Tentamen Algoritmiek

Dinsdag 12 juni 2012, 10.00 - 13.00

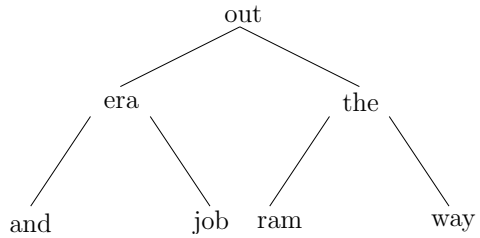
Dit tentamen bevat *vier* vragen. Geef een *duidelijke* toelichting bij al je antwoorden!

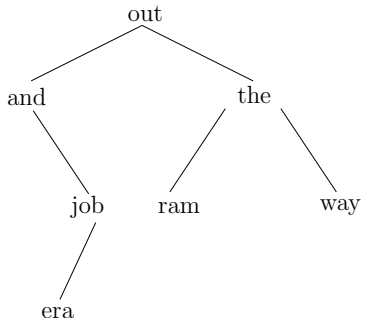
Vraag 1: Optimale binaire zoekbomen (30 punten)

Stel je voor dat je een vertaalprogramma schrijft dat Engelse woorden vertaalt in Nederlandse. We moeten dus het Engelse woord opzoeken in een datastructuur en het Nederlandse equivalent wat daarbij hoort ophalen. Aangezien woorden gesorteerd kunnen worden - op alfabetische volgorde - ligt het voor de hand om een binaire zoekboom te gebruiken. Echter, woorden komen niet allemaal even vaak voor en sommige binaire zoekbomen kosten daarom minder tijd om te doorzoeken dan andere. Hieronder vind je een rijtje met zeven Engelse woorden en hun relatieve frequentie; neem aan dat dit alle woorden zijn die voor kunnen komen. Daarnaast vind je twee zoekbomen: een *complete binaire zoekboom* en een alternatieve binaire zoekboom.

Tabel 1 - zeven Engelse woorden en hun frequenties

and	era	job	out	ram	the	way
0.20	0.05	0.10	0.15	0.05	0.30	0.15





De *gemiddelde verwachte padlengte* voor een boom definieren we als de som (over alle woorden) van het aantal woorden dat bekeken moet worden totdat dit woord gevonden wordt, maal de kans dat we dit woord zoeken, dus de gemiddelde verwachte padlengte voor een zoekboom met n woorden $w_1, \ldots, w_n$ is gedefinieerd als $\sum_{i=1}^{n}(h(w_i)+1)\cdot p_i$, waarbij $h(w_i)$ de diepte van woord w_i in de boom aangeeft en p_i de relatieve frequentie van dat woord.

a) Bereken de *gemiddelde verwachte padlengte* voor beide bomen.

Merk op dat iedere subboom van deze binaire bomen woorden bevat die in de woordenlijst een aaneengesloten geheel vormen, d.w.z., als de woordenlijst woorden $w_1, \ldots, w_n$ bevat, dan bevat iedere subboom de woorden w_i tot en met w_j voor zekere $1 \leq i \leq j \leq n$. In het bijzonder geldt voor de wortel (dat kan ieder woord w_r zijn met $1 \leq r \leq n$) dat de linker subboom de woorden w_1 t/m w_{r-1} bevat en de rechter subboom de woorden w_{r+1} t/m w_n. De optimale boom bevat dus de wortel w_r zodanig dat de som van de totale padkosten van de linker subboom plus die van de rechter subboom minimaal zijn.

b) We gaan het vinden van de optimale zoekboom oplossen met dynamisch programmeren. Hieronder vind je de recurrente betrekking voor de te verwachten kosten $e(i,j)$ voor een binaire zoekboom voor het i-de tot en met j-de woord. Verklaar hoe deze betrekking opgebouwd is en wat deze uitdrukt; besteed hierbij aandacht aan:

1) wat is het basisgeval?
2) wat zijn de subproblemen?
3) wat is het gevolg van een bepaalde keuze voor r?
4) en geef aan hoe (d.w.z. in welke volgorde) de tabel gevuld moet worden op basis van deze betrekking.

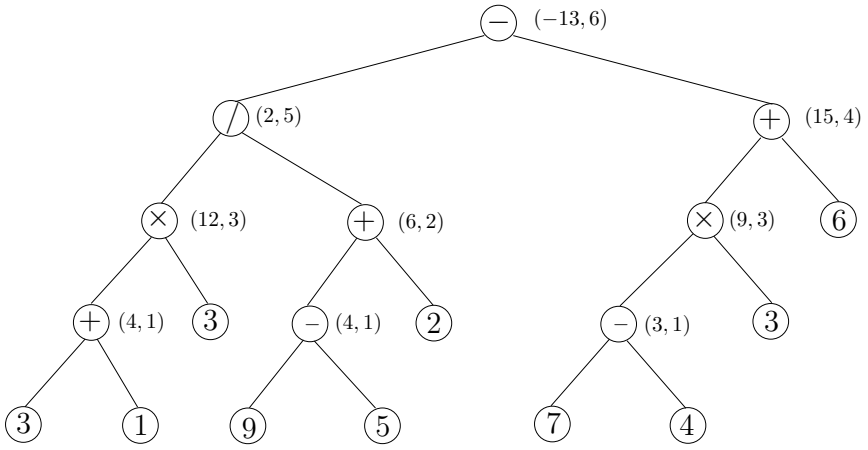
$$e(i,j) = \begin{cases} p_i & \text{als } i = j \\ 0 & \text{als } i > j \\ min_{i \le r \le j} e(i, r-1) + e(r+1, j) + \sum_{k=i}^{j} p_k & \text{als } i < j \end{cases}$$

c) Geef een algoritme voor het vullen van de array. Wat is de tijdscomplexiteit van dit algoritme? Motiveer je antwoord.

d) Een alternatief voor dit DP-algoritme is een gretig algoritme dat een zoekboom opbouwt door telkens het woord met de hoogste frequentie te kiezen als wortel van de (sub-)boom. Laat zien welke binaire zoekboom dit algoritme oplevert voor Tabel 1 en toon aan dat dit geen optimale oplossing is.

Vraag 2: Evaluatie rekenkundige expressie (22 punten)

Een rekenkundige expressie, zoals $((((3+1)\times 3)/((9-5)+2)) - (((7-4)\times 3)+6))$, die door een compiler geevalueerd moet worden, kan gerepresenteerd worden door een (geordende) binaire boom, zie bijgaande figuur voor de binaire boom horende bij deze expressie, met per interne knoop de waarde van die knoop (de linkerwaarde tussen haakjes).



Een knoop in deze boom ziet er als volgt uit:

```
struct knoop {
knoop* links;
knoop* rechts;
knoop* ouder;
char operator; // '+', '-', '/', 'x' of het ascii karakter NUL
int waarde;
int tijd;
}; // knoop
```

Het veld `operator` bevat de operator die bij een interne knoop hoort, of `NUL` als de knoop een blad is. Het veld `waarde` is initieel alleen voor de bladeren gevuld, en bevat dan de waarde die bij het blad hoort. Het veld `tijd` is initieel niet gevuld.

a) Geef een recursieve functie (in pseudo-code) `EVALUEER(knoop *wortel)` die voor iedere interne knoop het veld `waarde` vult met de waarde van de sub-expressie behorende bij de sub-boom waarvan die knoop de wortel is.

We willen weten hoeveel tijd het kost om de expressie uit te rekenen. Een operator kan pas worden toegepast als de deelexpressies in beide subbomen zijn uitgerekend. Deze beide deelexpressies kunnen parallel worden uitgerekend. Een vermenigvuldiging of deling kost twee tijdseenheden, een optelling of aftrekking een tijdseenheid. In de voorbeeldboom is de resulterende tijd aangegeven bij de interne knopen (de rechterwaarde tussen de haakjes).

b) Geef een recursieve functie `REKENTIJD(knoop *wortel)` die voor *iedere* knoop het veld `tijd` vult met de benodigde tijd om de deelexpressie bij die knoop uit te rekenen.

c) Nu gaan we een blad `oud` (zonder operator dus) vervangen door een subboom met drie knopen: een knoop `nieuw` met een operator, met als linkerkind een knoop met een waarde en als rechterkind een knoop met een waarde. In deze subboom zijn de velden `tijd` nog niet ingevuld. We kunnen bijvoorbeeld de knoop met waarde '6' vervangen door een subboom met '+' als operator en '2' en '4' als bladeren. Geef een niet-recursieve functie `VERVANG(knoop *oud, knoop *nieuw)` die deze vervanging uitvoert, en daarbij op een efficiente manier de velden `tijd` in de boom aanpast.

Vraag 3: NIM-voor-drie (23 punten)

De drie vrienden (althans, voorlopig nog wel) Adrie, Bert en Caspar spelen een variant van het spelletje NIM. Alledrie hebben ze een stapel lucifers voor zich en om beurten mogen ze van hun eigen stapel *twee* lucifers pakken, en daarvan *één* lucifer op een van de andere stapels leggen. Daarna is de volgende aan zet: eerst Adrie, dan Bert, dan Caspar en dan weer Adrie enzovoorts. Het spel is afgelopen als de speler die aan zet is nog maar een of twee lucifers heeft. Merk op dat het totaal aantal lucifers bij iedere zet één minder wordt. *Optimaal spel* betekent bij dit spel, dat spelers kiezen voor een zet waarbij ze (de grootste) kans om te winnen hebben. Als een speler in een bepaalde stand geen kans meer heeft om te winnen (bij optimaal spel van de andere spelers) kan hij een willekeurige zet doen.

Stel dat het spel begint met n lucifers voor iedereen (dus in totaal drie maal n lucifers)

a) Wat zijn de mogelijke toestanden en acties in dit spel? Geef het volledige toestand-actie-diagram voor dit spel, beginnend met Adrie en met ieder drie lucifers. Geef bij elke toestand aan voor wie deze winnend is / kan zijn.

b) Heeft Adrie in deze start-toestand (ieder drie lucifers, Adrie aan zet) een *winnende* strategie? Zo ja, geef aan wat die strategie is; zo nee, leg uit waarom!

c) Neem nu de situatie waarin Caspar aan zet is en er nog twee lucifers voor Adrie, twee voor Bert, en vijf voor Caspar zelf op tafel liggen. Laat met een toestand-actie-diagram zien dat Caspar niet meer kan winnen, maar dat Caspar wel kan bepalen of Adrie of Bert wint.

Vraag 4: Pizza's bezorgen (25 punten)

Een pizzabedrijf heeft vier vestigingen in de stad en een call-center waarbij de bestellingen binnenkomen. Iedere vestiging heeft één bezorger. Op enig moment komen er vier bestellingen binnen bij het call-center die over de vier vestigingen verdeeld moeten worden. Iedere vestiging kan in principe iedere klant bedienen, maar de bezorgtijd verschilt natuurlijk en is afhankelijk van de precieze locatie van klant en vestiging. Die bezorgtijd vind je hieronder (Tabel 2) voor iedere combinatie van klant en vestiging.

Tabel 2 - bezorgtijden per klant en vestiging

	klant 1	klant 2	klant 3	klant 4
vestiging A	12	35	7	6
vestiging B	8	17	10	13
vestiging C	14	14	12	6
vestiging D	9	21	9	12

Het pizza-bedrijf wil de *totale bezorgtijd* minimaliseren, en daarmee ook de tijd dat de bezorgers onderweg zijn.

a) Beschrijf in woorden een *backtracking* algoritme voor dit probleem voor algemene n en licht het toe aan de hand van het voorbeeld. Teken de eerste twee niveaus van de state-space-tree (dus de wortel en de kinderen van de wortel) *volledig* en werk de state-space-tree voor het *eerste* kind volledig uit. De rest van de state-space-tree hoef je niet uit te werken. Geef de volgorde aan waarin de knopen (deeloplossingen) bekeken worden.

b) Geef aan hoe bij *branch and bound* deeloplossingen worden opgebouwd, wat er bij iedere stap (knoop) in de state-space-tree wordt berekend en gecontroleerd, en hoe *best-fit-first* branch and bound werkt voor minimalisatieproblemen zoals dit probleem. Geef duidelijk aan wat de verschillen zijn met backtracking.

c) Beschrijf in woorden een *branch and bound* algoritme dat het pizza-bezorgprobleem oplost, pas het algoritme toe op het voorbeeld en teken de state-space-tree. Geeft duidelijk aan in welke volgorde de knopen bezocht worden. Geef ook aan welke ondergrens je voor het totaal aantal bezorgminuten gebruikt, en wat deze ondergrens *bij iedere deeloplossing* is.

Einde van dit tentamen. Veel succes!