element gelijk is aan de spil moet men stoppen

Partitioneren

1) Bij passief: spil wordt verwisseld met laatste element
   Bij actief: spil wordt verwisseld met voorlaatste element
2) Alle elementen kleiner dan de spil worden links in de array geplaatst, alle elem
   groter dan de spil rechts.
   1) Zoek van links naar rechts een element groter dan de spil
   2) Zoek van rechts naar links een element kleiner dan de spil
   3) Als er links een groot element en rechts klein element worden deze van plaats gewisseld
   4) Herhaal zolang de posities niet kruisen
3) Het eerste grote element wordt gewisseld met de spil

Kleine array's

Indien de deelrijen voldoende klein zijn is het interessant om over te gaan op insertion sort. Het aantal elementen waarbij er wordt overgegaan op cardsor i.p.v. recursief verder te gaan wordt de cut-off genoemd (bv 5)

Tijdscomplexiteit

Gemiddelde: $\Theta(n \log n)$ Beste: $\Theta(n \log n)$ Slechtste: $\Theta(n^2)$

Algoritme

recursief partitioneren

Standaard datastructuren

Stack

LIFO - structuur

UML

Stack
- data : array [] van Element - t : geheel getal
+ Stack() + empty() : boolean + push(x : Element) : / + pop() : Element + peek() : Element

Controle van haakjes

problemen : 1) niet correct genest
2) niet alle haakjes gesloten
3) te veel sluitende haakjes

methode : 1) openend haakje op stack plaatsen
2) sluitend haakje vergelijken met bovenste op de stack

Postfix en infix

Bepalen van de waarde van een postfix-uitdrukking

1) Uitdrukking van links naar rechts overlopen
2) Operanden op de stapel plaatsen
3) Als er een operator ontmoet wordt, wordt deze toegepast op de 2 bovenste elementen v/d stapel
4) Herhaal tot de volledige uitdrukking is doorlopen
5) Als de uitdrukking is doorlopen staat nog 1 element op de stapel : de oplossing

Van infix naar postfix

1) Uitdrukking van links naar rechts overlopen
2) Als operand dan rechtstreeks uitschrijven
3) Als operator of haakje
    1) Op stack plaatsen als :
        1) stack is leeg
        2) operator heeft hogere prioriteit dan bovenste op de stack
        3) openingshaakje
    2) alle operatoren met lagere en gelijke prioriteit v/d stack halen en uitschrijven en ingelezen operator op stack zetten
4) Als einde van de uitdrukking is bereikt, dan alle elementen van stack halen en uitschrijven.

Queue

FIFO - structuur

UML

Queue
- data : array [] van Element - k : geheel getal - s : geheel getal
+ Queue (n: geheel getal) + empty ( ) : boolean + enqueue (x: Element : / + dequeue ( ) : Element + peek ( ) : Element

Lijsten

UML

List
- data : array [] van Element - aantal: geheel getal
+ List (n: geheel getal) + empty ( ) : boolean + omvang ( ) : geheel getal + geefElem (p: geheel getal) : Element + geefPositie (x: Element) : geheel getal + verwijderElem (p: geheel getal) : Element + invoegenVoor (p: geheel getal, x: Element) : / + invoegenNa (p: geheel getal, x: Element) : / + vervang (p: geheel getal, x: Element) : /

Gelinkte lijsten

UML

Knoop
- data : Element
- volgende : Knoop
+ Knoop()

Lijst
- eerste : Knoop
+ zoek(x: Element): Knoop
+ verwijder(ref: Knoop): Element
+ voegToe(ref: Knoop): /

Anker

Als er geen anker gebruikt wordt, bestaan de volgende problemen:

1) element toevoegen aan een lege lijst
2) element vooraan aan de lijst toevoegen
3) 1e element v/e lijst verwijderen

Met een ankercomponent zijn deze problemen opgelost

Dubbel gelinkte lijst

- De knoop kent zowel de volgende als de vorige
- De lijst kent zowel de eerste als de laatste knoop

Hashtabellen

array met grootte N

- voor N kiest men best een priemgetal ⇒ betere verspreiding over de bucke

n : aantal hashcodes

- ideaal n niet veel groter dan N

loadfactor : $\frac{\#\text{elementen}}{N} \approx 0{,}75$