Achtste college complexiteit

18 maart 2008

Opgave 37/38 & Shellsort

a. Hoe werkt *Quicksort* op het rijtje 5 5 5 5 5 5 5 5 ? Hoeveel vergelijkingen en hoeveel verwisselingen worden gedaan?

b. Laat $n = 2^k$. Hoeveel vergelijkingen doet Quicksort voor een rijtje van n dezelfde getallen? En hoeveel verwisselingen?

c. Is Quicksort stabiel?

d. Laat n even zijn. Hoeveel vergelijkingen doet Quicksort op het rijtje $n, n-1, \ldots, 2, 1$? En hoeveel verwisselingen? Hoeveel verwisselingen worden gedaan op het reeds gesorteerde rijtje $1, 2, \ldots, n-1, n$?

a. Zoals bekend doet *Quicksort* $\frac{1}{2}n(n+3)-2 = \Theta(n^2)$ vergelijkingen op reeds gesorteerde rijtjes zoals $1, 2, 3, \ldots, n$. Bekijk wat er gebeurt met het gesorteerde rijtje als nu niet steeds het eerste array-element $A[p]$ wordt gekozen als pivot, maar de mediaan van $A[p]$, $A[r]$ en $A[\lceil \frac{p+r}{2} \rceil]$.

b. Neem aan dat we een rijtje met n verschillende getallen hebben. Hoe wordt het rijtje opgesplitst door Partitie als de pivot X het i-de element in grootte is (van klein naar groot: het 1-e element in grootte is de kleinste, het n-de in grootte de grootste)? M.a.w. wat is de q die Partitie oplevert?

c. Laat $n = 10$. We bekijken weer de "gewone" versie van Quicksort, dus waarbij als pivot steeds het eerste array-element wordt gekozen. Hoeveel vergelijkingen doet Quicksort als Partitie om en om de slechtste en de beste splitsing oplevert? Neem aan dat Partitie in elke stap $m + 1$ vergelijkingen doet voor een (deel)rijtje van m elementen. Vergelijk dit aantal met het aantal vergelijkingen dat gedaan wordt voor het gesorteerde rijtje. Idem met het aantal vergelijkingen dat Quicksort doet voor het rijtje waarbij in elke stap de (deel)rij in twee ongeveer gelijke stukken wordt verdeeld (best case). Ligt het aantal vergelijkingen dichter bij de best case of bij de worst case?

d. Maak aannemelijk dat een invoerrijtje waarbij afwisselend slecht en goed wordt gesplitst (zie **c.**) in orde van grootte evenveel vergelijkingen doet als het rijtje waarbij in elke ronde in twee gelijke delen wordt gesplitst ($\Theta(n \lg n)$ dus). Gebruik hiertoe dat in het eerste geval in *twee* rondes drie deelrijtjes zijn "ontstaan" van resp. 1, $\frac{m-1}{2}$ en $\frac{m-1}{2}$ (geval m is oneven) elementen, terwijl we in het tweede geval na *een* ronde een rijtje ter lengte $\frac{m-1}{2}$ en een rijtje ter lengte $\frac{m+1}{2}$ hebben. Hierin is m de lengte van het (deel)rijtje waarop we Quicksort (dus Partitie) loslaten.

Een en ander suggereert dat Quicksort het meestal goed ($= \Theta(n \lg n)$) doet: het gemiddelde zal "dus" rond $\Theta(n \lg n)$ liggen.

Shellsort* was een van de eerste sorteeralgoritmen met worst case complexiteit minder dan kwadratisch.

Shellsort sorteert in elke ronde deelrijtjes. In het begin veel korte rijtjes, waarbij de elementen uit een rijtje ver van elkaar liggen. Later weinig lange rijtjes, waarbij de elementen uit een rijtje dicht bij elkaar liggen. De rij wordt zo als het ware voorgesorteerd. In de laatste ronde wordt de rij dan als geheel gesorteerd. Shellsort sorteert met behulp van vergelijk-verwissel / compare-exchange (indien nodig) operaties.

*Donald Shell, 1959

Definitie

Een rij $A[1], A[2], \ldots, A[n]$ heet k-gesorteerd als geldt: $A[i] \leq A[i+k]$ voor elke $i = 1, 2, \ldots, n-k$.

Voorbeeld: Het rijtje

$$5, 8, 24, 13, 7, 18, 31, 19, 44, 63, 82, 29$$

is 4-gesorteerd (en overigens ook 6-gesorteerd).

Merk op: 1-gesorteerd = gesorteerd.

Shellsort gebruikt een rijtje stapgroottes (increments) $h_t, h_{t-1}, \ldots, h_2, h_1 = 1$. De rij A met n elementen wordt gesorteerd door achtereenvolgens subrijen te sorteren van elementen die telkens op afstand h_i van elkaar liggen. Met andere woorden: A wordt h_i-gesorteerd voor $i = t, \ldots, 1$. Aangezien $h_1 = 1$ sorteert Shellsort correct.

Merk op: bij het k-sorteren worden k deelrijtjes van elk maximaal $\lceil \frac{n}{k} \rceil$ elementen gesorteerd.

$h_3 = 6, h_2 = 3, h_1 = 1, n = 13$

81, 94, 11, 96, 12, 35, 17, 95, 28, 58, 41, 75, 15

↓ 6-sorteren

15, 94, 11, 58, 12, 35, 17, 95, 28, 96, 41, 75, 81

↓ 3-sorteren

15, 12, 11, 17, 41, 28, 58, 94, 35, 81, 95, 75, 96

↓ 1-sorteren

11, 12, 15, 17, 28, 35, 41, 58, 75, 81, 94, 95, 96

$h_3 = 6, h_2 = 3, h_1 = 1, n = 13$

81, 94, 11, 96, 12, 35, 17, 95, 28, 58, 41, 75, 15

↓ 6-sorteren

15, 94, 11, 58, 12, 35, 17, 95, 28, 96, 41, 75, 81

↓ 3-sorteren

15, 12, 11, 17, 41, 28, 58, 94, 35, 81, 95, 75, 96

↓ 1-sorteren

11, 12, 15, 17, 28, 35, 41, 58, 75, 81, 94, 95, 96

Insertion sort op het voorbeeldrijtje: 52 vergelijkingen.

Shellsort met Insertion sort: 44 vergelijkingen:

$81, 94, 11, 96, 12, 35, 17, 95, 28, 58, 41, 75, 15$

\# inversies $= 41$ $\downarrow$ 6-sorteren: 8 vergelijkingen

$15, 94, 11, 58, 12, 35, 17, 95, 28, 96, 41, 75, 81$

\# inversies $= 25$ $\downarrow$ 3-sorteren: 15 vergelijkingen

$15, 12, 11, 17, 41, 28, 58, 94, 35, 81, 95, 75, 96$

\# inversies $= 11$ $\downarrow$ 1-sorteren: 21 vergelijkingen

$11, 12, 15, 17, 28, 35, 41, 58, 75, 81, 94, 95, 96$

```
(1)   h = n div 2; // h_t = ⌊n/2⌋ dus
(2)   while h > 0 do
(3)       for i := h + 1 to n do
(4)           temp := A[i]; j := i;
(5)           while j − h > 0 do
              // invoegen op de juiste plek
(6)               if temp < A[j − h] then
(7)                   A[j] := A[j − h]; // schuif
(8)                   j := j − h;
(9)               else
(10)                  "exit binnenste while";
(11)              fi
(12)          od
(13)          A[j] := temp; // zet neer
(14)      od
          // de rij is nu h-gesorteerd
(15)      h := h div 2; // oorspronkelijke keuze van Shell: h_i = ⌊h_{i+1}/2⌋
(16)  od
```

Regel (4) t/m (13) is Insertion sort op deelarrays.

Een zeer belangrijke eigenschap van Shellsort is de volgende (zonder bewijs).
Stelling. Als een ℓ-gesorteerd array h-gesorteerd wordt met behulp van compare-exchanges, dan blijft het ℓ-gesorteerd.

Voorbeeld met $n = 12$ en incrementrijtje $6, 4, 3, 2, 1$:

$7, 19, 24, 13, 31, 8, 82, 18, 44, 63, 5, 29$	
$\downarrow$ 6-sorteren	
$7, 18, 24, 13, 5, 8, 82, 19, 44, 63, 31, 29$	6-gesorteerd
$\downarrow$ 4-sorteren	
$5, 8, 24, 13, 7, 18, 31, 19, 44, 63, 82, 29$	4-, 6-gesorteerd
$\downarrow$ 3-sorteren	
$5, 7, 18, 13, 8, 24, 31, 19, 29, 63, 82, 44$	3-, 4-, 6-gesorteerd
$\downarrow$ 2-sorteren	
$5, 7, 8, 13, 18, 19, 29, 24, 31, 44, 82, 63$	2-, 3-, 4-, 6-gesorteerd
$\downarrow$ 1-sorteren	
$5, 7, 8, 13, 18, 19, 24, 29, 31, 44, 63, 82$	1-, 2-, 3-, 4-, 6-gesorteerd

De complexiteit van Shellsort (= aantal arrayvergelijkingen) hangt in hoge mate af van de gekozen stapgroottes. De analyse is in het algemeen extreem moeilijk en nog zeer incompleet.

1. Stapgroottes: $h_t = \lfloor \frac{n}{2} \rfloor$, $h_i = \lfloor \frac{h_{i+1}}{2} \rfloor$ voor $i = t-1, \ldots, 1$. Dan $t = \lfloor \lg n \rfloor$. Voor het gemak nemen we $n = 2^k$, dus $t = k$.

   Stelling A. Het aantal arrayvergelijkingen dat Shellsort met deze incrementserie doet is in de worst case $\Omega(n^2)$.

   Bij het bewijs. Het is voldoende om een bad case aan te geven waarvoor het aantal vergelijkingen $\Omega(n^2)$ is.