# Uitwerking tentamen Algoritmiek, 12 juni 2012



1(a) ~~11:57~~
12:03

Linkerboom:

and era job out ram the way
$3*0{,}20 + 2*0{,}05 + 3*0{,}10 + 1*0{,}15 + 3*0{,}05 + 2*0{,}30 + 3*0{,}15$
$= 2{,}35$

niveau-orde: $1*0{,}15 + 2*0{,}05 + 2*0{,}30 + 3*0{,}20 + 3*0{,}10 + 3*0{,}05 + 3*0{,}15$
$= 2{,}35$

Rechterboom:

and era job out ram the way
$2*0{,}20 + 4*0{,}05 + 3*0{,}10 + 1*0{,}15 + 3*0{,}05 + 2*0{,}30 + 3*0{,}15$
$= 2{,}25$

niveau-orde:
$1*0{,}15 + 2*0{,}20 + 2*0{,}30 + 3*0{,}10 + 3*0{,}05 + 3*0{,}15 + 4*0{,}05$
$= 2{,}25$

12:08

(b)

1] Er zijn twee basisgevallen
* $i > j$, dan hebben we helemaal geen woorden $\Rightarrow$ ook geen kosten $\Rightarrow e(i,j) = 0$.
* $i = j$, dan hebben we één woord $w_i$, en er is maar één boom mogelijk: ($w_i$) met kosten $1 * p_i = p_i$

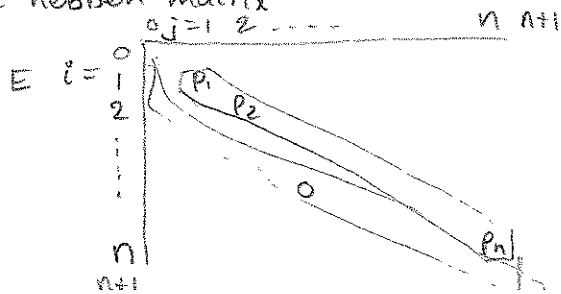
2] als $i < j$ zijn er meerdere kandidaten voor de wortel: $w_r$ met $i \le r \le j$
Die moeten we allemaal proberen.
Als we een $w_r$ proberen, komen $w_i \cdots w_{r-1}$ in de linkersubboom terecht. Daar moeten we een opt. BZB bij vinden, met kosten $e_{i,r-1}$. Omdat al deze woorden één niveau lager hangen in de totale boom voor $i, \ldots j$ (namelijk: onder $w_r$) komt er kosten $p_i + \cdots + p_{r-1}$ bij
Net zo krijgen we bij de rechtersubboom kosten $e_{r+1,j} + p_{r+1} + \cdots + p_j$
Ten slotte levert wortel $w_r$ zelf kosten op: $1 * p_r$
$\Rightarrow$ totale kosten bij wortel $w_r$:

$$e(i,r-1) + e(r+1,j) + p_i + \cdots + p_{r-1} + p_{r+1} + \cdots + p_j =$$
$$e(i,r-1) + e(r+1,j) + \sum_{k=i}^{j} p_k$$

12:19 NB.: formule gaat ook goed als $r = i$ of $r = j$, want $e_{i,i-1} = 0$ (linkersubboom is leeg) en $e_{j+1,j} = 0$ (rechter subboom is leeg)
12:33

3] Zie bij 2:
dat $w_i \cdots w_{r-1}$ dus in linker subboom van $w_r$ terechthomen, een niveau lager
dat $w_{r+1} \cdots w_j$ " " rechter " " " " " " " "

4] We hebben matrix




12:42

We zijn geïnteresseerd in $E(1,n)$
We beginnen met hoofddiagonaal (waar $i=j$) en met diagonaal daar links onder (de twee basisgevallen)
Vervolgens werken we diagonaal-na-diagonaal naar rechtsboven.
Voor element $E(i,j)$ gebruik je elementen

Die zijn dan al bekend

4:21

(c)

```
for i = 1 to n do
    E[i][i] = p_i
od
for i = 1 to n+1 do
    E[i][i-1] = 0
od
for d = 1 to n-1 do    // d is verschil tussen i en j op een diagonaal
    for i = 1 to n-d do
        j = i+d ;      // bepaal nu E[i][j]
        kanssom = 0 ;
        for k = i to j do
            kanssom = kanssom + p[k]
        od
        minim = n+1;
        for r = i to j do
            if E[i][r-1] + E[r+1][j] + kanssom < minim then
                minim = E[i][r-1] + E[r+1][j] + kanssom
            fi
        od
        E[i][j] = minim
    od
od
```

De looptijd / complexiteit van dit algoritme is $O(n^3)$, want er zijn drie geneste for lussen:

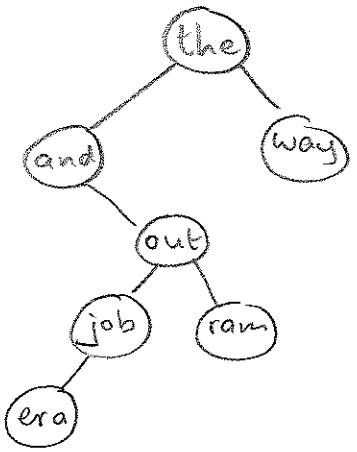
for d
  for i
    for k

of

for d
  for i
    for r.

d

Dit levert als wortel 'the', want hoogste frequentie




Linker subboom van 'the' bevat 'and' – 'ram' met als wortel 'and'
Rechter subboom van 'and' bevat 'era' – 'ram', met wortel 'out'
Linker subboom van 'out' bevat 'era' – 'job', met wortel 'job'
De rest ligt dan vast (want BZB)

Gemiddelde padlengte van deze boom is
$2*0{,}20 + 5*0{,}05 + 4*0{,}10 + 3*0{,}15 + 4*0{,}05 + 1*0{,}30 + \frac{3}{2}*0{,}15 = 2{,}30$

Dit is meer dan bij de rechter boom van (a)
$\Rightarrow$ geen optimale oplossing.

Niveau-orde: $1*0{,}30 + 2*0{,}20 + 2*0{,}15 + 3*0{,}15 + 4*0{,}10 + 4*0{,}05 + \text{Graa} = 2{,}30$

14:51.

2(a)

```
Evalueer ( knoop * wortel)
{   // pre: wortel ≠ NULL
                                int waardelinks, waarderechts;
    if (wortel->operator ≠ NUL)
    { Evalueer (wortel->links);   waardelinks = wortel->links->waarde;
      Evalueer (wortel->rechts);  waarderechts = wortel->rechts->waarde;
      switch (operator)
      { case +: wortel->waarde = waardelinks + waarderechts ;
                break
        case -: wortel->waarde = waardelinks - waarderechts ;
                break
        case /: wortel->waarde = waardelinks / waarderechts)
                break
        case *: wortel->waarde = waardelinks * waarderechts
      }//switch
    } // if
} // EVALUEER
```

14:59

b)
```
REKENTIJD (knoop *wortel)
// pre: wortel ≠ NULL
{ int tijdlinks, tijdrechts, maxim;

  if (wortel -> operator == NUL)   // waarde ⇔ geen operator
    wortel -> tijd = 0
  else
  { REKENTIJD (wortel -> links);
    tijdlinks = wortel -> links -> tijd;
    REKENTIJD (wortel -> rechts);
    tijdrechts = wortel -> rechts -> tijd;
    if (tijdlinks >= tijdrechts)
      maxim = tijdlinks;
    else
      maxim = tijdrechts;
    if ((wortel -> operator == '+') or (wortel -> operator == '-'))
      wortel -> tijd = maxim + 1;
    else                    // / of *
      wortel -> tijd = maxim + 2;
  }
}
```

14:08

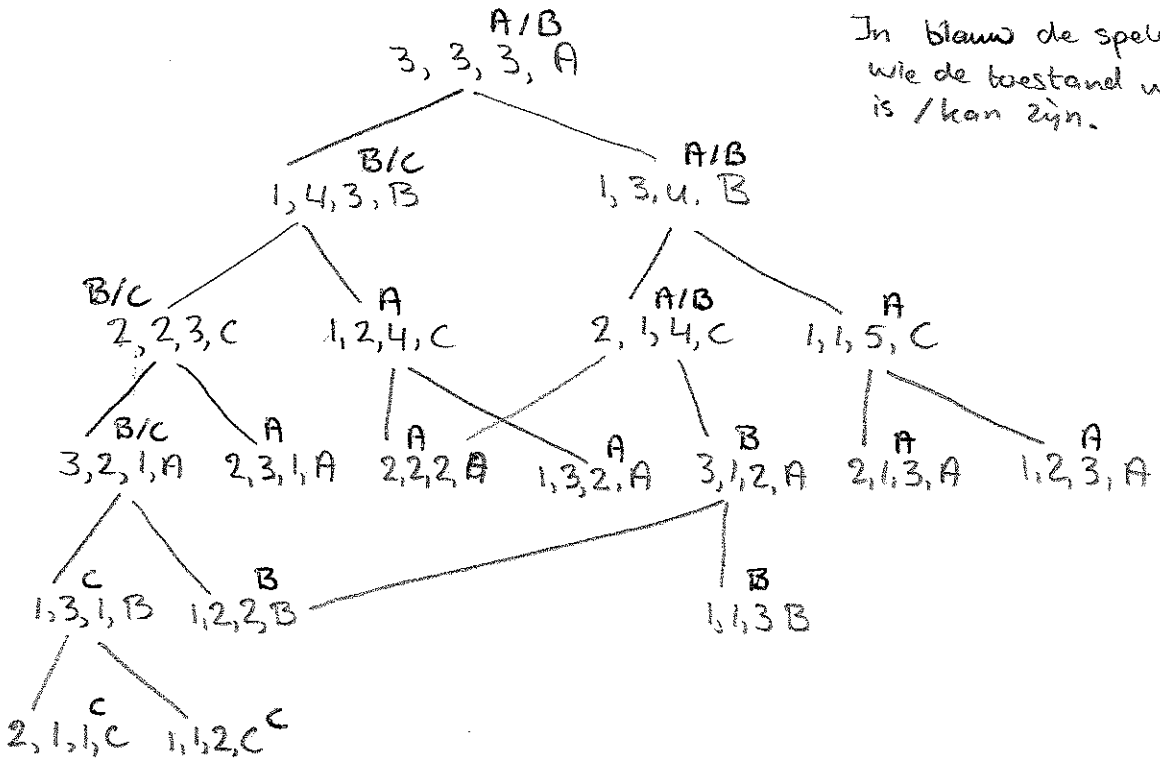
c)
```
VERVANG (knoop *oud, knoop *nieuw)
{ knoop *ouderoud, *loper; bool klaar; int tijdlinks, tijdrechts,
                                        maxim, nieuwtijd;
  // hang de subboom op zijn plaats
  ouder_oud = oud -> ouder
  if (oud == ouderoud -> links)          //hmm, wat als oud de wortel is?
    ouderoud -> links = nieuw            //bonus voor wie daaraan denkt
  else
    ouderoud -> rechts = nieuw
  nieuw -> ouder = ouderoud;             <- delete oud;
    // pas velden 'tijd' aan,
    // eerst onderin de nieuwe subboom,
    // en vervolgens van beneden naar boven
  nieuw -> links -> tijd = 0
  nieuw -> rechts -> tijd = 0
  nieuw -> tijd = 0;   // tijdelijke waarde.
  loper = nieuw;
  klaar = false;
  while (loper ≠ NULL && not klaar)
  { tijdlinks = loper -> links -> tijd;
    tijdrechts = loper -> rechts -> tijd;
    if (tijdlinks >= tijdrechts)
      maxim = tijdlinks;
    else
      maxim = tijdrechts;
```

```
if ((loper -> operator == '+') or (loper -> operator == '-'))
   nieuwtijd = maxim +1;
else    // / of *
   nieuwtijd = maxim +2;

if (nieuwtijd > loper -> tijd)
{ loper -> tijd = nieuwtijd;
   loper = loper -> ouder;
}
else
  klaar = true;

} // while

} // vervang
```

15:21

3) a) toestanden: drietallen getallen: de aantallen lucifers van Adrie, Bert en Caspar (a,b,c)
met een indicatie wie aan de beurt is.: A, B of C.
Er moet in ieder geval gelden dat $a+b+c \leq 3n$
Ook geldt dat $a \geq 1$, $b \geq 1$, $c \geq 1$

acties: overgangen van ene toestand naar andere toestand, waarbij aantal van 'speler-aan-beurt' met 2 afneemt, en aantal van een van de andere spelers met 1 toeneemt, waarna volgende speler aan beurt is



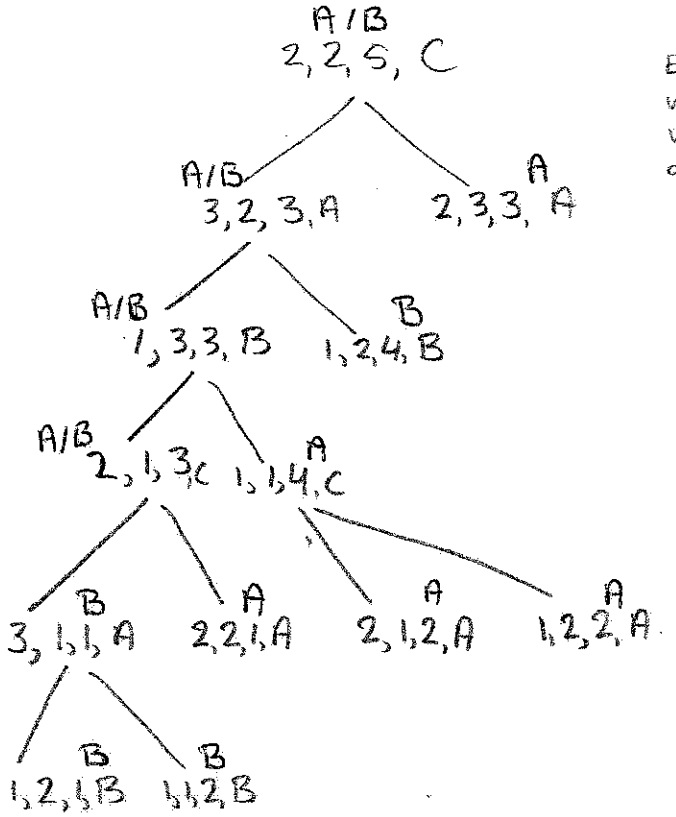

In blauw de speler(s) voor wie de toestand winnend is / kan zijn.

15:36

b) Adrie heeft niet een per se winnende strategie,
maar als hij/zij de eerste lucifer aan C (Caspar) geeft, kán Adrie winnen.
Namelijk als Caspar straks een lucifer aan Bert geeft.
Als Adrie eerste lucifer aan Bert geeft, zal Adrie zeker niet winnen

15:40.

c)




Er is geen enkele mogelijke eindtoestand waarin C wint ⇒ C kan niet meer winnen, zelfs niet bij dom spel van de andere spelers

C heeft op twee momenten de mogelijkheid om te bepalen of A of B gaat winnen.

Als C zijn eerste lucifer aan B geeft, wint A direct.
Als C zijn eerste lucifer aan A geeft, krijgt hij drie zetten later nogmaals de mogelijkheid om óf B óf A te laten winnen.

(7)

1 (a) Backtracking:

Een oplossing bestaat uit de toewijzing van iedere vestiging aan een klant.
Deze oplossing bouwen we stap voor stap op:
eerst vestiging A aan een klant toewijzen,
dan vestiging B aan een klant toewijzen (een andere dan A)
dan vestiging C aan weer een andere klant toewijzen enz.

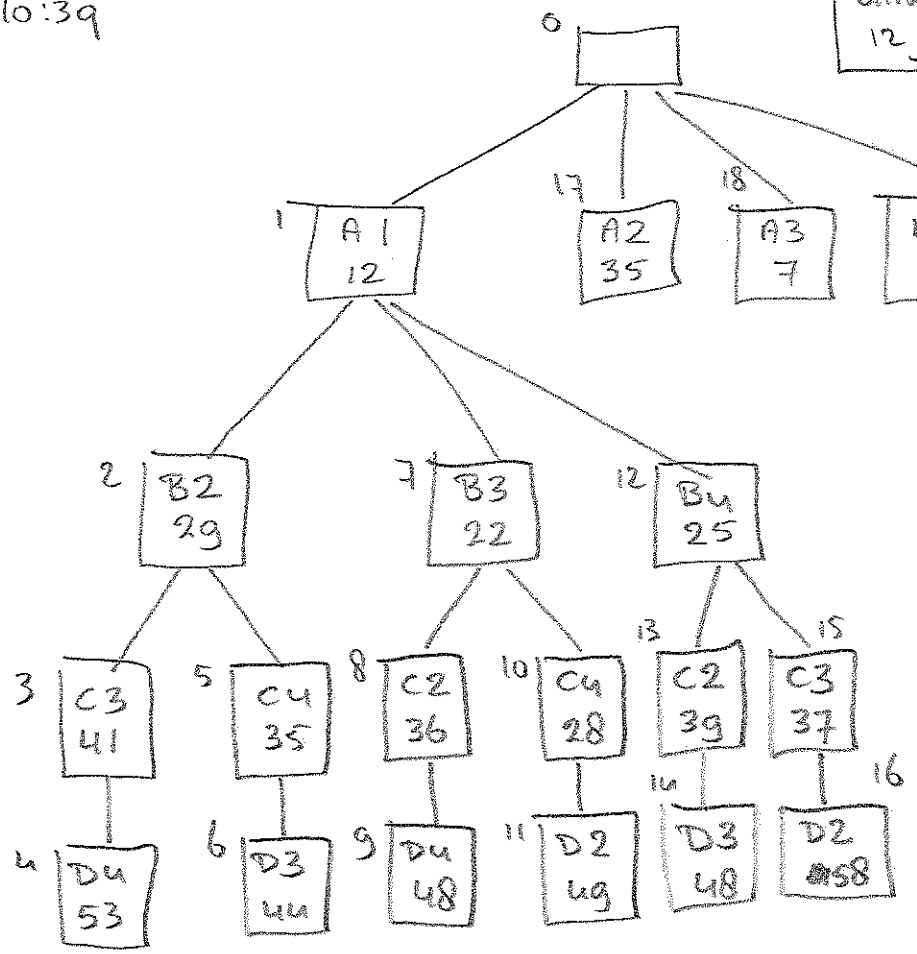
Bij elke toewijzing van een vestiging aan een klant kiezen we een klant.
In principe proberen we alle nog mogelijke klanten, in volgorde 1, 2, ..., n.

Als we bij een deeloplossing alle mogelijke uitbreidingen tot volledige oplossingen hebben bekeken, kiezen we voor de laatste vestiging in deze deeloplossing de volgende klant. Zo krijgen we volgende deeloplossing die we weer helemaal uitwerken, enz.

Bij elke deeloplossing houden we ook de kosten tot-dan-toe bij.
Als die al meer zijn dan de bezorgtijd van de beste complete oplossing tot dan toe, werken we de deeloplossing niet meer uit.

10:36

10:39




Nummers van A3 en A4 zullen bij compleet uitwerken van de boom hoger worden.

10:44

b) Branch and Bound

Ook hier worden oplossingen stap voor stap opgebouwd.
Bij iedere knoop / deeloplossing die je maakt, wordt een ondergrens berekend voor de mogelijke waarde van een complete oplossing die uit de deeloplossing gemaakt kan worden.
Als deze ondergrens $\geq$ de waarde van de beste complete oplossing tot dan toe is, wordt de deeloplossing niet uitgewerkt.
Ook wordt in principe bij elke deeloplossing gecontroleerd of hij nog wel uitgebreid kan worden tot een complete oplossing. Zo nee, dan wordt er niet eens een ondergrens berekend.
Er wordt ook geen ondergrens berekend als er nog maar één complete oplossing uit de deeloplossing te vormen is. In dat geval wordt die complete oplossing direct gemaakt, en de waarde daarvan berekend.
Ook als de deeloplossing zelf een complete oplossing is, wordt direct de waarde berekend.

Best Fit Branch & Bound

Iedere stap kies je de deeloplossing met de laagste ondergrens.
Deze deeloplossing wordt op alle mogelijke manieren één stap uitgebreid tot nieuwe (grotere) deeloplossingen, die op de hierboven beschreven wijze geëvalueerd worden.

10:54
10:56

Verschillen met backtracking

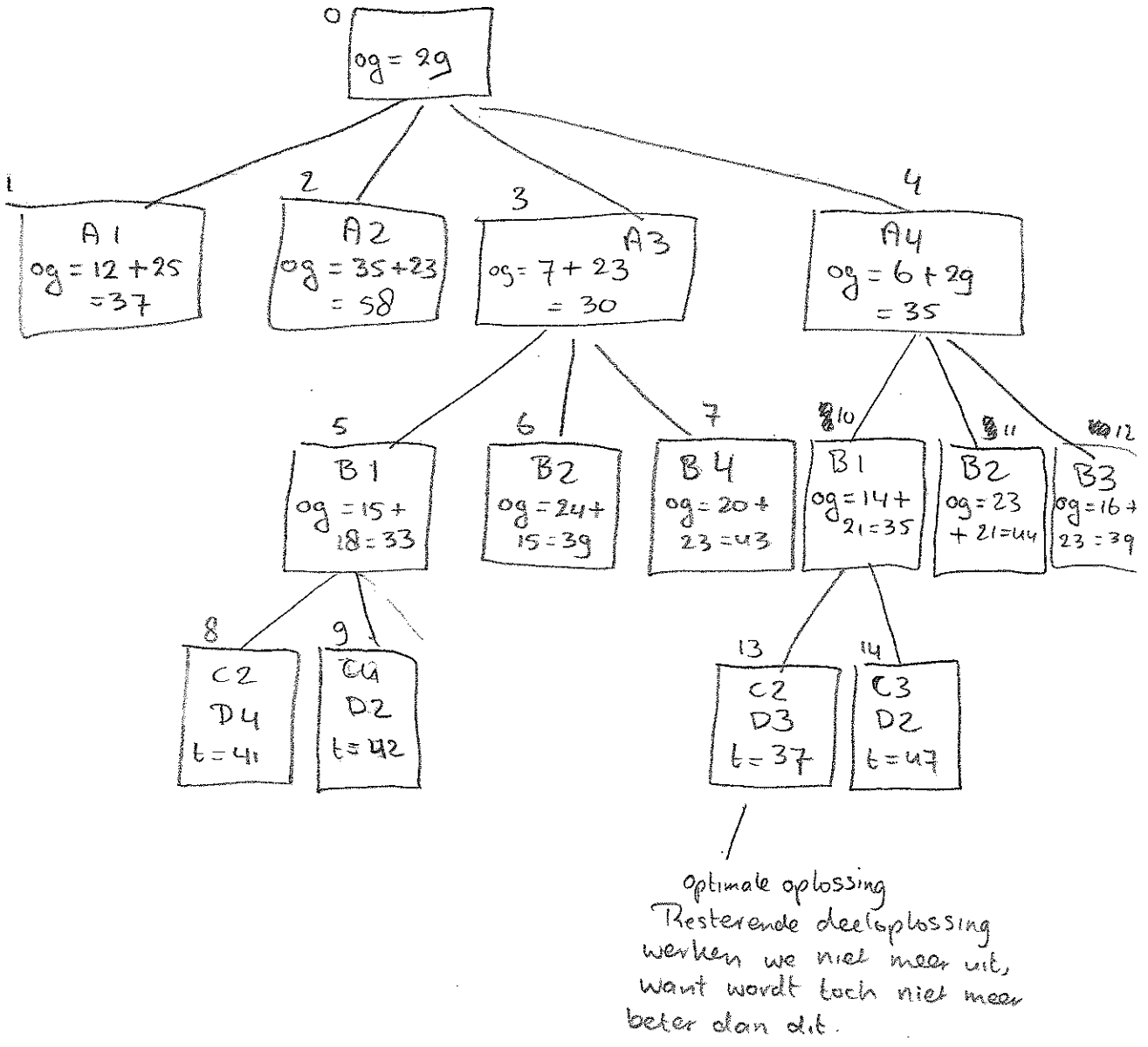
* er worden meerdere deeloplossingen tegelijkertijd bijgehouden, ipv. alleen (een pad naar) één deeloplossing
* dit kan doordat bij uitbreiding van een deeloplossing meteen al zijn kinderen worden gegenereerd, ipv alleen het eerst(volgend)e kind
* bij iedere deeloplossing wordt een ondergrens berekend (dit kan ook bij backtracking maar is niet standaard)
* deze ondergrens wordt niet alleen gebruikt om subbomen te snoeien (wat ook bij backtracking kan), maar ook om de zoekvolgorde te bepalen. De deeloplossing met de laagste ondergrens werken we uit

11:02

(c) We bouwen oplossingen dus in stappen op, zoals bij (a) beschreven

Als ondergrens gebruiken we:
tijd die van huidige deeloplossing +
voor elke resterende vestiging de laagst mogelijke tijd naar resterende klanten.

Bij dit probleem is elke deeloplossing nog uit te breiden tot complete oplossing, dus daar hoeven we niet op te controleren.




11:16
11:18

11:22.