Hertentamen Algoritmiek
Dinsdag 19 juli 2022, 13.00 – 16.00 uur

Wanneer er in een opgave gevraagd wordt om uitleg, toelichting of motivatie van je antwoord, is het belangrijk om die ook te geven.
De aantallen punten die bij het begin van elke opgave vermeld worden, zijn indicatief. Ze kunnen dus nog iets wijzigen.
Veel succes!

---

1. [24 pt] Onze twee helden Jackie en Vivianne spelen een variant van het spel Nim. Er liggen bij het begin van het spel $n \geq 1$ lucifers op de tafel, en om de beurt mogen de twee spelers er 1, 2 of 3 weghalen. Hierbij is één beperking: als in de vorige beurt (van de andere speler dus) x lucifers zijn weggehaald, mogen er in de huidige beurt niet opnieuw x lucifers worden weggehaald. In de eerste beurt van de beginnende speler (wanneer er geen vorige beurt is) is deze beperking niet van toepassing.

   Het spel is afgelopen als er geen zet meer mogelijk is. In deze variant betekent dat niet per se dat alle lucifers van tafel zijn. De speler die op dat moment aan de beurt zou zijn, verliest. De andere speler wint dan.

   (a) Wat zijn voor dit spel de toestanden en de acties (voor algemene $n \geq 1$)?
   Wanneer is een toestand een eindtoestand? (misschien kun je deze subvraag gemakkelijker beantwoorden nadat je onderdeel (b) hebt gemaakt).

   We noemen een toestand *winnend* voor een speler als die speler gaat winnen bij optimaal spel van beide spelers vanaf die toestand.

   (b) Teken de complete toestand-actie-boom (ook de eindtoestanden) van dit spel voor het geval dat $n = 5$ en dat Jackie begint. Teken toestanden die (feitelijk) equivalent zijn, toch iedere keer opnieuw.
   Geef bij *elke* toestand ook aan of die winnend is voor J (Jackie) of voor V (Vivianne), te beginnen bij de eindtoestanden.

   (c) De algemene opzet voor een functie `winnend (...)`, die met behulp van brute force de toestand-actie-boom van een tweepersoonsspel afloopt om te bepalen of de huidige toestand (bij optimaal spel van beide spelers) winnend is voor de speler die nu aan de beurt is, ziet er als volgt uit:

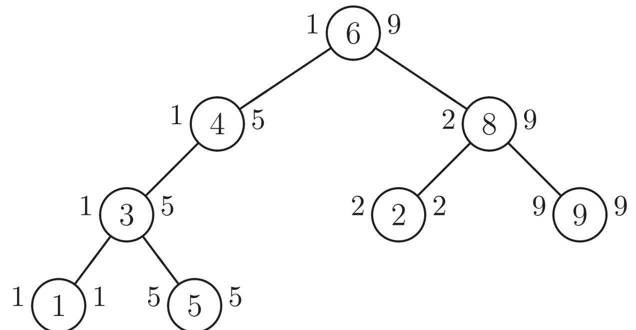
```
winnend (stand)
{
  if eindstand (stand) then
    ...
  else
    for alle mogelijke zetten i do
      kopie := stand;
      doezet (kopie,i);
      if not winnend (kopie) then
        return true;
      fi
    od
    return false;
  fi
}
```

Maak van deze algemene opzet een concrete C++-functie `bool winnend (...)` voor de variant van Nim die Jackie en Vivianne spelen. De functie moet dus gebruik blijven maken van brute force, maar moet wel stoppen met het aflopen van mogelijke zetten in een toestand, als er een winnende zet is gevonden.

---

2. [24 pt] Deze opgave gaat over binaire bomen, bestaande uit knopen uit de klasse `knoop` die er als volgt uitziet:

```
class knoop
{ public:
    knoop* links;
    knoop* rechts;
    int info;
    int mini;
    int maxi;
}; // knoop
```



In de boom hierboven zijn de getallen *in* de knopen de velden `info`. De getallen naast de knopen zijn de velden `mini` (links van de knoop) en `maxi` (rechts). Deze velden zijn bedoeld voor de minimale, respectievelijk maximale info-waarde in de subboom met die knoop als wortel. Voor een blad zijn deze waardes gewoon gelijk aan de info-waarde. Voor interne knopen is dit doorgaans niet het geval. Bijvoorbeeld de minimale info-waarde in de subboom met wortel 4 is 1, en de maximale info-waarde in die subboom is 5.

Bij aanvang hebben de velden `mini` en `maxi` in de boom overigens **nog geen zinvolle waarde**. Ze gaan gevuld worden bij onderdeel (b).

(a) Schrijf een *recursieve* C++-functie `void wandel (knoop *w)` die de info-waardes in de binaire boom (of subboom) met wortel `w` in symmetrische volgorde op het scherm afdrukt. Voor de boom hierboven betekent dat de volgorde 1, 3, 5, 4, 6, 2, 8, 9.

(b) Schrijf een *recursieve* C++-functie `void vulMiniMaxi (knoop *w)` die in elke knoop in de (sub)boom met wortel `w` de velden `mini` en `maxi` vult met de juiste waarde.

(c) Een binaire zoekboom is een binaire boom waarin alle info-waardes verschillend zijn, en waar voor elke knoop geldt dat alle knopen in de linker subboom een lagere info-waarde hebben dan de knoop zelf, en alle knopen in de rechter subboom een hogere info-waarde hebben. In het bijzonder is ook een lege boom per definitie een binaire zoekboom. De voorbeeldboom hierboven is echter geen binaire zoekboom.

Ga ervanuit dat alle velden `mini` en `maxi` in de boom de juiste waardes hebben. Schrijf een *recursieve* C++-functie `bool isBZBoom (knoop *w)` die controleert of de binaire (sub)boom met wortel `w` een binaire zoekboom is of niet. In het eerste geval wordt `true` geretourneerd, in het tweede geval `false`.

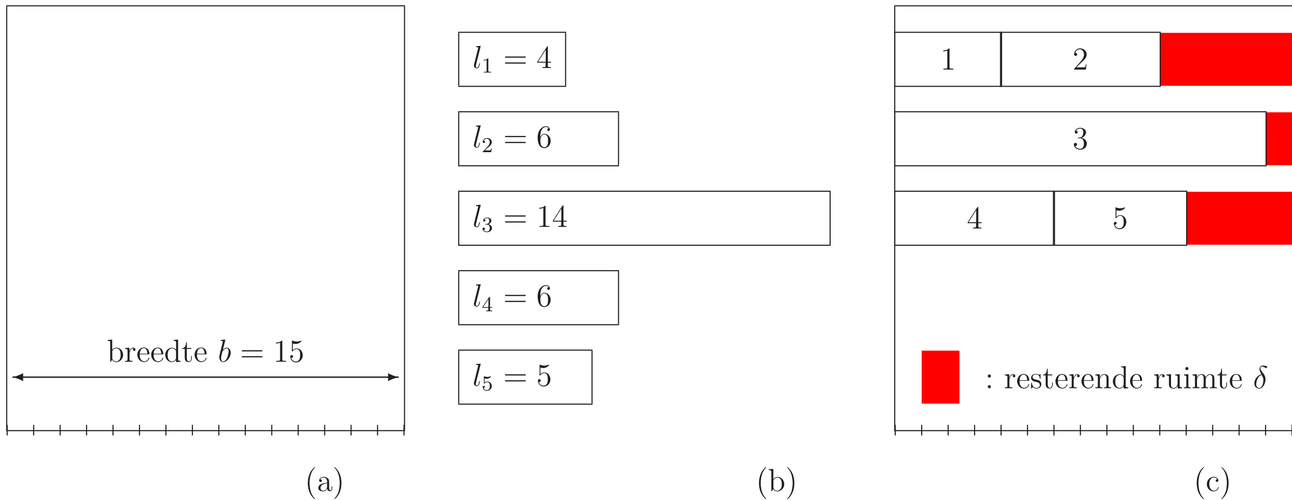
---

3. [34 pt] In Rotterdam hebben ze niet alleen een beroemde voetbalclub, maar ook een beroemde containerhaven. Daar moeten elke dag ingewikkelde logistieke problemen worden opgelost. Een daarvan is het bepalen van een optimale plaatsing van de containers die op vrachtwagens worden aangevoerd. Voordat ze op een schip gezet worden, worden ze met behulp van een kraan in rijen op het haventerrein geplaatst. De vraag is alleen: welke containers worden in welke rij gezet?

Er zijn N containers, genummerd 1 tot en met N. Ze komen één voor één aan in de containerhaven, in de volgorde van nummering, en in diezelfde volgorde worden ze op het haventerrein gezet.

Het haventerrein heeft een gegeven breedte $b \geq 1$, en de containers worden in rijen op het terrein opgesteld. Een rij bevat containers met opeenvolgende nummers: rij 0 bevat de eerste container(s), rij 1 de volgende container(s), enzovoort. Als container j in een bepaalde rij wordt gezet, komt container $j + 1$ dus in dezelfde rij, of in de volgende rij.

Elke container j heeft zijn eigen lengte $l_j \geq 1$. Alle getallen zijn integers en voor elke j geldt dat $l_j \leq b$. De totale lengte van een rij wordt gevormd door de som van de afzonderlijke containerlengtes. Deze som moet binnen de breedte van het haventerrein passen. Het haventerrein is groot genoeg voor alle rijen met containers. Elke rij begint zoveel mogelijk naar links. Zie Figuur 1 voor een voorbeeld.



Figuur 1: Het havenprobleem. (a) Het haventerrein. (b) Vijf containers, elk met hun eigen lengte. (c) Mogelijke plaatsing van de containers.

Vaak zal het niet lukken om met een rij containers de breedte van het haventerrein volledig te benutten. We proberen dit echter wel zo goed mogelijk te doen. We definiëren daarvoor zogenaamde *rijkosten*: als er aan het eind van een rij nog δ ruimte over is, dan bedragen de rijkosten voor die rij δ^2. Doordat we het kwadraat nemen, kan het handig zijn om een container die nog wel in een bepaalde rij past, toch niet in die rij te zetten, maar in de volgende rij, om te voorkomen dat in die volgende rij een grotere ruimte over blijft.

In het voorbeeld van Figuur 1 is δ voor rij 0 gelijk aan 5, voor rij 1 gelijk aan 1 en voor rij 2 gelijk aan 4. Dat geeft totale rijkosten van $5^2 + 1^2 + 4^2 = 42$. Het gaat erom om deze totale rijkosten te minimaliseren.

(a) Om het havenprobleem op te lossen, gaan we allereerst voor alle mogelijke i, j met $1 \leq i \leq j \leq N$ de waarde `rijkosten[i][j]` berekenen: de rijkosten voor een rij met daarin containers i tot en met j. Als de totale lengte van deze containers meer is dan de breedte b, moet `rijkosten[i][j]` gelijk worden aan `INT_MAX`.

Geef een C++-functie `void berekenRijkosten (int b, int N, int lengte[])` die array `rijkosten` vult met de juiste waardes. Je mag ervan uitgaan dat `rijkosten` een globaal, twee-dimensionaal array is dat groot genoeg is om alle waardes `rijkosten[i][j]` te bevatten. Het array `lengte` bevat de containerlengtes: `lengte[i]` $= l_i$, voor $1 \leq i \leq N$.

Probeer ervoor te zorgen dat de tijdcomplexiteit van de functie kwadratisch is in N. Als dit niet lukt, kun je nog wel het grootste deel van de punten verdienen.

(b) We gaan nu ook naar deelproblemen van het havenprobleem kijken. Voor $0 \leq j \leq N$ definiëren we *totaalkosten*(j): de minimale totale rijkosten voor het havenprobleem

met alleen containers 1 tot en met j.

Geef een recurrente betrekking waar $totaalkosten(j)$ aan voldoet. Dat wil zeggen: druk $totaalkosten(j)$ uit in $totaalkosten(i)$ voor $i < j$. Licht ook toe waarom $totaalkosten(j)$ aan de betrekking voldoet. Denk aan het basisgeval en aan de recursieve stap van de recurrente betrekking.

Hint: vraag je af welke containers allemaal bij container j in de laatste rij kunnen staan.

Als je het antwoord op dit onderdeel niet weet, dan kun je het 'kopen' van de docent. Wellicht kun je dan wel de hierna volgende onderdelen maken.

(c) Bij dynamisch programmeren slaan we alle waardes $totaalkosten(j)$ op in een tabel (array) `totaalkosten`. We noemen het dan `totaalkosten[j]` (met blokhaken).

Geef een niet-recursieve C++-functie `int bepaalTotaalkosten (int b, int N, int lengte[])` die met behulp van bottom-up dynamisch programmeren, gebruikmakend van de recurrente betrekking uit onderdeel (b), de waarde `totaalkosten[N]` berekent (en retourneert). Je mag ervan uitgaan dat het globale, tweedimensionale array `rijkosten` uit onderdeel (a) al is ingevuld.

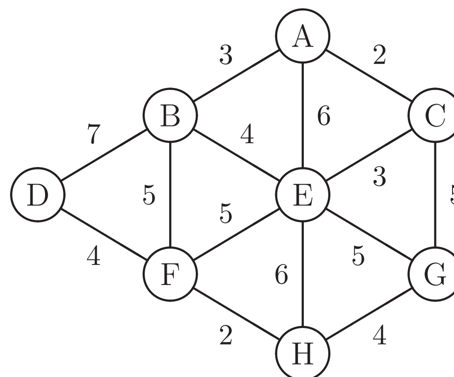
(d) Wat is de worst-case tijdcomplexiteit van je functie `bepaalTotaalkosten` uit onderdeel (c)? Motiveer je antwoord door een basisoperatie aan te wijzen en te berekenen hoe vaak die operatie wordt uitgevoerd.

---

4. [18 pt]

(a) Pas het algoritme van Dijkstra toe op onderstaande graaf, beginnend in knoop A. Geef voor elke stap van het algoritme duidelijk aan welke knoop erbij wordt gekozen in U (= verzameling knopen waarvan de kortste afstand vanaf A bekend is) en welke labels door die keuze veranderen en hoe. Licht toe hoe je uitwerking gelezen moet worden.

Geef ook de resulterende boom van kortste paden met daarin de bijbehorende lengtes van de kortste paden vanaf A.



(b) Geef een voorbeeld waaruit blijkt dat het algoritme van Dijkstra niet altijd werkt voor een samenhangende gewogen graaf met negatieve gewichten. Dat wil zeggen: geef een samenhangende gewogen graaf met minstens één negatief gewicht, waarbij het algoritme van Dijkstra vanuit een bepaalde startknoop niet voor alle andere knopen het kortste pad vindt.

Teken ook de boom met 'kortste paden' die het algoritme oplevert, en laat zien dat een van de paden vanaf de startknoop in die boom toch niet een kortste pad is.

Een graaf met drie knopen is al genoeg, maar meer mag ook. De graaf mag geen *kringen* met negatief totaalgewicht bevatten.