Uitwerking tentamen Algoritmiek, maandag 12 juni 2023

23.07

1(a)
```
bool voorwaardenOK (int n, int W, bool inDeelverz[], int totGewicht)
{ bool OK;
  if (totGewicht <= W)
  { OK = true
    for (int i=0; OK && i<=n-2; i++)
    { if (inDeelverz[i] && inDeelverz[i+1])
         OK = false;
    }
  }
  else
    OK = false;

  return OK;
}
```

23.13

(b)
```
int maxWaarde (int n, int i, int W, bool inDeelverz[], int totWaarde,
               int totGewicht, int waarde[], int gewicht)
{ int waarde1, waarde2;
  if (i==n) // complete deelverzameling ligt vast.
  { if (voorwaardenOK (n, W, inDeelverz, totGewicht))
       return totWaarde;
    else
       return -1; // slechter dan elke andere waarde
  }
  else // i<n
  { inDeelverz[i] = false; // bekijk geval dat object i niet
                           // in deelverzameling zit.
    waarde1 = maxWaarde (n, i+1, W, inDeelverz, totWaarde,
                         totGewicht, waarde, gewicht);
    inDeelverz[i] = true; // geval dat i wel in deelverzameling zit.
    waarde2 = maxWaarde (n, i+1, W, inDeelverz, totWaarde +
       waarde[i], totGewicht + gewicht[i], waarde, gewicht);
    return max (waarde1, waarde2);
  }
}
```

23.29

(c) Alle deelverzamelingen van $\{0,1,\ldots,n-1\}$ worden opgebouwd. Dat zijn er 2^n. Immers elk van de n objecten kan wel of niet in de deelverzameling zitten. Dus twee mogelijkheden per object. De twee mogelijkheden per object, kunnen op alle manieren met elkaar gecombineerd worden. Er zijn dus $2 \times 2 \times \ldots \times 2 = 2^n$ deelverzamelingen

Voor elke deelverzameling wordt de functie voorwaardenOK aangeroepen. Die vergt in het slechtste geval $\Theta(n)$ tijd, als de for-lus helemaal wordt afgemaakt, met n-1 iteraties.
In totaal krijgen we dus een worst case tijdcomplexiteit in $\Omega(n * 2^n)$ (Grote Ω, want er gebeurt natuurlijk nog meer dan het aanroepen van voorwaardenOK).

23.39

(d) Een functie die backtracking implementeert ziet er als volgt uit:

```
int maxWaarde2 (int n, int i, int W, bool inDeelverz[], int totWaarde,
      int totGewicht, int waarde[], int gewicht[])
{ int waarde1, waarde2;

  if (i==n)   // deelverzameling is compleet; controles zijn al gedaan.
    return totWaarde
  else
  { inDeelverz[i] = false;
    waarde1 = maxWaarde2(n, i+1, W, inDeelverz, totWaarde,
      totGewicht, waarde, gewicht);

    if (totGewicht + gewicht[i] <= W)
    { if (i==0 || !inDeelverz[i-1])  // we mogen object i toevoegen
                                     // aan deelverzameling
      { inDeelverz[i] = true;
        waarde2 = maxWaarde2 (n, i+1, W, inDeelverz, totWaarde
          + waarde[i], totGewicht + gewicht[i], waarde, gewicht);
      }
      else
        waarde2 = -1
    }
    else waarde2 = -1

    return max (waarde1, waarde2);
  }
} // maxWaarde2
```

We voeren dus al controles uit, als we een object aan de deelverzameling toevoegen. Dat hoeven we dan aan het eind niet meer te doen.

23.56

2(a)

(i)

top-down DP is recursief

bottom-up DP is iteratief

(ii)

bottom-up: alle deelproblemen worden opgelost.

top-down: alleen deelproblemen die nodig zijn. worden opgelost.

(iii)

bottom-up: vaak kan geheugenruimte worden uitgepaard

top-down: er kan geen geheugenruimte worden uitgespaard.

00.01

(b) {2, 3, 5}

{3, 4}

00.03

(c)

	j=0	1	2	3	4	5	6
i=0	1	0	0	0	0	0	0
1	1	0	0	1	0	0	0
2	1	1	0	1	1	0	0
3	1	1	0	1	1	1	1

00.09

(d)

$$F(i,j) = \begin{cases} 1 & \text{als } i \geq 0 \text{ en } j = 0 \\ 0 & \text{als } i = 0 \text{ en } j > 0 \\ 1 & \text{als } i > 0,\ j \geq v_i \text{ en } F(i-1, j-v_i) = 1 \\ F(i-1, j), & \text{anders} \end{cases}$$

De waarde $j=0$ is altijd te vormen door nul objecten in de deelverzameling te kiezen

Zonder objecten $(i=0)$ kun je geen positieve waarde j vormen

($F(i-1, j-v_i) = 1$:) We kiezen object i in de deelverzameling en gebruiken objecten $1..i-1$ voor de resterende waarde $j-v_i$

(anders:) We kiezen object i niet, en proberen waarde j met objecten $1..i-1$ te vormen.

00.21

(e)

```
int losOpSubsetSum (int n, int waarde [], int S)
{ int F[n+1][S+1];

   for (int i=0; i <= n; i++)
      F[i][0] = 1;
   for (int j=1; j <= S; j++)
      F[0][j] = 0;
```

```
  for (int i=1; i<=n; i++)
  { for (int j=1; j<=S; j++)
    { if ( j >= waarde[i] && F[i-1][j-waarde[i]] == 1 )
        F[i][j] = 1;      // kies object i
      else
        F[i][j] = F[i-1][j];     // kies object i niet
    } // for j
  } // for i

  return F[n][S]
}
```

00.31

00.38

3 (a)

Begintoestand:

* Voor elke rij i kiezen we de laagste waarde kosten[i][j] in die rij. Deze waardes van alle rijen bij elkaar tellen we op. Deze som is de ondergrens

Algemene deeloplossing

* Stel dat we al voor personen 1, 2, .., k een job hebben uitgekozen, en we nog voor personen k+1, .., n een job moeten kiezen:

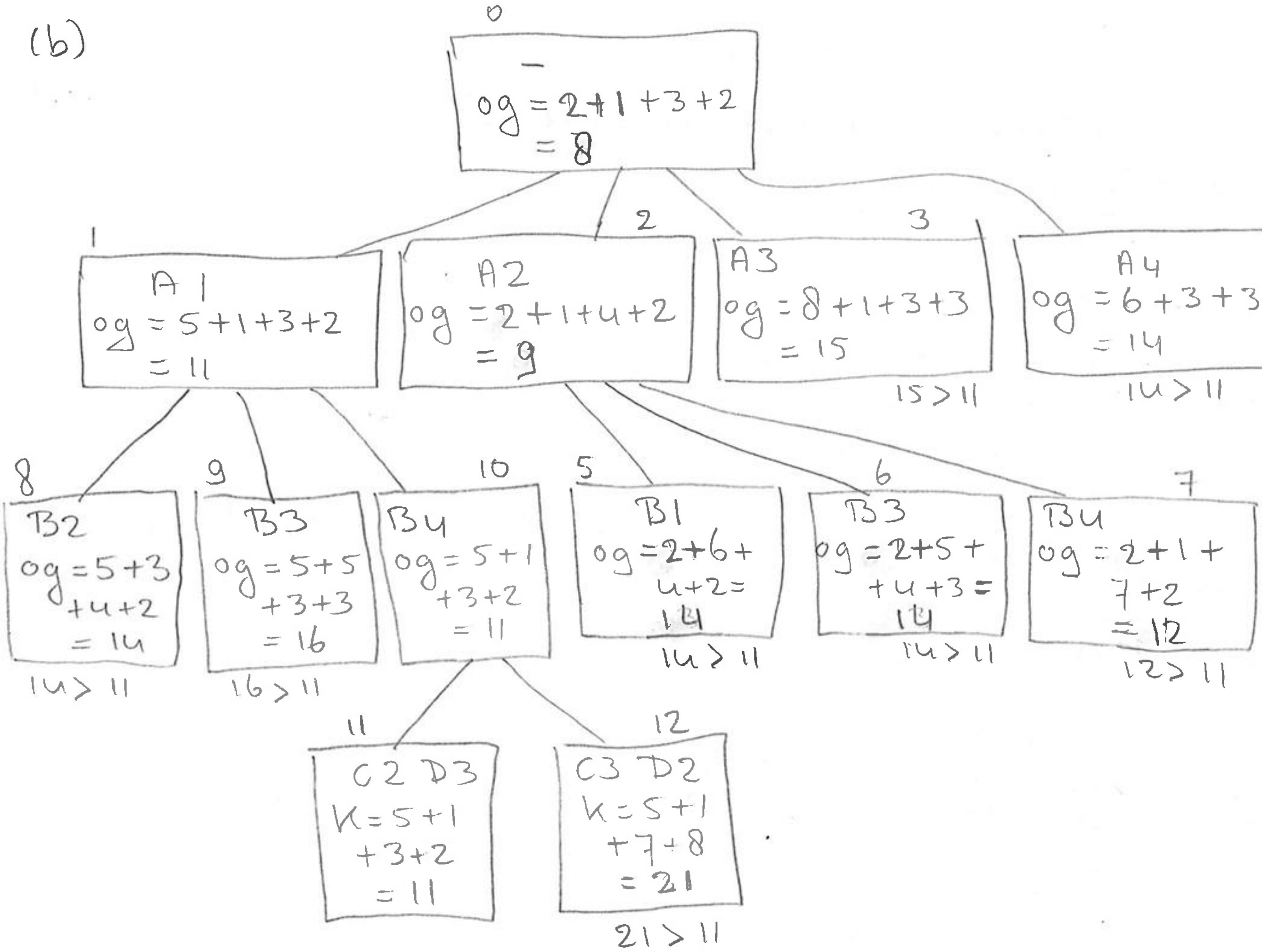
(a) Tel de kosten voor de reeds toegewezen k jobs bij elkaar op

(b) – Kijk voor elke rij/persoon i = k+1, .., n wat de laagste waarde kosten[i][j] in deze rij is in kolommen / bij jobs die nog niet zijn uitgekozen
Tel ook deze kosten bij elkaar op

* Tel de som bij (a) en de som bij (b) bij elkaar op. Dat is onze ondergrens.

00.47

(b)



Bij de knopen die niet verder zijn uitgewerkt (die zijn genoemd) staat de reden: de ondergrens is groter dan de kosten 11 van de complete oplossing die we hebben gevonden.

De optimale oplossing is dus A1, B4, C2, D3, met kosten 11

01.04 / 01.09

(c) Als we bij de berekening van de ondergrens per kolom zouden werken in plaats van per rij, wordt de ondergrens bij de begintoestand:

$4+2+2+1 = 9$

Dat is hoger dan de ondergrens 8 die we bij de berekening per rij vonden, en daarmee dus dichter bij de echte waarde.
Je zou met deze hogere ondergrens kunnen verwachten dat ondergrenzen verderop in de state space tree ook hoger zijn, zodat je meer knopen kunt snoeien, omdat hun ondergrens $\geq$ de kosten 11 zijn die we ook nu zullen vinden.
De ondergrens per kolom lijkt dus bruikbaarder.

01.20

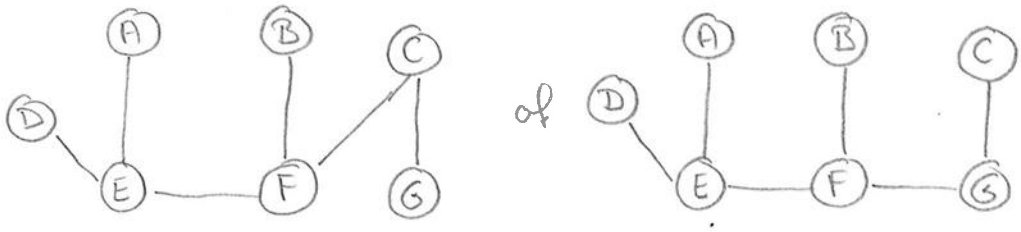
Uitwerking tentamen Algoritmiek, maandag 12 juni 2023

10.46

4) We passen het algoritme van Prim toe; op 'kladpapier':

A	B	C	D	E	F	G	actie
0	∞	∞	∞	∞	∞	∞	begin met A
–	9	∞	4	3	∞	∞	kies E, vanaf A
–	9	∞	2	–	7	∞	kies D, vanaf E
–	9	∞	–	–	7	∞	kies F, vanaf E
–	6	5	–	–	–	5	kies C, vanaf F (of G)
–	6	–	–	–	–	4	kies G, vanaf C (of C, vanaf G)
–	6	–	–	–	–	–	kies B, vanaf F (niet vanaf C, want tak FB is eerder bekeken dan CB)

Dit geeft



(a) volgordes 3 en 4 zijn goed

(b) (alleen) boom 1 is goed.

10.56