Uitwerking tentamen Algoritmiek, dinsdag 5 juni 2018

11.07

1(a)

* We keken alleen naar kringen die begonnen in een vaste knoop, zeg knoop a. Immers, als je een Hamiltonkring hebt, dan is het totaalgewicht onafhankelijk van de knoop vanwaaruit je die kring begint te doorlopen. Je loopt over dezelfde n takken. Je kunt dus zonder probleem een vaste beginknoop kiezen
* We keken alleen naar kringen waarin knoop b vóór knoop c bezocht wordt. Immers, je kunt een Hamiltonkring met startknoop a zowel in de ene richting ('linksom') als in de andere richting ('rechtsom') doorlopen. In de ene richting zal b vóór c bezocht worden, in de andere richting c vóór b. Omdat de graaf ongericht is, leveren beide richtingen hetzelfde totaalgewicht op

11.15 10:25

(b) Bij best-first branch-and-bound worden oplossingen stap voor stap opgebouwd. Bij minimalisatieproblemen wordt bij elke deeloplossing een ondergrens berekend voor de waarde van een complete oplossing die uit de deeloplossing kan voortkomen. Bij een complete oplossing wordt de waarde van de object-functie berekend, en (de waarde van) de beste complete oplossing wordt onthouden.

Tijdens het algoritme wordt steeds de deeloplossing met de laagste ondergrens als eerstvolgende uitgewerkt, door die uit te breiden met alle mogelijke waardes voor de volgende component. Al haar kinderen worden dus geconstrueerd, waarbij dan meteen ondergrenzen worden berekend.

uit alle nog openstaande deeloplossingen

Een deeloplossing wordt niet meer uitgewerkt, (er wordt gesnoeid),
* als ze niet meer aan de eisen van het probleem voldoet
* als er precies één geldige oplossing uit voort kan komen. In dat geval wordt die geldige oplossing meteen gemaakt.
* als haar ondergrens hoger of gelijk is aan de waarde van de beste, tot nu toe gevonden complete oplossing.

Branch slaat op het vertakken van een deeloplossing naar al haar kinderen

Bound slaat op het bepalen van een ondergrens voor elke deeloplossing.

Best-first slaat op de keuze van de deeloplossing met de laagste ondergrens, als eerstvolgende om uit te werken.

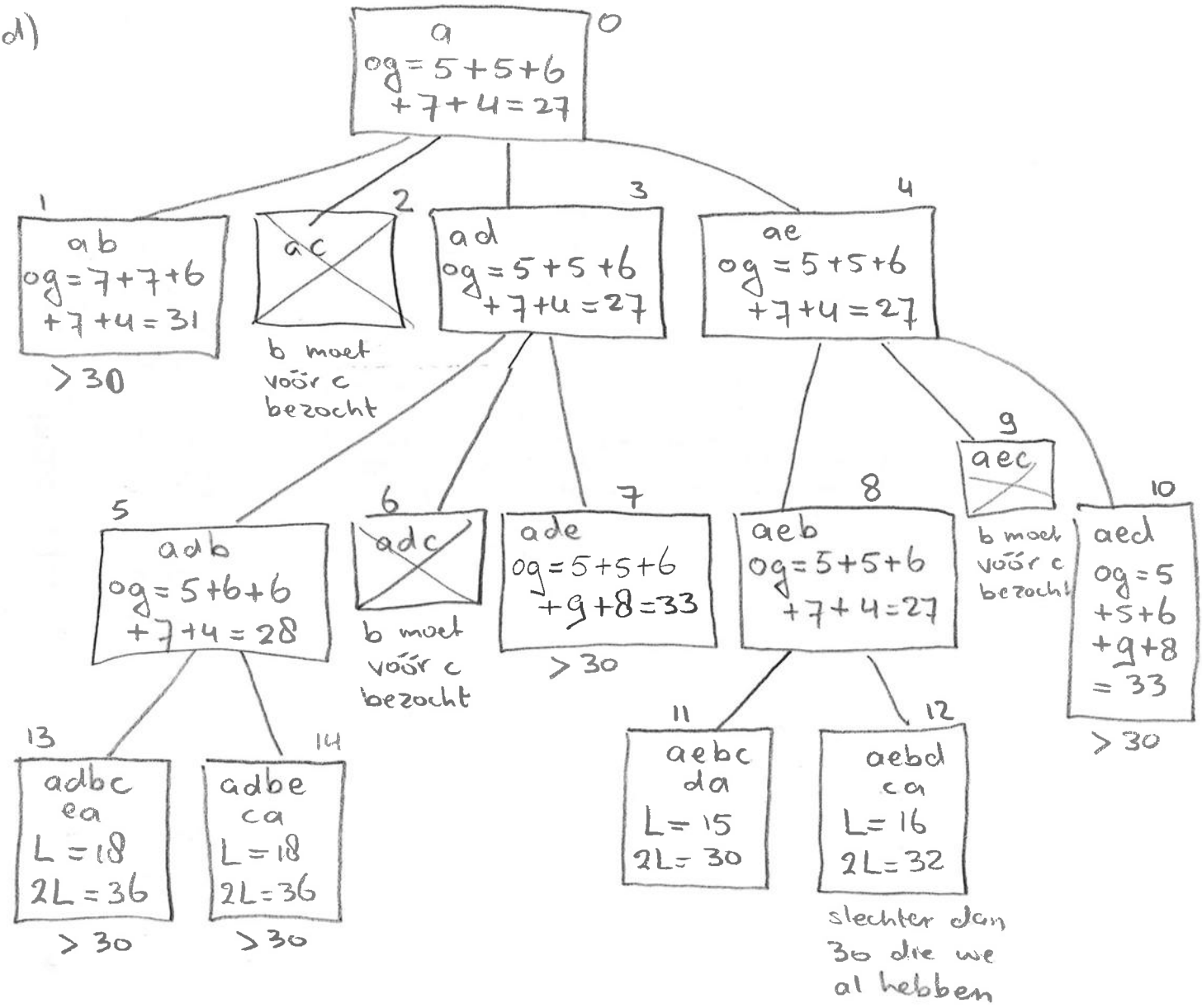
10.40.

11.15
(c)

Omdat het handelsreizigersprobleem een minimalisatieprobleem is, maken we gebruik van een ondergrens (zie (b)).

Voor elke knoop bepalen we twee 'aangrenzende' takken met de laagste gewichten. Die twee gewichten bij elkaar opgeteld vormt de bijdrage van de knoop. De ondergrens is nu gelijk aan de som van de bijdragen van alle knopen.

11.21
(d)



De optimale oplossing is dus aebcda (met lengte 15).

11.37.

2 (a)

```
void vulouderniveau (knoop *w, knoop *oud, int niv.)
{
    if (w ≠ NULL)
    { w -> ouder = oud;
      w -> niveau = niv;
      vulouderniveau (w -> links, w, niv+1);
      vulouderniveau (w -> rechts, w, niv+1);
    }
}
```

11.43

(b)

```
void vulnullafstand (knoop *w)
{
    if (w ≠ NULL)
    { vulnullafstand (w -> links);
      vulnullafstand (w -> rechts);
      if (w -> links == NULL || w -> rechk == NULL)
        w -> nullafstand = 0;
      else // beide kinderen niet NULL
      {
          if (w -> links -> nullafstand < w -> rechts -> nullafstand)
            w -> nullafstand = w -> links -> nullafstand + 1;
          else
            w -> nullafstand = w -> rechts -> nullafstand + 1;
      }
    }
}
```

11.48

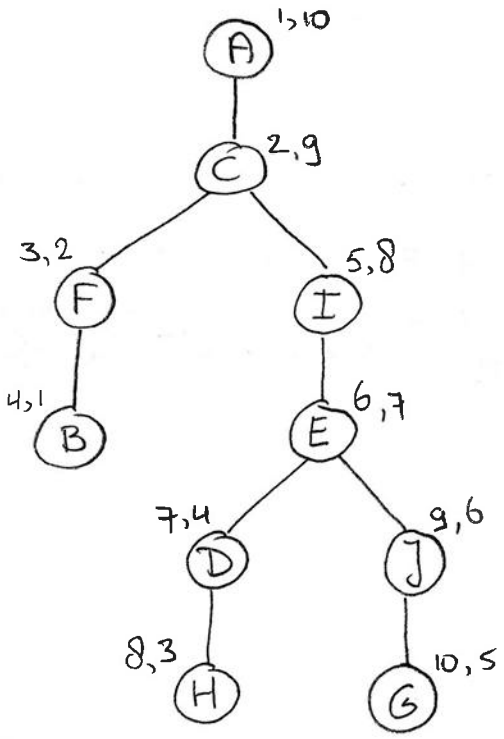
(c)

```
void voegtoe (knoop *w, knoop *nieuw)
{ knoop *loper;        int nieuwafstand;
  bool doorgaan;

  nieuw -> nullafstand = 0;

    // zoek plek om nieuw toe te voegen
  loper = w;
  while (loper -> nullafstand > 0)
  { if (loper -> links -> nullafstand < loper -> rechts -> nullafstand)
      loper = loper -> links;
    else
      loper = loper -> rechts;
  }
    // loper heeft nu nullafstand = 0, en dus
    // een NULL-pointer links of rechts
  if (loper -> links == NULL)
    loper -> links = nieuw;
  else
    loper -> rechts = nieuw;

  nieuw -> ouder = loper;
  nieuw -> niveau = loper -> niveau + 1;

    // pas nu, zonodig, nullafstand velden aan op het pad
    // naar de wortel
  doorgaan = true;
  while (loper ≠ NULL && doorgaan)
  {
    if (loper -> links == NULL || loper -> rechts == NULL)
      nieuwafstand = 0
    else
    {
      if (loper -> links -> nullafstand <
            loper -> rechts -> nullafstand)
        nieuwafstand = loper -> links -> nullafstand + 1;
      else
        nieuwafstand = loper -> rechts -> nullafstand + 1;
    }
    if (nieuwafstand == loper -> nullafstand)
      doorgaan = false
    else
    { loper -> nullafstand = nieuwafstand;
      loper = loper -> ouder;
    }
  } } //while
```

12.10

3(a)

De DFS-boom



Supersripts van een knoop zijn respectievelijk het nummer van eerste-keer-bereiken en het nummer van helemaal-afhandelen.

12.18 12.20

(b)

```
void dfs(v)
{
    teller++
    mark[v] = teller;
    for (alle buren w van v) do
        if (mark[w] == 0)    // nog niet bezocht
            dfs(w);
}
```

12.25

4 (a)

Als $j=0$, willen we van stad 0 naar stad 0.
Daarvoor hoeven we geen bus te pakken, zodat de kosten 0 zijn.

Als $j \geq 1$, hebben we wel één of meer bussen nodig. In onze busreis zullen we een keer een laatste bus pakken, die ons naar stad j voert. Laat i de stad zijn waarvandaan deze bus vertrekt. Dan moeten we eerst zo goedkoop mogelijk in stad i zien te komen. Dat kost kosten(i). Vervolgens moeten we de rechtstreekse bus van stad i naar stad j nemen. Dat kost prijs[i][j]. De totaalkosten zijn dan kosten(i) + prijs[i][j].
We weten à priori niet wat stad i is. Het kan elke stad vóór stad j zijn, dus $0 \leq i < j$. We nemen het minimum hiervan, omdat we de minimale kosten naar stad j willen hebben.

12.39

(b)

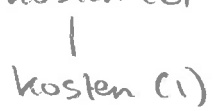
kosten(0)

n=0 : 1 aanroep, waar je met return 0 meteen uitspringt

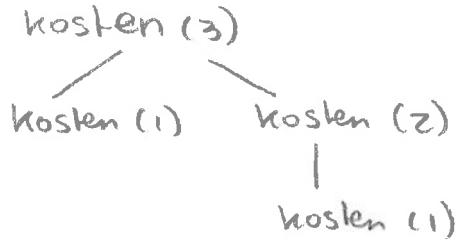
n=1 : 1 aanroep, de for-lus in de else heeft 0 iteraties

kosten(1)

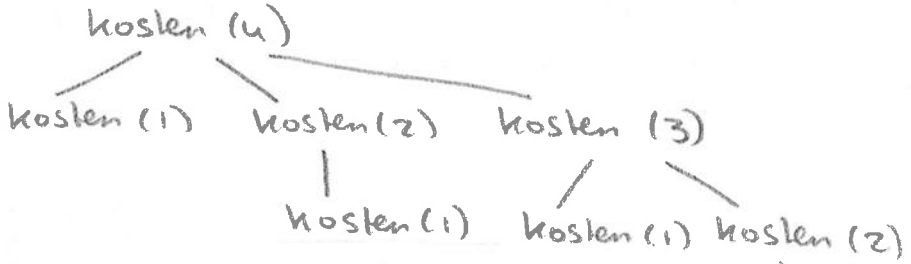
n=2 : 2 aanroepen :



n=3 : 4 aanroepen



n=4 : 8 aanroepen



12.46

(c)

```
int kostenDP (int n)
{ int i, j, temp, hulp;
  kosten [0] = 0;
  for (j = 1 ; j ≤ n ; j++ )
  { temp = prijs [0][j];    // i = 0
    for (i = 1 ; i < j ; i++)
    { hulp = kosten [i] + prijs [i][j];
      if (hulp < temp)
        temp = hulp;
    }
    kosten [j] = temp;
  }
  return kosten [n];
}
```

12.52

(d)

j=0 kosten [0] = 0

j=1 kosten [1] = min (0+12) = 12

j=2 kosten [2] = min (0+18, 12+5) = 17

j=3 kosten [3] = min (0+24, 12+13, 17+7) = 24

j=4 kosten [4] = min (0+31, 12+20, 17+13, 24+8) = 30

Hierbij heb ik dus de formule uit onderdeel (a) gebruikt, waarin ook kosten (0) + prijs [0][j] voorkomt.

12.57

(e) Een basisoperatie is de berekening van hulp:

```
hulp = kosten [i] + prijs [i][j];
```

Deze wordt in elke iteratie van de binnenste for-lus uitgevoerd.

Voor een gegeven j kent de binnenste for-lus j-1 iteraties (van i=1 t/m i=j-1). De variabele j doorloopt in de buitenste for-lus de waardes 1, 2, ..., n.

Dat levert dus achtereenvolgens (1-1), (2-1), ..., (n-1) ofwel 0, 1, ..., n-1 iteraties van de binnenste for-lus op.

De basisoperatie wordt dus

$$0 + 1 + \cdots + n-1 = \frac{1}{2} * n * (n-1) = \frac{1}{2}n^2 - \frac{1}{2}n$$

keer uitgevoerd. Dit is in $\Theta(n^2)$.
De tijdcomplexiteit is dus in $\Theta(n^2)$
Zou.