Vijfde college complexiteit

7 maart 2025

Selectie is $O(n)$

Insertion sort

Recurrente betrekkingen

Probleem

Gegeven een array $A = A[1], A[2], \ldots, A[n]$ met n verschillende getallen. Gegeven is verder een geheel getal k met $1 \leq k \leq n$. Gevraagd de $A[i]$ die groter is dan precies $k-1$ andere $A[j]$'s. M.a.w.: we zoeken de k-de in grootte (in volgorde van klein naar groot).

Complexiteit

Het selectieprobleem is $O(n)$.

We bewijzen dit door een algoritme te geven dat de k-de in grootte vindt in $O(n)$ vergelijkingen in de worst case*.

*Blum, Floyd, Pratt, Rivest & Tarjan, 1973

Algoritme:

1. Verdeel de getallen in $\lfloor \frac{n}{5} \rfloor$ groepjes van 5* elementen, en één groepje met de resterende n mod 5.

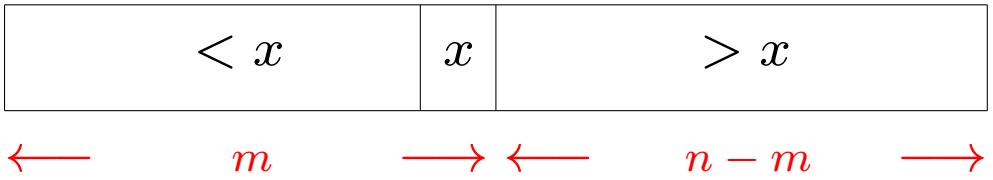
2. Vind de mediaan van elk van de $\lceil \frac{n}{5} \rceil$ groepjes (van maximaal 5), bijvoorbeeld via Bubblesort of opgave 24.

3. Vind de mediaan x van de in stap 2 gevonden $\lceil \frac{n}{5} \rceil$ medianen: recursie†.

*deze 5 is niet zomaar willekeurig gekozen, maar is optimaal

†de mediaan van ℓ elementen is de $\lceil \frac{\ell}{2} \rceil$-de in grootte.

4. Partitioneer (ongeveer zoals bij Quicksort) alle elementen rond x. Stel dat m getallen $\leq x$ zijn en $n-m$ getallen $> x$.



5. Vind de k-de in grootte uit m stuks als $k \leq m$, of de $(k-m)$-de uit $n-m$ als $k > m$: recursie.

Na stap 3. van het algoritme geldt:

- ten minste $3 \cdot (\lceil \frac{1}{2} \lceil \frac{n}{5} \rceil \rceil - 2) \geq \frac{3n}{10} - 6$ elementen zijn groter (resp. kleiner) dan x
- er zijn dus hooguit $\frac{7n}{10} + 6$ elementen $\leq x$ (resp. $\geq x$) (*).

Gevolg: in stap 5. wordt het algoritme recursief aangeroepen op hooguit $\frac{7n}{10} + 6$ elementen (*).

(*) om preciezer te zijn: $\lceil \frac{7n}{10} \rceil + 6$

Zie opgave 30 voor een voorbeeld.

12	15	11	2	9	5	0	7	3	21	44	40	1	18	20	32	19	35	37	39
13	16	14	8	10	16	6	33	4	27	49	46	52	25	51	34	43	56	72	79
17	23	24	28	29	30	31	36	42	47	50	55	58	60	63	65	66	67	81	83
22	45	38	53	61	41	62	82	54	48	59	57	71	78	64	80	70	76	85	87
96	95	94	86	89	69	68	97	73	92	74	88	99	84	75	90	77	93	98	91

Blauw is de mediaan der medianen (x in stap 3) groen is gegarandeerd lager, evenals het linker oranje gedeelte; rood en het rechter oranje stuk zijn gegarandeerd hoger. In het algemeen zijn gegarandeerd ongeveer $\frac{3n}{10}$ elementen uit het array kleiner dan x en gegarandeerd ongeveer $\frac{3n}{10}$ elementen groter dan x. In het ergste geval ga je de recursie dus in op de resterende $\approx 70\%$ van het array.

Laat $T(n)$ het aantal vergelijkingen zijn dat dit (recursieve) algoritme doet in de worst case.
Onder de aanname dat $T(n)$ stijgend is geldt:

$$T(n) \leq \begin{cases} \texttt{const} & n \leq \ldots \\ T(\lceil \frac{n}{5} \rceil) + T(\lceil \frac{7n}{10} \rceil + 6) + O(n) & n > \ldots \end{cases}$$

$< n$ $\leq dn$ $\uparrow$

als $n > 20$ bijv. 80

De term $O(n)$ komt van stap 1, 2 en 4 samen.

Bewering:

$T(n) \leq cn$ voor geschikte c en $n \geq \cdots$

ofwel: $T(n) \in O(n)$

Opgave 26

Elk algoritme voor het selectieprobleem dat is gebaseerd op arrayvergelijkingen doet in de worst case ten minste $\lceil \frac{n}{2} \rceil$ van zulke vergelijkingen.

Gevolg

Het selectieprobleem is $\Omega(n)$

Eindconclusie

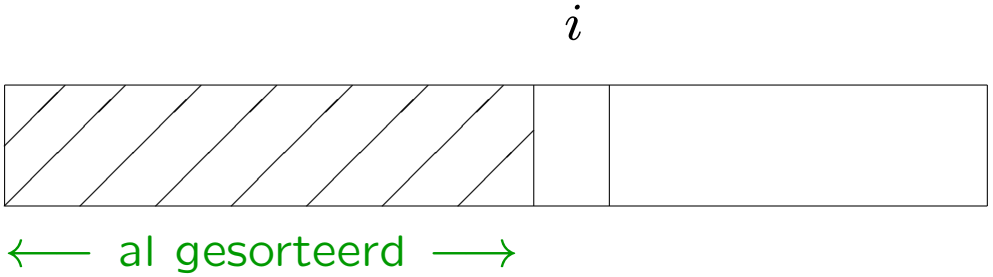
Het selectieprobleem is $\Theta(n)$

Probleem

Gegeven een rij (array) $A = A[1], \ldots, A[n]$. Sorteer A oplopend, dus $A[i] \leq A[i+1]$ voor alle i $(1 \leq i < n)$; $A[i] < A[i+1]$ als alle elementen verschillend zijn.

We bekijken eerst sorteeralgoritmen die gebaseerd zijn op het doen van arrayvergelijkingen

Insertion sort



Conceptueel idee: itereer over i en voeg $A[i]$ op de juiste plek in het gesorteerde stuk $A[1], \ldots, A[i-1]$ in door herhaald te vergelijken met de linkerbuur en te verwisselen indien nodig.

Het algoritme is gebaseerd op het doen van arrayvergelijkingen ($A[i] < A[j]$).

(1) **for** $i := 2$ **to** n **do**

// nu $A[i]$ op de juiste plek in $A[1], \ldots, A[i-1]$ invoegen

(2) $x := A[i];$

(3) $j := i - 1;$

(4) **while** $j > 0$ **and** $A[j] > x$ **do** (&) // $x = A[i]$

(5) $A[j+1] := A[j];$

(6) $j := j - 1;$

(7) **od**

(8) $A[j+1] := x;$

(9) **od**

- het aantal arrayvergelijkingen (&) is een goede maat voor de complexiteit.

- insertion sort doet eigenlijk steeds compare-exchange operaties: vergelijk en verwissel (indien nodig). Deze zijn hier vermomd als verschuivingen, waarna pas in de laatste stap $A[i]$ daadwerkelijk wordt neergezet.

- de verwisselingen zijn steeds buurverwisselingen.

- de for-loop (regel 1) doet precies $n-1$ iteraties
- de test (&), dus $A[j] > x$, gebeurt minstens één keer per iteratie van de for-loop, dus totaal *minstens* $n-1$ keer: $\#(\&) \geq n-1$
- het vergelijken van array-elementen (want $x = A[i]$) is een kenmerkende operatie voor dit sorteeralgoritme

Dan:

- regels 1, 2, 3 en 8 gebeuren elk precies $n-1$ keer*
- het eerste deel van regel 4 ($j > 0$) gebeurt per iteratie van de for-loop hooguit één keer vaker dan het tweede deel (&), dus in totaal hooguit $n-1$ keer. Dat betekent dat $\#(j > 0) \leq 2 \cdot \#(\&)$
- regels 5 en 6 gebeuren alleen als tweede test in regel 4 waar is, en vinden dus ook in totaal hooguit even vaak plaats als (&)

*eigenlijk gebeurt in regel 1 de test $i \leq n$ precies n keer (vergelijk met een while-loop), en dat is in orde van grootte hooguit even vaak als (&)

We tellen het aantal vergelijkingen $A[j] > x$ (waar $x = A[i]$).

1. Best case: $B(n) = \sum_{i=2}^{n} 1 = n - 1$
2. Worst case: $W(n) = \sum_{i=2}^{n} (i - 1) = \frac{1}{2} n(n-1)$
3. Average case*: $A(n) = \frac{1}{4} n(n-1) + n - \sum_{i=1}^{n} \frac{1}{i} \in \Theta(n^2)$

*onder de aanname dat alle $A[i]$'s verschillend zijn en dat alle $n!$ permutaties (ordeningen) van $A[1]$ t/m $A[n]$ even waarschijnlijk zijn. We middelen dan over alle mogelijke permutaties en dat zijn in essentie alle mogelijke invoerrijtjes. We mogen voor het gemak zelfs wel aannemen dat A de getallen 1 t/m n bevat. Zie verder het college.

Het minimale aantal vergelijkingen van Insertion sort is $n-1$.

- de eerste vergelijking op regel 4 is aanvankelijk altijd waar (want $i > 1$), dus de tweede vergelijking ($A[j] > x$) wordt elke iteratie minstens één keer uitgevoerd. Totaal $n-1$ iteraties, dus ten minste $n-1$ keer.

- $n-1$ kan worden gehaald: daartoe moet voor *elke* i de test (&) precies 1 keer worden gedaan, en dat gebeurt d.e.s.d.a. voor *elke* i ofwel direct $A[i-1] \leq A[i]$ is, ofwel $j \leq 0$ vóór de tweede vergelijking $A[j] > x$. Dat laatste komt alleen voor als $i = 2$ (en $A[1] > A[2]$).

- *twee* goede rijtjes: oplopend gesorteerd of alleen de eerste twee elementen staan verkeerd om:
  $A[1] \leq A[2] \leq \cdots \leq A[n]$ of $A[2] < A[1] \leq A[3] \leq \cdots \leq A[n]$.

Het maximale aantal vergelijkingen van Insertion sort is $\frac{1}{2}n(n-1)$.

- de tweede vergelijking op regel 4 kan alleen gebeuren zolang $j > 0$: dat is hooguit $i-1$ keer per iteratie van de for-loop. Totaal dus hooguit $\sum_{i=2}^{n}(i-1) = \frac{1}{2}n(n-1)$ keer.
- dit kan worden gehaald: daartoe moet in elke doorgang (voor elke i dus) $A[i]$ met alle voorgaande $A[j]$ vergeleken worden. Dit betekent dat voor alle $j = i-1, \ldots, 2$ moet gelden dat $A[j] > A[i]$. Voor $j = 1$ wordt de test (&) nog wel gedaan, maar maakt de uitslag daarvan niet meer uit.
- slechte rijtjes: alle rijtjes waarvoor voor elke $A[i]$ met $i \geq 2$ geldt dat $A[i]$ het kleinste of het op één na kleinste element is van $A[1], \ldots, A[i]$.
- Zie verder opgave 7c.

Definitie: een inversie van de permutatie $A[1], A[2], \ldots, A[n]$ is een paar $(A[i], A[j])$ waarvoor $i < j$ en $A[i] > A[j]$. M.a.w.: een inversie is een paar $(A[i], A[j])$ dat verkeerd om staat.

Merk op: *elk* sorteeralgoritme moet *alle* aanwezige inversies opheffen.

Verder: als een sorteeralgoritme altijd hooguit één inversie opheft per arrayvergelijking, dan is het aantal vergelijkingen dat wordt gedaan om $A[1], \ldots, A[n]$ te sorteren *ten minste* het aantal inversies van A.

Ten slotte: een buurverwisseling (zoals bij Insertion sort) heft altijd precies één inversie op (aangenomen dat de buurelementen verkeerd om staan).

Stelling

Het maximale aantal inversies dat kan voorkomen in een rijtje van n verschillende waarden is $\binom{n}{2} = \frac{1}{2}n(n-1)$. Dit treedt alleen op bij een omgekeerd (= aflopend) gesorteerd rijtje.

Gevolg

Elk sorteeralgoritme dat sorteert met behulp van arrayvergelijkingen en dat per vergelijking hooguit één inversie opheft, doet ten minste $\frac{1}{2}n(n-1)$ vergelijkingen in de worst case.

Conclusie: Insertion sort is optimaal voor wat betreft de worst case, binnen de klasse van algoritmen gebaseerd op het doen van arrayvergelijkingen, waarbij per vergelijking hooguit één inversie wordt opgeheven (bijvoorbeeld via buurverwisselingen).

Stelling

Het *gemiddeld* aantal inversies in een permutatie van n verschillende waarden (bijvoorbeeld de getallen 1 t/m n) is $\frac{1}{4}n(n-1)$. Dit onder de aanname dat alle $n!$ permutaties even waarschijnlijk zijn.

Gevolg

Elk algoritme dat sorteert met behulp van arrayvergelijkingen en dat per vergelijking ten hoogste één inversie opheft, doet ten minste $\frac{1}{4}n(n-1)$ vergelijkingen in de average case.

Conclusie: Insertion sort doet gemiddeld $\frac{1}{4}n(n-1)+n-1-\sum_{i=2}^{n}\frac{1}{i} \in \Theta(n^2)$ vergelijkingen, dus Insertion sort is optimaal in orde van grootte wat betreft de average case en binnen de beschreven klasse van algoritmen.

Voor sorteeralgoritmen gebaseerd op arrayvergelijkingen, waarbij per arrayvergelijking hooguit één inversie wordt opgeheven*, geldt:

- # arrayvergelijkingen $\geq$ # inversies invoerarray
- # arrayvergelijkingen in de worst case $\geq \frac{1}{2}n(n-1)$

Als je een beter sorteeralgoritme (dat gebaseerd is op arrayvergelijkingen) wilt, moet je elementen verwisselen die verder van elkaar liggen, zoals gebeurt bij Mergesort, Quicksort, Shellsort (en veel meer).

*zoals bij algoritmen die gebruikmaken van buurverwisselingen, zoals Insertion sort en Bubblesort

Extra opgave, zie werkcollege:
Stel dat $A[i]$ en $A[i+k]$ $(k > 0)$ verkeerd om staan en dat we die verwisselen. Hoeveel inversies worden dan ten minste respectievelijk ten hoogste opgeheven?

Situatie:

met $A[i] > A[i+k]$. Verwissel nu $A[i]$ en $A[i+k]$.

Complexiteit van recursieve algoritmen:

- in het algemeen is hier het aantal recursieve aanroepen een goede maat voor de hoeveelheid werk
- maar je kunt ook hier het aantal keer dat de basisoperatie wordt uitgevoerd tellen
- een en ander leidt tot recurrente betrekkingen

Als voorbeeld bekijken we een recursief algoritme voor het bepalen van het maximum van n getallen, opgeslagen in een array A. Basisoperatie: vergelijken van array-elementen.

```
int grootste(int A[ ], n) {
    if n = 1 then
        return A[n];
    else
        max := grootste(A, n − 1);
        if A[n] > max then
            max := A[n];
        fi
    fi
    return max;
}
```

werk per aanroep ($n > 1$)

Laat $C(n)$ = aantal arrayvergelijkingen ($A[n] > max$), dan geldt:

$$C(n) = \begin{cases} 0 & n = 1 \\ C(n-1) + 1 & n > 1 \end{cases}$$

Oplossing: $C(n) = n - 1$

- Rij van Fibonacci: $0, 1, 1, 2, 3, 5, 8, 13, 21, \ldots$
- Vanaf het derde element: som van de voorgaande twee
- Een recurrente betrekking is een voorschrift om een waarde $T(n)$ te berekenen door middel van zijn voorganger(s), dus bijvoorbeeld $T(n-1)$, of $T(\frac{n}{2})$, of ...

Voor de Fibonacci-getallen geldt:

$$T(n) = \begin{cases} 0 & n = 0 \\ 1 & n = 1 \\ T(n-1) + T(n-2) & n > 1 \end{cases}$$

Een ander voorbeeld van een recurrente betrekking:

$$T(n) = \begin{cases} 1 & n = 1 \\ 2T(\frac{n}{2}) + n & n = 2^k > 1 \end{cases}$$

- ▷ vorm een vermoeden voor een gesloten formule voor $T(n)$ door middel van:
  - herhaalde substitutie en afleiden algemene vorm, of:
  - probeer wat termen door te rekenen: zie je een patroon?
- ▷ bewijs ten slotte de formule met volledige inductie

Oplossing: $T(n) = n + n \lg n \in \Theta(n \lg n)$

De vorige recurrente betrekking, maar nu voor algemene n, dus niet alleen voor tweemachten:

$$T(n) = \begin{cases} 1 & n = 1 \\ 2T(\lfloor \frac{n}{2} \rfloor) + n & n > 1 \end{cases}$$

Dan geldt: $T(n) \in O(n \lg n)^*$ (en overigens ook $T(n) \in \Theta(n \lg n)$).

Dit kan worden bewezen door met behulp van volledige inductie bijvoorbeeld aan te tonen dat $T(n) \leq 2n \lg n$ voor alle $n \geq 2$.

*voor een algemene stelling, zie bijvoorbeeld de Smoothness Rule in The Design & Analysis of Algorithms van A. Levitin

- het kan vaak helpen om termen los te laten staan, waardoor je een sommatie beter kan herkennen, en/of de regelmaat eerder kunt ontdekken
- volledige inductie:

  (i) *zwak*: neem aan dat het geldt voor $n-1$*

  (ii) *sterk*: neem aan dat het geldt voor *alle* voorgaande n†
- nog een voorbeeld (opgave 14b):

$$T(n) = \begin{cases} 1 & n = 1 \\ 2T(n-1)+1 & n > 1 \end{cases}$$

*of voor $\frac{n}{2}$ of $\cdots$

†met $n = 2^k, k \geq 0$ of $\cdots$

En ten slotte nog twee voorbeelden om op te lossen.

(i) $$T(n) = \begin{cases} 3 & n = 1 \\ T(n-1) + n - 1 & n > 1 \end{cases}$$

Oplossing: $T(n) = 3 + \frac{1}{2}n(n-1)$

(ii) $$T(n) = \begin{cases} 0 & n = 1 \\ 2T(\frac{n}{4}) + \sqrt{n} & n = 4^k > 1 \end{cases}$$

Oplossing: $T(n) = \sqrt{n}\log_4 n = \frac{1}{2}\sqrt{n}\lg n$

- **Volgende college:** vrijdag 14 maart, 11.00 – 12.45,
  Gorlaeusgebouw zaal BW.0.40

- **Straks werkcollege:** vrijdag 7 maart, 13.15 – 15.00,
  Gorlaeusgebouw zalen BW.0.29 en BW.0.30
  Opgaven: 7, extra, opgave 2c tentamen juni 2024, 14aeg

- **Eerste huiswerkopgave:**
  zie website; deadline maandag 10 maart 2025;
  inleveren via Brightspace

- **Website:**
  liacs.leidenuniv.nl/~graafjmde/COMP/