19.53

1(a) Een topologische ordening van de knopen in de graaf is een ordening van de knopen, zodat elke tak in de graaf loopt van een knoop eerder in de ordening naar een knoop later in de ordening.

19.56

(b)

```
begin met een lege ordening;
while (de DAG bevat nog knopen)
{ kies in de DAG een knoop x zonder inkomende takken;
  voeg x toe aan de ordening;
  verwijder x met zijn uitgaande takken uit de DAG;
}
```

19.58

(c) We tellen allereerst voor elke knoop in de DAG het aantal inkomende takken. Vervolgens correspondeert iedere iteratie van de while-lus met een rij in de tabel hieronder. De getallen in een rij zijn de aantallen resterende inkomende takken in de DAG. Als er een – staat, is de knoop al uit de DAG verwijderd:

	A	B	C	D	E	F	G	
iteratie 1:	1	1	0	3	2	0	3	verwijder C
" 2:	0	1	–	3	1	0	3	verwijder A
" 3:	–	0	–	2	1	0	3	verwijder B
" 4:	–	–	–	1	1	0	3	verwijder F
" 5:	–	–	–	1	0	–	2	verwijder E
" 6:	–	–	–	0	–	–	1	verwijder D
" 7:	–	–	–	–	–	–	0	verwijder G
								klaar

Als er meerdere knopen zonder inkomende takken zijn, verwijderen we de alfabetisch eerste.

De resulterende topologische ordening is dus: C, A, B, F, E, D, G.

20.07

(d) Laat $x_1 \to x_2 \to \cdots \to x_n \to x_1$ voor zekere $n \geq 2$ een kring zijn in de gerichte cyclische graaf, waarbij alle x_i's verschillend zijn. Bekijk een willekeurige ordening van de knopen in de graaf. Ook de knopen $x_1, x_2, \ldots, x_n$ hebben een plaats in die ordening. Laat x_{i_0} de eerste knoop uit onze kring in de ordening zijn. Dan staat x_{i_0-1} (of x_n als $x_{i_0} = x_1$) later in de ordening dan x_{i_0}, terwijl er wel een tak is van x_{i_0-1} (of x_n) naar x_{i_0}. De ordening is dus geen topologische ordening. Omdat onze ordening willekeurig was, kan er dus geen topologische ordening bestaan.

20.14

Uitwerking tentamen Algoritmiek, donderdag 6 juni 2019

22.58

2(a)

```
void vuloud (knoop *w)
{
   if (w ≠ NULL)
   { w -> oud = NULL;  // kan later door 'echte' waarde worden
     if (w -> links ≠ NULL)                    //overschreven.
     { vuloud (w -> links);
       w -> links -> oud = w;
     }
     if (w -> rechts ≠ NULL)
     { vuloud (w -> rechts);
       w -> rechts -> oud = w;
     }
   }
}
```

23.05.

(b)

```
void vulmaxim (knoop *w)
{
   if (w ≠ NULL)
   { w -> maxim = w -> info;
     if (w -> links ≠ NULL)
     { vulmaxim (w -> links);
       if (w -> links -> maxim > w -> maxim)
         w -> maxim = w -> links -> maxim;
     }
     if (w -> rechts ≠ NULL)
     { vulmaxim (w -> rechts);
       if (w -> rechts -> maxim > w -> maxim)
         w -> maxim = w -> rechts -> maxim;
     }
   }
}
```

23.09

Uitwerking tentamen Algoritmiek, donderdag 6 juni 2019

23.10
(c)

De algemene variant:

```
int verwijdermax (knoop *w)      // Pre: w is wortel van binaire
{ knoop *loper, *loper2          // boom met ≥ 2 knopen.
  int maxinfo;
  maxinfo = w->maxim;
  loper = w;
  while (loper->info ≠ maxinfo)
  { if ( loper->links ≠ NULL && loper->links->maxim == maxinfo)
      loper = loper->links;
    else
      loper = loper->rechts;
  }

  // knoop met maximale waarde gevonden
  // vind nu een willekeurig blad in zijn subboom
  loper2 = loper;
  while (loper2->links ≠ NULL || loper2->rechts ≠ NULL)
  { if (loper2->links ≠ NULL)
      loper2 = loper2->links;
    else
      loper2 = loper2->rechts;
  }

  // loper2 is nu een blad
  if (loper2 ≠ loper)   // maximale waarde stond niet in een blad
    loper->info = loper2->info;  // verplaats info-waarde

  loper = loper2->oud;
  if (loper2 == loper->links)    // knip loper2 uit de boom
    loper->links = NULL;
  else
    loper->rechts = NULL;
  delete loper2;
```

Z.O.Z.

```
    // loop terug omhoog en pas de maxim-velden aan
while (loper ≠ NULL)
{
   loper -> maxim = loper -> info;
   if (loper -> links ≠ NULL && loper -> links -> maxim > loper -> maxim)
     loper -> maxim = loper -> links -> maxim;
   if (loper -> rechts ≠ NULL && loper -> rechts -> maxim > loper -> maxim)
     loper -> maxim = loper -> rechts -> maxim;
    loper = loper -> oud;
}
return maxinfo;
}
```

07.37

3 (a)

Het toewijzingsprobleem is een minimalisatieprobleem, dus maken we bij **branch-and-bound gebruik** van een ondergrens.

07.40

(b)

Een algemene ondergrens is:

$$\text{huidigekosten} + \sum_{\substack{\text{personen } i \\ \text{die nog geen} \\ \text{job hebben} \\ \text{gekregen}}} (\text{de laagste waarde in rij } i \text{ van array kosten voor zover het om jobs } j \text{ gaat die nog beschikbaar zijn})$$

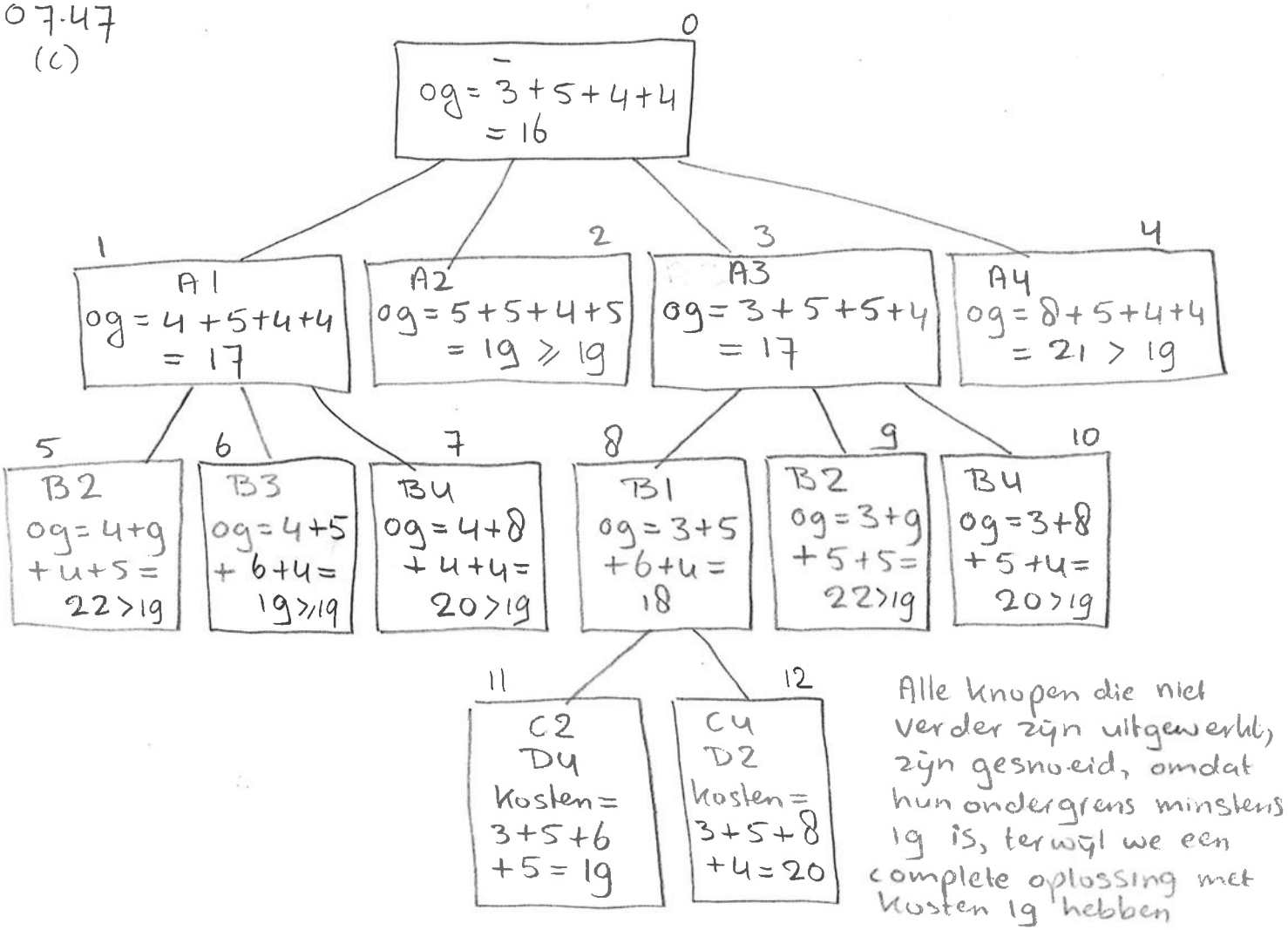
Hierbij is huidigekosten gelijk aan de kosten van de huidige gedeeltelijke toewijzing.

Voor de begintoestand is huidigekosten = 0, en wordt de ondergrens

$$\sum_{\text{alle personen } i} (\text{de laagste waarde in rij } i \text{ van array kosten})$$

07.47

(c)



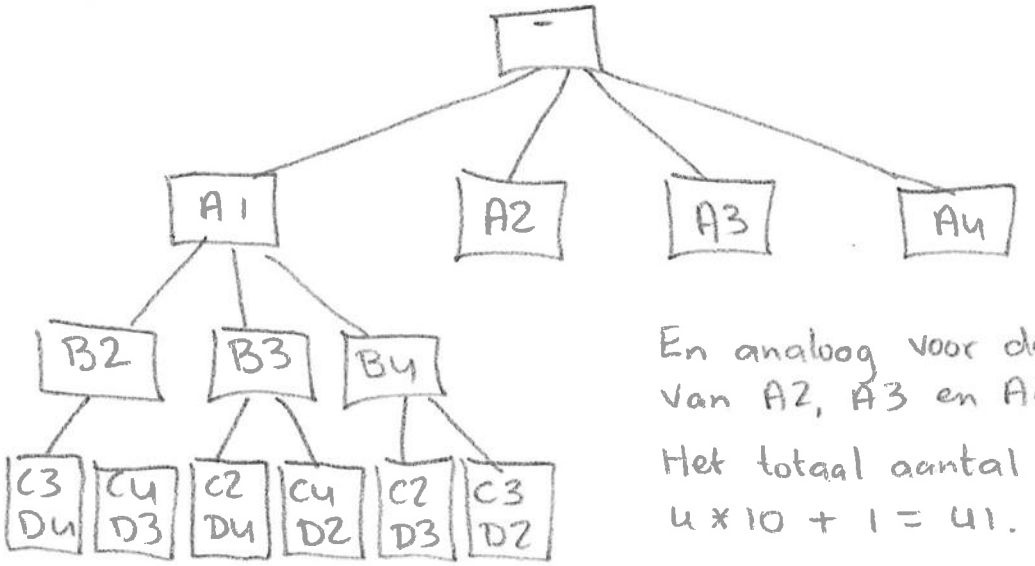
Alle knopen die niet verder zijn uitgewerkt, zijn gesnoeid, omdat hun ondergrens minstens 19 is, terwijl we een complete oplossing met kosten 19 hebben

De optimale oplossing is dus A3, B1, C2, D4 met totale kosten 19

08.02

(d)

In het slechtste geval ziet de state space tree bij best-first branch-and-bound voor een toewijzingsprobleem met n=4 er als volgt uit



En analoog voor de subbomen van A2, A3 en A4.

Het totaal aantal knopen is dan $4 * 10 + 1 = 41$.

08.08.

4 (a i)

stap 1: knoop A krijgt kleur 1, eerste knoop, eerste kleur
" 2: knoop B krijgt kleur 2, want 1 mag niet vanwege tak A-B
" 3: knoop C krijgt kleur 1, want kan gewoon
" 4: knoop D krijgt kleur 2, want 1 mag niet, vanwege tak A-D en C-D
" 5: knoop E krijgt kleur 1, want kan gewoon
" 6: knoop F krijgt kleur 3, want 1 kan niet vanwege tak E-F
2 kan niet, vanwege tak B-F en D-F
" 7: knoop G krijgt kleur 4, want 1 kan niet vanwege tak C-G
2 kan niet vanwege tak D-G
3 kan niet vanwege tak F-G

06.50

(a ii)

De for-lus kent N-1 iteraties

De instructie "kleur knoop i ..." is niet helemaal precies beschreven. Een mogelijke uitwerking van die instructie maakt gebruik van een boolean array OK, waarbij OK[m] aangeeft of kleur m nog mag voor knoop i, als volgt:

```
for (m = 1 to N)
  OK[m] = true;
for (j = 0 to i-1)
  if (graaf[i][j] == 1)  // tak i-j bestaat in de graaf
    OK[kleuring[j]] = false  // de kleur van j mag niet meer
m=1;
while (!OK[m])
  m++;
kleuring[i] = m;
```

Deze uitwerking bevat drie lussen, die respectievelijk N, i en (maximaal) i iteraties hebben. Elk van die iteraties vergt constante tijd, zodat de tijd complexiteit van de uitwerking hierboven in $\Theta(N+i+i)$ is, en dus in $\Theta(N)$.

Dit gebeurt iedere iteratie van de for-lus uit de opgave, zodat de totale tijdcomplexiteit in $\Theta(N^2)$ is.

07.06

07.09

(b)

```
void bepaalminkleuren (int N, int kleuring[], int i, int K,
                                    int & minkleuren)
{ int m, j; bool OK;

  if (i==N)  // totale graaf gekleurd
  { if (K < minkleuren)
      minkleuren = K;
  }
  else   // i < N
  { for (m=1; m <= K && K<minkleuren; m++)   // probeer kleur m voor knoop i,
    { OK = true;                             // als dat mag
      j=0;
      while (OK && j<i)
      { if (graaf[i][j]==1 && kleuring[j]==m)
          OK = false;     // kleur m mag niet
        else
          j++;
      }
      if (OK)
      { kleuring[i] = m;
        bepaalminkleuren (N, kleuring, i+1, K, minkleuren);
      }
    } // for
    // probeer de nieuwe kleur K+1, als dat tenminste nog zin
    // heeft voor het minimale aantal kleuren
    if (K+1 < minkleuren)
    { kleuring[i] = K+1;
      bepaalminkleuren (N, kleuring, i+1, K+1, minkleuren);
    }
  } // else. i<N
}
```

// het heeft nog zin om dit
// te proberen (in een
// vorige iteratie zou
// minkleuren gelijk aan K
// kunnen zijn geworden)

07.31