Complexiteit

Uitwerking Opgave 26

Opgave 26:
Aan te tonen: 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. Voor het selectieprobleem hebben we aangenomen dat het invoerarray verschillende waardes bevat; in dat geval is de k-de in grootte namelijk uniek gedefinieerd.

We bewijzen dit uit het ongerijmde, d.w.z. we veronderstellen dat het niet zo is. Dan is er dus een correct werkend algoritme voor het selectieprobleem dat in de worst case echt minder dan $\lceil \frac{n}{2} \rceil$ vergelijkingen doet (*). Met andere woorden, dit algoritme doet voor *elk* invoerrijtje minder dan $\lceil \frac{n}{2} \rceil$ vergelijkingen. In het bijzonder dus voor rijtjes bestaande uit verschillende elementen.
Dat het algoritme minder dan $\lceil \frac{n}{2} \rceil$ arrayvergelijkingen doet betekent dat het ten minste één array-element niet bekijkt (= niet vergelijkt met een ander array-element); daarvoor zijn immers minstens $\lceil \frac{n}{2} \rceil$ arrayvergelijkingen nodig. Dit leidt als volgt tot een tegenspraak.

Stel dat het algoritme $A[j]$ als k-de in grootte aanwijst, maar $A[\ell]$ nooit vergeleken heeft. Dan kunnen we twee rijtjes B en C construeren, waarbij het algoritme op minstens één van de twee een verkeerd antwoord geeft. We nemen als invoerrijtjes B en C rijtjes die gelijk zijn aan A, met als enig verschil dat $B[\ell]$ groter is dan alle elementen van A en $C[\ell]$ kleiner dan alle elementen van A. Dan is dus $B[\ell]$ de grootste uit B en $C[\ell]$ de kleinste uit C. We nemen even aan dat $j \neq \ell$. Het geval dat $j = \ell$ moet apart bekeken worden.[1]
Het algoritme doet noodzakelijkerwijs precies hetzelfde op B en C als op A, want $A[\ell]$ werd niet bekeken en de vergeleken waardes zijn precies hetzelfde, en daarmee ook de gegeven antwoorden en het hele verloop van het algoritme. Het geeft dus ook voor *beide* rijtjes het j-de element als de k-de in grootte: dus $B[j]$ wordt aangewezen als de k-de in grootte van B en $C[j]$ als de k-de in grootte van C.
Voor $k = 1$ geeft dit meteen al een tegenspraak, aangezien $C[j]$ niet de kleinste waarde uit C is, maar $C[\ell]$. Voor $k = n$ geheel analoog: $B[j]$ is niet de grootste van B, want dat is $B[\ell]$. In beide gevallen wordt dus ofwel voor B ofwel voor C een verkeerd antwoord gegeven.

-voor het vervolg zie volgende pagina-

[1] Het algoritme heeft $A[j]$ als k-de in grootte aangewezen en $A[j]$ niet vergeleken met een ander array-element. Maak nu een nieuwe invoer B die overal gelijk is aan A, behalve op positie j. Het algoritme zal op B precies hetzelfde doen als op A (want $A[j]$ werd niet bekeken) en geeft dus ook $B[j]$ als k-de in grootte. Als $k = 1$ kiezen we $B[j]$ groter dan alle elementen van A, anders kleiner dan alle elementen van A. In beide gevallen wijst het algoritme het verkeerde element als k-de in grootte aan. Immers als $k = 1$ is $B[j]$ niet de kleinste waarde uit B, en als $1 < k \leq n$ is $B[j]$ wel de kleinste, maar dus niet de k-de in grootte. Tegenspraak.

Ook voor het geval dat $1 < k < n$ geeft dit een tegenspraak: als $C[j]$ de k-de in grootte uit B is, dan zijn er behalve $C[\ell]$ slechts $k-2$ elementen kleiner, dus er zijn in totaal ook maar $k-2$ elementen kleiner dan $B[j]$, en derhalve kan $B[j]$ niet de k-de in grootte van B zijn. Analoog omgekeerd. Het algoritme wijst dus voor ten minste één van de twee invoerrijtjes B of C het verkeerde element als k-de element in grootte aan. Dit is in tegenspraak met de correctheid van het algoritme. De aanname dat minstens één element niet bekeken is kan dus niet waar zijn. Conclusie: de oorspronkelijke aanname (*) is onjuist.