16:30 **Uitwerking hertentamen Algoritmiek, woensdag 7 augustus 2013**

1 (a)

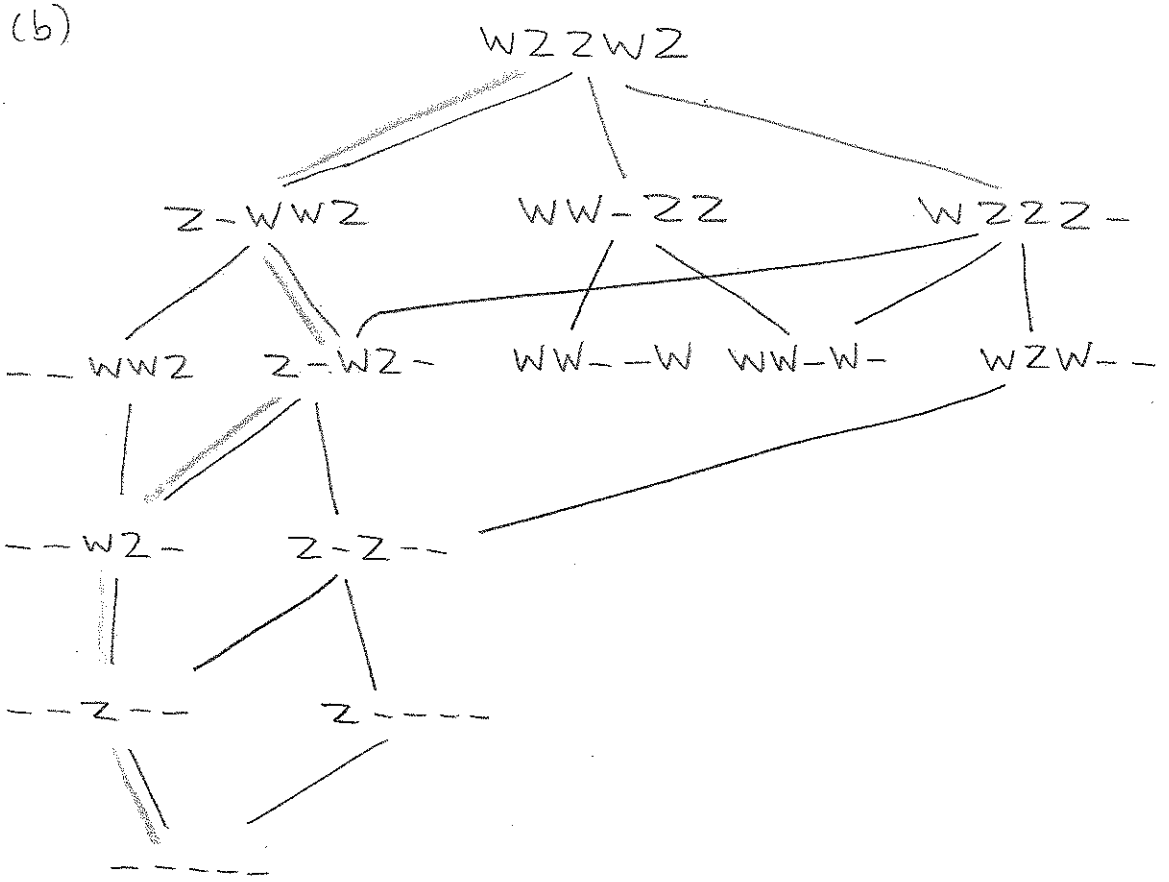
Toestanden: rijtjes van lengte n waarbij op elke positie een zwarte steen, een witte steen of geen steen ligt

Acties: het weghalen van een zwarte steen uit het rijtje, gevolgd door het omdraaien van de directe buurstenen (voor zover die er zijn)

Eindtoestanden: alle toestanden zonder zwarte stenen.

16:35

(b)



De rij kan weggespeeld worden, bijvoorbeeld door achtereenvolgens de stenen op posities 2, 5, 1, 4, 3 weg te halen (het rode pad in de toestand-actie-ruimte)

16:41

(c) In eerste instantie hebben we één aaneengesloten rij met n stenen, waarvan een oneven aantal zwarte stenen. Laat het aantal zwarte stenen m zijn.

We nemen nu de eerste zwarte steen vanaf links weg.

- Als links daarvan nog stenen liggen, dan waren die allemaal wit, maar nu is de meest rechtse steen in dat linker gedeelte zwart geworden. In het linker gedeelte van de rij is dus nu een oneven aantal (nl. 1) stenen zwart
- Als rechts van de weggehaalde zwarte steen nog stenen liggen, dan waren $m-1$ daarvan al zwart, want we hebben de eerste zwarte steen vanaf links weggehaald.

De eerste steen rechts van de weggehaalde steen is omgedraaid.
Als die zwart was, is die wit geworden, en zijn er 'rechts' nog m-2 zwarte stenen over
Als die wit was, is die zwart geworden, zodat er 'rechts' toch weer m zwarte stenen zijn.
In beide gevallen (m-2 of m) is het aantal zwarte stenen in het rechter gedeelte oneven, want m is oneven.

Na het weghalen van de eerste zwarte steen hebben we dus een aantal deelrijen met stenen (0, 1 of 2 deelrijen) die elk een oneven aantal zwarte stenen bevatten.

Zolang er nog deelrijen over zijn, halen we nu uit de eerste deelrij, die een oneven aantal zwarte stenen bevat, de eerste zwarte steen weg. Deze deelrij levert dan op zijn beurt 0, 1 of 2 nieuwe deelrijen op, elk met een oneven aantal zwarte stenen.

Ofwel: zolang er nog deelrijen zijn, bevat elke deelrij een oneven aantal zwarte stenen, en kunnen we nog een volgende steen weghalen. Het algoritme stopt pas als er geen deelrijen met stenen meer zijn, dwz: als alle stenen zijn weggespeeld.

17:03

(d) Neem maar de rij ZZWWWW. (in de eerste zet)
Bij de gretige strategie wordt ... de eerste zwarte steen weggehaald, en houden we -WWWWW over. Het spel stopt meteen, terwijl nog lang niet alle stenen zijn weggespeeld.
Inderdaad, nu werkt de gretige strategie niet.

17:07.

Uitwerking hertentamen Algoritmiek, woensdag 7 augustus 2013

11:18

2(a)

```
if (A[0] < A[1])
{ mini = 0;
  maxi = 1;
}
else
{ mini = 1;
  maxi = 0;
}
for (i=2; i<n; i++)
{ if (A[i] < A[mini])
    mini = i
  else
    if (A[i] > A[maxi])
      maxi = i
}
```

11:22

In het beste geval doet dit algoritme 1 vgl. vóór de for-lus + n-2 vgl in de for-lus (1 vgl. per iteratie, als we steeds een nieuw minimum hebben) ⇒ totaal n-1 vergelijkingen.

In het slechtste geval doet dit algoritme 1 vgl. vóór de for-lus + 2*(n-2) vgl. in de for-lus (2 vgl. per iteratie, als we geen enkele keer een nieuw minimum hebben) ⇒ totaal 2n-3 vergelijkingen

11:26

(b)

```
void selectie2 (int A[], int i, int & gr, int & kl)
{   // pre :  i >= 2 , even
    if (i==2)
      if (A[0] < A[1])
      { gr = 1;
        kl = 0;
      }
      else
      { gr = 0;
        kl = 1;
      }
    else  // i > 2
    { selectie2 (A, i-2, gr, kl);
      if (A[i-2] < A[i-1])
      { if (A[i-1] > A[gr])
          gr = i-1;
        if (A[i-2] < A[kl])
          kl = i-2;
      }
      else      // A[i-2] > A[i-1]
      { if (A[i-2] > A[gr])
          gr = i-2;
        if (A[i-1] < A[kl])
          kl = i-1;
      }
    }
```

11:34

```
}
```

(c)
```
void selectie3 (int A[], int links, int rechts, int &gr, int &kl)
  // pre : links < rechts ,  rechts-links+1 is 2-macht.
{
  if (rechts == links+1)   // maar 2 elementen
    if (A[links] < A[rechts]);
    { gr = rechts;
      kl = links;
    }
    else
    { gr = links;
      kl = rechts;
    }
  else     // rechts > links+1
  { midden = (links + rechts) / 2;
    selectie3 (A, links, midden, gr, kl);
    selectie3 (A, midden+1, rechts, gr2, kl2);
    if (A[gr2] > A[gr])
      gr = gr2;
    if (A[kl2] < A[kl])
      kl = kl2;
  }
}  // selectie3
```

11:42

Aantal vergelijkingen selectie2

$Vgl2(2) = 1$

$Vgl2(i) = Vgl2(i-2) + 3$

$\Rightarrow 1 + 3 * \frac{n-2}{2} = \frac{3}{2}n - 2$

Aantal vergelijkingen selectie3

$Vgl3(2) = 1$

$Vgl3(i) = 2 * Vgl3(i/2) + 2$

$\Rightarrow$	1,	4	10	22	46
$n =$	2,	4	8	16	32

$\Rightarrow \frac{3}{2}n - 2$

Niet gevraagd

11:49

3 (a)

```
bool vol (knoop *wortel)
// pre: wortel ≠ NULL
{
  if (wortel -> links == NULL  && wortel -> rechts == NULL) // blad
    return true
  else
    if (wortel -> links != NULL && wortel -> rechts != NULL) // 2 kinderen
      return ( vol (wortel -> links) && vol (wortel -> rechts));
    else      // 1 kind
      return false.
}
```

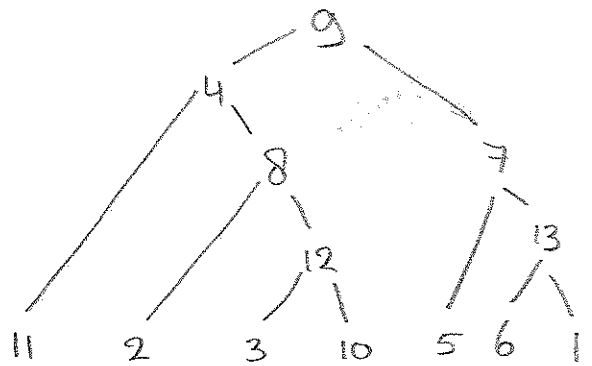
11:55

(b) Postorde-wandeling (LRW)

10 (0), 15 (0), 8 (1), 9 (0), 16 (1), 7 (0), 3 (0), 5 (1), 2 (1), 13 (0), 4 (0), 17 (0), 12 (1), 6 (0), 14 (1), 11 (1), 1 (1)

11:59

(c)



12:02

(d)

```
Void minmax (knoop* wortel, int niveau)
{ // pre: wortel ≠ NULL, subboom met ingang wortel is vol.
  if (wortel -> links != NULL)  // geen blad ⇒ 2 kinderen
  { minmax (wortel -> links, niveau +1) ;
    minmax (wortel -> rechts, niveau +1);
    if (niveau % 2 == 0)    // even ⇒ minimaliseren
      if (wortel -> links -> waarde < wortel -> rechts -> waarde)
        wortel -> waarde = wortel -> links -> waarde;
      else
        wortel -> waarde = wortel -> rechts -> waarde;
    else   // niveau is oneven ⇒ maximaliseren
      if (wortel -> links -> waarde > wortel -> rechts -> waarde)
        wortel -> waarde = wortel -> links -> waarde ;
      else
        wortel -> waarde = wortel -> rechts -> waarde;
  }
  // else: blad ⇒ waarde-veld is al bekend
} // minmax
```

12:13

4(a) Bij best-fit-first branch and bound bouwen we oplossingen stap voor stap op. Bij iedere deeloplossing die we genereren berekenen we onmiddelijk een bovengrens (want maximalisatieprobleem) voor de waarde die een complete oplossing kan hebben die uit de deeloplossing voortkomt (dit is 'bound').
Tijdens het algoritme pakken we steeds de deeloplossing met de hoogste bovengrens ('best-fit-first') en werken die uit:
Dat wil zeggen: we breiden de deeloplossing op alle mogelijke manieren een stap uit naar grotere deeloplossingen ('branch').
Bij al die nieuwe deeloplossingen berekenen we natuurlijk weer een bovengrens.

We breiden een deeloplossing niet verder uit:
* als we zien dat er geen geldige complete oplossing uit voort kan komen (de deeloplossing is infeasible)
* als er nog precies één complete oplossing uit voort kan komen. In dat geval construeren we deze complete oplossing, berekenen haar echte waarde (geen bovengrens) en als deze waarde hoger is dan de hoogste tot nu toe gevonden echte waarde, onthouden we die
* als de bovengrens van de deeloplossing ≤ hoogste tot nu toe gevonden echte waarde. Dan kan de deeloplossing toch geen hogere waarde meer opleveren.

12:30

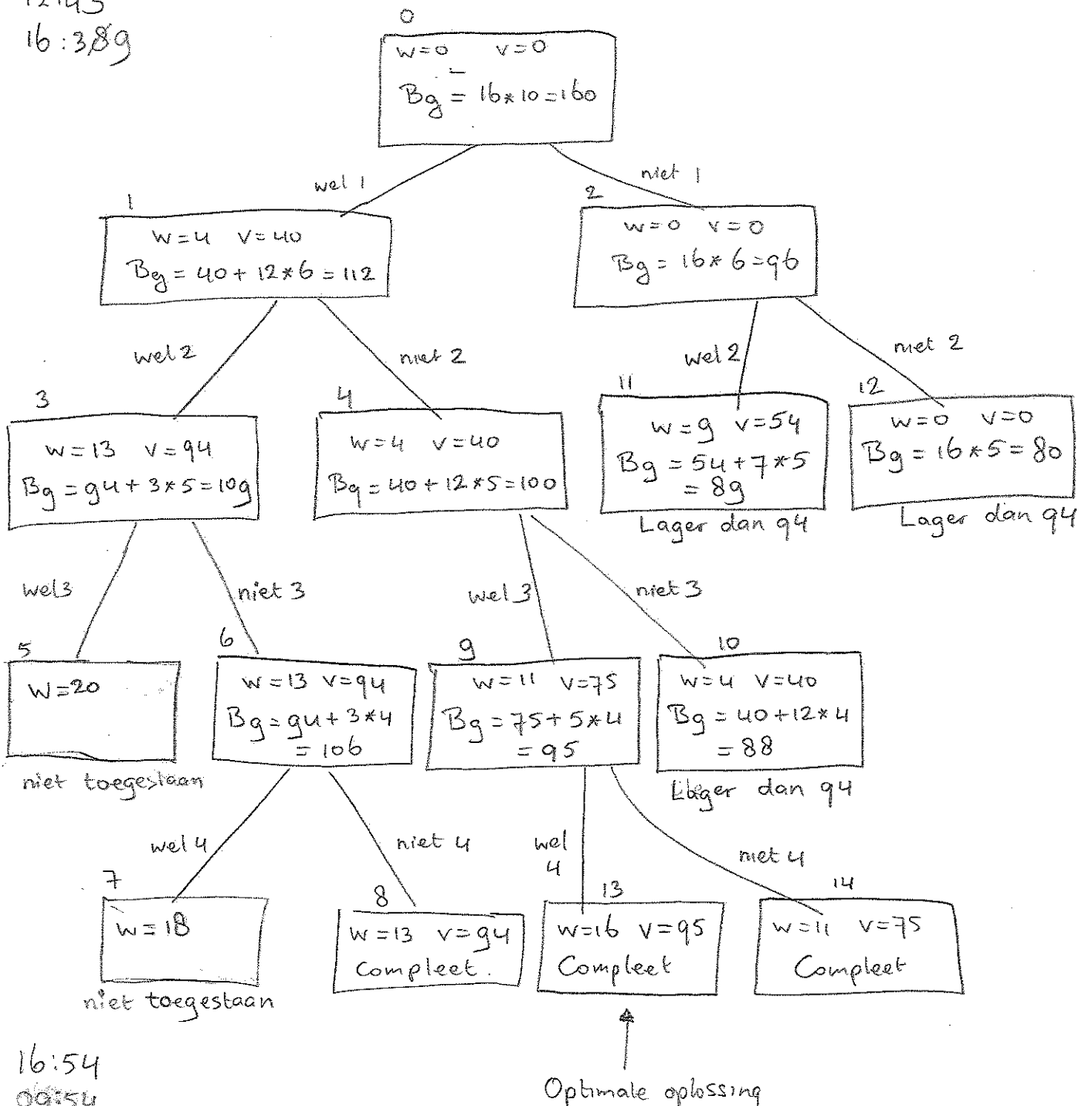
(b) De objecten zijn gesorteerd op v/w.
Bij aanvang is de knapzak leeg en is dus nog de hele capaciteit W beschikbaar. In het beste geval kunnen we hele knapzak vullen met objecten met de maximale waarde per gewicht v_1/w_1
Dat kan een totale waarde $W * \frac{v_1}{w_1}$ opleveren.

Na de i-de stap hebben we voor objecten 1 t/m i al bepaald of die wel / niet in de knapzak gaan. De (wel) gekozen objecten hebben totaalwaarde v en totaalgewicht w.
Van de totale capaciteit van de knapzak is nog $W-w$ over.
In het beste geval kunnen we die resterende capaciteit helemaal vullen met objecten met de maximale nog mogelijke waarde per gewicht. De maximale nog mogelijke waarde per gewicht is v_{i+1}/w_{i+1}, want we mogen alleen nog uit objecten $i+1$ t/m n kiezen (en de objecten zijn gesorteerd op v/w).

Als dit lukt, voegen we nog $(W-w) * \frac{v_{i+1}}{w_{i+1}}$ toe aan de waarde die we nu al in de knapzak hebben, zodat we totaal

$$v + (W-w) * \frac{v_{i+1}}{w_{i+1}}$$ zouden krijgen.

12:43
16:389



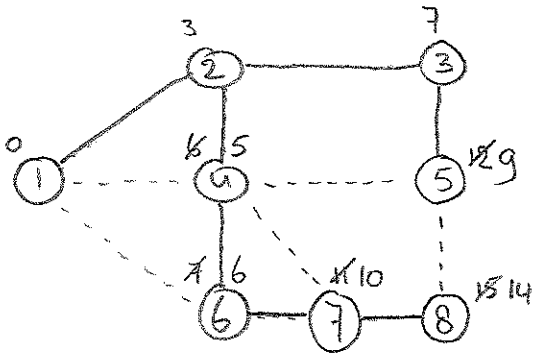
16:54
09:54
Controle.
09:59

5

We houden in een tabel bij wat de kortst mogelijke afstanden op een bepaald moment in het algoritme zijn, voor de op dat moment bereikbare knopen.

1	2	3	4	5	6	7	8
0	∞	∞	∞	∞	∞	∞	∞
-	3	∞	6	∞	7	∞	∞
-	-	7	5	∞	7	∞	∞
-	-	7	-	12	6	11	∞
-	-	7	-	12	-	10	∞
-	-	-	-	9	-	10	∞
-	-	-	-	-	-	10	15
-	-	-	-	-	-	-	14
-	-	-	-	-	-	-	-

Begin met knoop 1
Kies knoop 2, vanaf 1
Kies knoop 4, vanaf 2
Kies knoop 6, vanaf 4
Kies knoop 3, vanaf 2
Kies knoop 5, vanaf 3
Kies knoop 7, vanaf 6
Kies knoop 8, vanaf 7
Klaar



De doorgetrokken lijnen vormen de boom van kortste paden.

10:14
Controle
10:17