

be-OI 2022

Finale - JUNIOR
zaterdag 23 april 2022

Invullen in **HOOFDLETTERS** en **LEESBAAR** aub

VOORNAAM :
NAAM :
SCHOOL :

O

Gereserveerd

Finale van de Belgische Informatica-olympiade (duur : maximum 2u)

Algemene opmerkingen (lees dit aandachtig voor je begint)

- Controleer of je de juiste versie van de vragen hebt gekregen (die staat hierboven in de hoofding).
 - De categorie **beloften** is voor leerlingen tot en met het 2e middelbaar,
 - de categorie **junior** is voor het 3e en 4e middelbaar,
 - de categorie **senior** is voor het 5e middelbaar en hoger.
- Vul duidelijk je voornaam, naam en school in, **alleen op dit eerste blad**.
- Jouw antwoorden moet je invullen op de daarop voorziene antwoordbladen, die je achteraan vindt.
- Als je door een fout buiten de antwoordkaders moet schrijven, schrijf dan alleen verder op hetzelfde blad papier (desnoods op de achterkant).
- Schrijf **duidelijk leesbaar** met blauwe of zwarte **pen of balpen**.
- Je mag alleen schrijfgerief bij je hebben. Rekentoestel, GSM, ... zijn **verboden**.
- Je mag altijd extra kladpapier vragen aan de toezichthouder of leerkracht.
- Wanneer je gedaan hebt, geef je deze eerste bladzijde terug (met jouw naam erop), en de pagina's met jouw antwoorden. Al de rest mag je bijhouden.
- Voor alle code in de opgaven werd **pseudo-code** gebruikt. Op de volgende bladzijde vind je een **beschrijving** van de pseudo-code die we hier gebruiken.
- Als je moet antwoorden met code, mag dat in **pseudo-code** of in eender welke **courante programmeertaal** (zoals Java, C, C++, Pascal, Python, ...). We trekken geen punten af voor syntaxfouten.

Veel succes!

De Belgische Informatica Olympiade wordt mogelijk gemaakt dankzij de steun van onze leden:



©2022 Belgische Informatica-olympiade (beOI) vzw
Dit werk is vrijgegeven onder de licentie: Creative Commons Naamsvermelding 2.0 België

Overzicht pseudo-code

Gegevens worden opgeslagen in variabelen. Je kan de waarde van een variabele veranderen met \leftarrow . In een variabele kunnen we gehele getallen, reële getallen of arrays opslaan (zie verder), en ook booleaanse (logische) waarden : waar/just (**true**) of onwaar/fout (**false**). Op variabelen kan je wiskundige bewerkingen uitvoeren. Naast de klassieke operatoren $+$, $-$, \times en $/$, kan je ook $\%$ gebruiken: als a en b allebei gehele getallen zijn, dan zijn a/b en $a\%b$ respectievelijk het quotiënt en de rest van de gehele deling (staartdeling). Bijvoorbeeld, als $a = 14$ en $b = 3$, dan geldt: $a/b = 4$ en $a\%b = 2$. In het volgende stukje code krijgt de variabele *leeftijd* de waarde 17.

```
geboortejaar  $\leftarrow$  2003
leeftijd  $\leftarrow$  2020 - geboortejaar
```

Als we een stuk code alleen willen uitvoeren als aan een bepaalde voorwaarde (conditie) is voldaan, gebruiken we de instructie **if**. We kunnen eventueel code toevoegen die uitgevoerd wordt in het andere geval, met de instructie **else**. Het voorbeeld hieronder test of iemand meerderjarig is, en bewaart de prijs van zijn/haar cinematicket in een variabele *prijs*. De code is bovendien voorzien van commentaar.

```
if (leeftijd  $\geq$  18)
{
    prijs  $\leftarrow$  8 // Dit is een stukje commentaar
}
else
{
    prijs  $\leftarrow$  6 // Goedkoper!
}
```

Soms, als een voorwaarde onwaar is, willen we er nog een andere controleren. Daarvoor kunnen we **else if** gebruiken, wat neerkomt op het uitvoeren van een andere **if** binnen in de **else** van de eerste **if**. In het volgende voorbeeld zijn er 3 leeftijdscategorieën voor cinematickets.

```
if (leeftijd  $\geq$  18)
{
    prijs  $\leftarrow$  8 // Prijs voor een volwassene.
}
else if (leeftijd  $\geq$  6)
{
    prijs  $\leftarrow$  6 // Prijs voor een kind van 6 of ouder.
}
else
{
    prijs  $\leftarrow$  0 // Gratis voor kinderen jonger dan 6.
}
```

Wanneer we in één variabele tegelijk meerdere waarden willen stoppen, gebruiken we een array. De afzonderlijke elementen van een array worden aangeduid met een index (die we tussen vierkante haakjes schrijven achter de naam van de array). Het eerste element van een array *arr* heeft index 0 en wordt genoteerd als *arr*[0]. Het volgende element heeft index 1, en het laatste heeft index $N - 1$ als de array N elementen bevat. Dus als de array *arr* de drie getallen 5, 9 en 12 bevat (in die volgorde) dan is *arr*[0] = 5, *arr*[1] = 9 en *arr*[2] = 12. De lengte van *arr* is 3, maar de hoogst mogelijke index is slechts 2.

Voor het herhalen van code, bijvoorbeeld om de elementen van een array af te lopen, kan je een **for**-lus gebruiken. De notatie **for** ($i \leftarrow a$ to b step k) staat voor een lus die herhaald wordt zolang $i \leq b$, waarbij i begint met de waarde a en telkens verhoogd wordt met k aan het eind van elke stap. Het onderstaande voorbeeld berekent de som van de elementen van de array arr , veronderstellend dat de lengte ervan N is. Nadat het algoritme werd uitgevoerd, zal de som zich in de variabele sum bevinden.

```
sum ← 0
for (i ← 0 to N - 1 step 1)
{
    sum ← sum + arr[i]
}
```

Een alternatief voor een herhaling is een **while**-lus. Deze herhaalt een blok code zolang er aan een bepaalde voorwaarde is voldaan. In het volgende voorbeeld delen we een positief geheel getal N door 2, daarna door 3, daarna door 4 ... totdat het getal nog maar uit 1 decimaal cijfer bestaat (d.w.z., kleiner wordt dan 10).

```
d ← 2
while (N ≥ 10)
{
    N ← N/d
    d ← d + 1
}
```

We tonen algoritmes vaak in een kader met wat extra uitleg. Na **Input**, definiëren we alle parameters (variabelen) die gegeven zijn bij het begin van het algoritme. Na **Output**, definiëren we de staat van bepaalde variabelen nadat het algoritme is uitgevoerd, en eventueel de waarde die wordt teruggegeven. Een waarde teruggeven doe je met de instructie **return**. Zodra **return** wordt uitgevoerd, stopt het algoritme en wordt de opgegeven waarde teruggegeven.

Dit voorbeeld toont hoe je de som van alle elementen van een array kan berekenen.

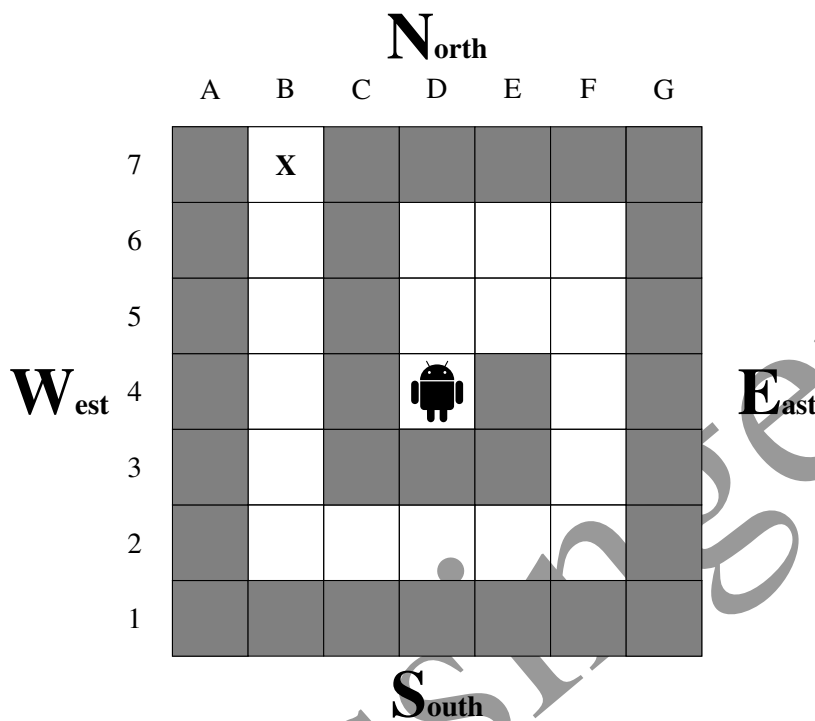
```
Input : arr, een array van  $N$  getallen.
         $N$ , het aantal elementen van de array.
Output : sum, de som van alle getallen in de array.

sum ← 0
for (i ← 0 to N - 1 step 1)
{
    sum ← sum + arr[i]
}
return sum
```

Opmerking: in dit laatste voorbeeld wordt de variabele i enkel gebruikt om de tel bij te houden van de **for**-lus. Er is dus geen uitleg voor nodig bij **Input** of **Output**, en de waarde ervan wordt niet teruggegeven.

Vraag 1 – Nono de kleine robot

Nono de kleine robot zit vast in het midden van het labirint.



Nono kan zich enkel verplaatsen naar de witte cellen, maar niet naar de grijze cellen (de grijze cellen stellen muren voor). In het begin zit Nono in het midden van het labirint in de cel met coördinaten **D4**. Hij moet de uitgang bereiken: dit is de cel met coördinaten **B7** en is aangeduid met een **X**.

Men kan Nono één cel verplaatsen per keer met het commando `Go()`. Dit commando verwacht één parameter die het volgende kan zijn: N, S, E of W. Elke parameter stelt één van de vier windrichtingen voor die op deze figuur staan aangeduid, waarbij het noorden zoals gewoonlijk vanboven staat.

Op de volgende pagina stellen we meerder algoritmes voor om Nono uit het labirint te laten ontsnappen. We willen voor elk algoritme weten of Nono zich op het einde na de uitvoer van het programma op de cel **X** bevindt of als hij botst tegen een muur (en in dat geval vragen we de coördinaten van de muur waartegen hij gebotst is).

Sommige algoritmes gebruiken arrays. Bijvoorbeeld `Dir←[N,E,S,W,N]` uit algoritme 4 initialiseert een array met 5 elementen die gelijk zijn aan `Dir[0]=N`, `Dir[1]=E`, `Dir[2]=S`, `Dir[3]=W` en `Dir[4]=N`.

Algorithme 1

```

Go(N)
for (i ← 1 to 2 step 1)
{
  Go(E)
}
for (i ← 1 to 3 step 1)
{
  Go(S)
}
for (i ← 1 to 4 step 1)
{
  Go(W)
}
for (i ← 1 to 5 step 1)
{
  Go(N)
}

```

Algorithme 2

```

Go(N)
i ← 1
while (i ≤ 2)
{
  Go(E)
  i ← i+1
}
while (i ≥ 0)
{
  Go(S)
  i ← i-1
}
while (i ≤ 4)
{
  Go(W)
  i ← i+1
}
i ← 0
while (i < 5)
{
  Go(N)
  i ← i+1
}

```

Algorithme 3

```

Go(N)
j ← 2
for (i ← 1 to j step 1)
{
  Go(E)
}
j ← j+1
for (i ← 1 to j step 1)
{
  Go(S)
}
j ← j+1
for (i ← 1 to j step 1)
{
  Go(W)
}
j ← j+1
for (i ← 1 to j step 1)
{
  Go(N)
}

```

Algorithme 4

```

Dir ← [N,E,S,W,N]
Stp ← [1,2,3,4,5]
for (j ← 0 to 4 step 1)
{
  for (i ← 1 to Stp[j] step 1)
  {
    Go(Dir[j])
  }
}

```

Algorithme 5

```

Dir ← [N,E,W,E,S,N,S,W,E,W,N]
Stp ← [1,2,1,1,2,1,2,4,2,1,5]
for (j ← 0 to 10 step 1)
{
  for (i ← 1 to Stp[j] step 1)
  {
    Go(Dir[j])
  }
}

```

Q1(a) [5 ptn]	Leidt Algoritme 1 Nono naar de cel met een X ? Indien je antwoord « Nee » is en Nono botst met een muur, geef dan ook de coördinaten van de eerste muur waarmee Nono botst.
Oplossing: Ja	
Q1(b) [5 ptn]	Leidt Algoritme 2 Nono naar de cel met een X ? Indien je antwoord « Nee » is en Nono botst met een muur, geef dan ook de coördinaten van de eerste muur waarmee Nono botst.
Oplossing: Nee, F1	
Q1(c) [5 ptn]	Leidt Algoritme 3 Nono naar de cel met een X ? Indien je antwoord « Nee » is en Nono botst met een muur, geef dan ook de coördinaten van de eerste muur waarmee Nono botst.
Oplossing: Ja	
Q1(d) [5 ptn]	Leidt Algoritme 4 Nono naar de cel met een X ? Indien je antwoord « Nee » is en Nono botst met een muur, geef dan ook de coördinaten van de eerste muur waarmee Nono botst.
Oplossing: Ja	
Q1(e) [5 ptn]	Leidt Algoritme 5 Nono naar de cel met een X ? Indien je antwoord « Nee » is en Nono botst met een muur, geef dan ook de coördinaten van de eerste muur waarmee Nono botst.
Oplossing: Nee, C3	

Vraag 2 – Piek posities

Gegeven een tabel $a[]$ met n getallen, een *piek positie van $a[]$* is een positie i in de tabel zodat $a[i]$ minstens even groot is als het element links ervan en het element rechts ervan (indien die bestaan). Let op dat tabellen geïndexeerd worden vanaf 0: het eerste element staat op positie 0 en het laatste op positie $n - 1$.

Bijvoorbeeld, als $a = [2, 7, 4, 5]$, dan zijn de piek posities van $a[]$ de posities 1 en 3 (overeenkomstig met de waardes 7 en 5). Een ander voorbeeld: als $a = [6, 6, 6]$, dan zijn alle posities (0, 1 en 2) piek posities. Het is eenvoudig om aan te tonen dat elke tabel minstens een piek positie heeft.

Q2(a) [1 pt]	Geef alle piek posities van de tabel $a = [1, 2, 3, 2, 1]$.
Oplossing: 2	
Q2(b) [1 pt]	Geef alle piek posities van de tabel $a = [5, 4, 3, 2, 1]$.
Oplossing: 0	
Q2(c) [2 ptn]	Geef alle piek posities van de tabel $a = [4, 2, 5, 3]$.
Oplossing: 0, 2	
Q2(d) [2 ptn]	Geef alle piek posities van de tabel $a = [0, 10, 10, 0, 0]$.
Oplossing: 1, 2 en 4	

Er zijn verschillende manieren om een piek positie te vinden, maar het is zeer gemakkelijk om een fout te maken! Je vriend Alex heeft je vorige nacht om 3 uur 's morgens de volgende programma's gestuurd. Hij beweert dat zijn programma's een piek positie kunnen vinden in eender welke tabel $a[]$. Je denkt dat Alex waarschijnlijk heel moe was toen hij de programma's schreef, en je besluit om ze na te kijken.

We veronderstellen dat, voor elk programma, de tabel met getallen, $a[]$, $n \geq 2$ elementen bevat.

Programma 1:

```

for (i ← 1 to n-2 step 1)
{
  if (a[i] >= a[i-1] and a[i] >= a[i+1])
  {
    return i
  }
}
if (a[0] > a[n-1])
{
  return 0
}
else
{
  return n-1
}

```

Q2(e) [1 pt]	Welke waarde geeft programma 1 als $a = [9, 8, 7, 6, 5]$?
Oplossing: 1	

Q2(f) [1 pt]	Welke waarde geeft programma 1 als $a = [0, 1, 3, 4]$?
Oplossing: 2	
Q2(g) [1 pt]	Welke waarde geeft programma 1 als $a = [0, 0, 0, 0]$?
Oplossing: 3	
Q2(h) [1 pt]	Welke waarde geeft programma 1 als $a = [1, 0, 0, 1]$?
Oplossing: 3	
Q2(i) [2 ptn]	Geeft programma 1 een piek positie terug voor om het even welke tabel $a[]$?
Oplossing: Nee	

Programma 2:

```
i ← 0
for (j ← 1 to n-1 step 1)
{
    if (a[j] > a[i])
    {
        i ← j
    }
}
return i
```

Q2(j) [1 pt]	Welke waarde geeft programma 2 als $a = [4, 3, 2, 1]$?
Oplossing: 0	
Q2(k) [1 pt]	Welke waarde geeft programma 2 als $a = [0, 1, 0, 5, 0, 5, 0]$?
Oplossing: 3	
Q2(l) [2 ptn]	Geeft programma 2 een piek positie terug voor om het even welke tabel $a[]$?
Oplossing: Ja	

Programma 3:

```
if (a[0] > a[n-1])
{
    return 0
}
else
{
    return n-1
}
```


Q2(m) [1 pt]	Welke waarde geeft programma 3 als $a = [1, 2, 3]$?
---------------------	---

Oplossing: 2

Q2(n) [1 pt]	Welke waarde geeft programma 3 als $a = [2, 2, 1]$?
---------------------	---

Oplossing: 0

Q2(o) [2 ptn]	Geeft programma 3 een piek positie terug voor om het even welke tabel $a[]$?
----------------------	--

Oplossing: Nee

Oplossingen

Programma 4:

```

for (i ← 0 to n-2 step 1)
{
  if (a[i] >= a[i+1])
  {
    return i
  }
}
return n-1

```

Q2(p) [1 pt]	Welke waarde geeft programma 4 als $a = [1, 2, 3, 2, 1]$?
---------------------	---

Oplossing: 2

Q2(q) [1 pt]	Welke waarde geeft programma 4 als $a = [0, 1, 0, 5, 0]$?
---------------------	---

Oplossing: 1

Q2(r) [1 pt]	Welke waarde geeft programma 4 als $a = [2, 2, 1, 0, 5]$?
---------------------	---

Oplossing: 0

Q2(s) [2 ptn]	Geeft programma 4 een piek positie terug voor om het even welke tabel $a[]$?
----------------------	--

Oplossing: Ja

Programma 5:

```

i = n / 2
while (i > 0 and a[i-1] > a[i])
{
  i ← i-1
}
while (i < n-1 and a[i+1] > a[i])
{
  i ← i+1
}
return i

```

Q2(t) [1 pt]	Welke waarde geeft programma 5 als $a = [1, 2, 3, 4, 5]$?
---------------------	---

Oplossing: 4

Q2(u) [1 pt]	Welke waarde geeft programma 5 als $a = [5, 4, 3, 2, 1]$?
---------------------	---

Oplossing: 0

Q2(v) [1 pt]	Welke waarde geeft programma 5 als $a = [4, 3, 2, 2, 6, 5]$?
---------------------	--

Oplossing: 4

Q2(w) [1 pt]	Welke waarde geeft programma 5 als $a = [0, 4, 2, 5, 3]$?
---------------------	---

Oplossing: 1

Q2(x) [2 ptn]	Geeft programma 5 een piek positie terug voor om het even welke tabel $a[]$?
----------------------	--

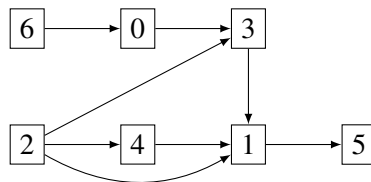
Oplossing: Ja

Oplossingen

Vraag 3 – De computerfabriek

Het bedrijf be-OI vader en zonen produceert computeronderdelen. Een computeronderdeel is een complex object en om het te produceren heeft men opnieuw andere computeronderdelen nodig, die op hun beurt zelf weer samengesteld zijn uit andere computeronderdelen, en zo verder. Het bedrijf beschikt over alle machines die nodig zijn om alles te produceren, maar het kan niet alle machines tegelijkertijd laten werken. Men moet dus *een geschikte volgorde vinden om de onderdelen te produceren, in de veronderstelling dat elke machine slechts één keer kan worden gebruikt*.

Elke machine heeft een nummer van 0 tot en met $n - 1$ en het bedrijf heeft de afhankelijkheden tussen de machines voorgesteld met behulp van een diagram zoals in het volgende voorbeeld.



Op dit diagram wordt elke machine voorgesteld door een rechthoek die het nummer van de machine bevat. Een pijl die gaat van machine i naar machine j duidt aan dat machine i moet gebruikt worden voor machine j gebruikt wordt, omdat de onderdelen geproduceerd door machine i nodig zijn voor de onderdelen geproduceerd door machine j (de geproduceerde hoeveelheden maken hier niet uit: het volstaat dat machine i gebruikt wordt voor machine j gebruikt wordt). Bijvoorbeeld, machine 2 en machine 0 moeten gebruikt worden voor machine 3 gebruikt wordt. Elke volgorde waarin machine 3 gebruikt wordt voordat machine 2 of machine 0 gebruikt wordt, is dus onmogelijk. Om ervoor te zorgen dat het diagram altijd zinnig is, veronderstellen we dat het diagram nooit een *kring* bevat: meer bepaald het is niet mogelijk om van een machine te vertrekken en dan een aantal pijlen te volgen zodat je terug bij dezelfde machine uit komt.

Zoals je ziet, kunnen sommige machines onmiddellijk gebruikt worden.

Q3(a) [1 pt]	Welke machines kunnen onmiddellijk gebruikt worden volgens het gegeven diagram? Geef ze allemaal.
Oplossing: 2 en 6	

Q3(b) [1 pt]	Welke machines kunnen gebruikt worden volgens het gegeven diagram als men enkel machine 2 al gebruikt heeft (ter herinnering: elke machine mag maar één keer gebruikt worden)? Geef ze allemaal.
Oplossing: 4 en 6	

Welk van de volgende volgordes zijn mogelijk volgens het gegeven diagram, rekening houdende met de gegeven beperkingen?

	Mogelijk	Onmogelijk	Volgorde waarin de machines worden gebruikt
Q3(c) [1 pt]	<input type="checkbox"/>	<input checked="" type="checkbox"/>	0,1,2,3,4,5,6
Q3(d) [1 pt]	<input checked="" type="checkbox"/>	<input type="checkbox"/>	6,0,2,3,4,1,5
Q3(e) [1 pt]	<input type="checkbox"/>	<input checked="" type="checkbox"/>	6,5,4,3,2,1,0
Q3(f) [1 pt]	<input checked="" type="checkbox"/>	<input type="checkbox"/>	2,4,6,0,3,1,5
Q3(g) [1 pt]	<input type="checkbox"/>	<input checked="" type="checkbox"/>	2,4,1,6,0,3,5

Er is duidelijk een specifieke voorwaarde waar de machines aan moeten voldoen (in een diagram zoals hetgeen gegeven is) zodat de machines onmiddellijk kunnen worden gebruikt.

Q3(h) [2 ptn]		Welke uitspraak uit de gegeven uitspraken beschrijft precies die machines die onmiddellijk kunnen worden gebruikt?
	<input type="checkbox"/>	De machines die een even aantal aankomende pijlen hebben.
	<input type="checkbox"/>	De machines die hoogstens 2 aankomende pijlen hebben.
	<input checked="" type="checkbox"/>	De machines die geen enkele aankomende pijl hebben.
	<input type="checkbox"/>	De machines die minstens 1 aankomende pijl hebben.
	<input type="checkbox"/>	De machines die geen enkele aankomende pijl hebben en een even (machine)nummer hebben.

We zoeken nu een algoritme om een volgorde te vinden waarin de machines kunnen gebruikt worden, zodat die volgorde compatibel is met het diagram. Het diagram wordt gegeven als een matrix $M[n][n]$ van booleaanse waardes, waarbij $M[i][j]$ de waarde **true** heeft als en alleen als er een pijl is die gaat van machine i naar machine j .

Dit is de matrix die hoort bij het diagram dat als voorbeeld gegeven is.
De cellen met de waarde **true** zijn gemarkeerd om de leesbaarheid te vergroten.

	0	1	2	3	4	5	6
0	false	false	false	true	false	false	false
1	false	false	false	false	false	true	false
2	false	true	false	true	true	false	false
3	false	true	false	false	false	false	false
4	false	true	false	false	false	false	false
5	false	false	false	false	false	false	false
6	true	false	false	false	false	false	false

$M[i][j]$

$M[2][4]$ is omcirkeld met een volle lijn, terwijl $M[4][2]$ omcirkeld is met een stippellijn.

$M[2][4] = \text{true}$ want er is een pijl die gaat van machine 2 naar machine 4.

$M[4][2] = \text{false}$ want er is geen pijl die gaat van machine 4 naar machine 2.

Het algoritme dat we voorstellen gebruikt een datastructuur genaamd *wachtrij*. Het gaat over een wachtrij met een begin en een einde. We kunnen elementen toevoegen aan het einde van de wachtrij en we verwijderen elementen te beginnen bij het begin van de wachtrij. We beschikken over functies: `Empty()`, hetgeen de wachtrij leeg maakt (de wachtrij bevat dan geen enkel element meer); `Insert(v)`, hetgeen element v toevoegt aan het **einde** van de wachtrij; `Remove()`, hetgeen het element in het **begin** van de wachtrij teruggeeft en dat element verwijdert uit de wachtrij; en ten slotte `IsEmpty()`, hetgeen **true** terug geeft als de wachtrij leeg is en **false** als de wachtrij minstens één element bevat.

Het algoritme is als volgt.

1. We beginnen met elke machine die onmiddellijk gebruikt kan worden toe te voegen aan de wachtrij.
2. Vervolgens herhalen we dezelfde acties totdat de wachtrij leeg is:
 - (a) we nemen een machine weg uit het begin van de wachtrij en we gebruiken deze machine;

- (b) als we door het gebruiken van deze machine ook andere machines kunnen gebruiken, voegen we deze machines toe aan de wachtrij.

Opmerkingen.

Een booleaanse waarde `bool` kan direct in de volgende instructie gebruikt worden: `if (bool) {...}`.

De operator `not` gedraagt zich als volgt op booleaanse waarden: `not true` is gelijk aan `false` en `not false` is gelijk aan `true`.

Dit is een implementatie van het algoritme, waarbij sommige stukken code ontbreken:

Input : Een matrix $M[n][n]$ met grootte $n \times n$, die de afhankelijkheden tussen de n machines voorstelt.

Output : Dit gebruikt de machines in een volgorde die compatibel is met M .

`Empty()`

`D` \leftarrow `[0,...,0]` // (een array van lengte n , geïnitieerd met 0'en)

for ($j \leftarrow 0$ to $n-1$ step 1)

{

for ($i \leftarrow 0$ to $n-1$ step 1)

 {

if ($M[i][j]$)

 {

$D[\dots] \leftarrow \dots$ // (A) en (B)

 }

 }

if (...) // (C)

 {

 Insert (j)

 }

}

while (**not** IsEmpty())

{

$m \leftarrow$ Remove()

 Gebruik machine m

for ($i \leftarrow 0$ to $n-1$ step 1)

 {

if ($M[m][i]$)

 {

$D[i] \leftarrow \dots$ // (D)

if (...) // (E)

 }

 Insert (i)

 }

 }

}

}

Q3(i) [2 ptn]

Wat is de uitdrukking (A) die ontbreekt in het algoritme?

Oplossing: j

Q3(j) [2 ptn] Wat is de uitdrukking (B) die ontbreekt in het algoritme?

Oplossing: $D[j] + 1$

Q3(k) [2 ptn] Wat is de voorwaarde (C) die ontbreekt in het algoritme?

Oplossing: $D[j] = 0$

Q3(l) [2 ptn] Wat is de uitdrukking (D) die ontbreekt in het algoritme?

Oplossing: $D[i] - 1$

Q3(m) [2 ptn] Wat is de voorwaarde (E) die ontbreekt in het algoritme?

Oplossing: $D[i] = 0$

Oplossingen

Vraag 4 – LIDAR

LIDAR is een technologie die licht of lasers gebruikt om afstanden of hoogtes van objecten te bepalen. Je maakt deel uit van een team dat wil uitmeten hoe hoog de verschillende delen van de Chinese Muur net zijn. Wanneer je hoog boven de Muur vliegt, merk je plots dat je LIDAR toestel een probleem heeft: in plaats van de hoogte van een enkel segment te meten, meet het de hoogte van verschillende opeenvolgende segmenten, en telt ze samen op. Je had enkel een vergunning om vandaag over de Muur te vliegen, en er is geen tijd om terug te gaan en de LIDAR te herconfigureren. Gezien jij de expert algoritmiëk van het team bent, vragen je collega's je om de echte hoogtes van de segmenten af te leiden van de data die de scan verzameld had.

Als vereenvoudiging zullen we de Muur voorstellen als een lijst $W[]$ van n gehele getallen die positief of gelijk aan 0 zijn ($W[i] \geq 0$). Elke $W[i]$ is de hoogte van een segment van de Muur.

De LIDAR maakt q scans, elke scan stelt de som van opeenvolgende elementen van $W[]$ voor.

Voorbeeld: neem bijvoorbeeld een muur met $n = 4$ segmenten: $W = [4, 3, 5, 8]$.

Zoals gewoonlijk start de index met 0, dus hier is $W[0] = 4$, $W[1] = 3$, $W[2] = 5$ en $W[3] = 8$.

De resultaten van 2 scans door de LIDAR zouden $W[1 \dots 3] = W[1] + W[2] + W[3] = 3 + 5 + 8 = 16$ en $W[0 \dots 1] = W[0] + W[1] = 4 + 3 = 7$ kunnen zijn (let op de notatie: $W[i \dots j] = W[i] + \dots + W[j]$).

Voor de volgende vier vragen bekijken we een muur $W[]$ met 4 segmenten.

De LIDAR heeft de metingen $W[0 \dots 2] = 32$, $W[0 \dots 3] = 42$ en $W[2 \dots 3] = 17$ gemaakt.

Q4(a) [1 pt]	Hoe hoog is $W[3]$?
Oplossing: 10, want $W[3] = W[0 \dots 3] - W[0 \dots 2] = 42 - 32 = 10$	
Q4(b) [1 pt]	Hoe hoog is $W[2]$?
Oplossing: 7, want $W[2] = W[0 \dots 2] + W[2 \dots 3] - W[0 \dots 3] = 32 + 17 - 42 = 7$	
Q4(c) [1 pt]	Wat is de waarde van $W[0 \dots 1]$?
Oplossing: 25, want $W[0 \dots 1] = W[0 \dots 3] - W[2 \dots 3] = 42 - 17 = 25$	
Q4(d) [1 pt]	Hoe veel verschillende muren zijn compatibel met de LIDAR metingen?
Oplossing: 26, want enkel $W[0]$ en $W[1]$ zijn onbekend maar hun som moet 25 zijn (denk erom dat een hoogte 0 kan zijn).	

Voor we het echte probleem oplossen, proberen we eerst de metingen effectief te simuleren. Daarmee kunnen we dan tests genereren zodat we kunnen nakijken dat we alles correct doen.

Laat ons eerst een eenvoudig, maar traag algoritme implementeren. De inputs zijn de muur $W[]$ en zijn lengte n , het aantal scans q , en de linkse en rechtse eindpunten van de scans in de arrays $left[]$ en $right[]$. De output is een array $Lidar[]$ waarvan de elementen de som $W[left[i] \dots right[i]]$ zijn, zoals gemeten door de scans.

```

Input  : n, W[], q, left[], right[]
Output : Lidar[]

Lidar[] is initialized to q zeroes.

for (i ← 0 to ... step 1) // (i)
{
    for (j ← left[i] to ... step 1) // (ii)

```



```

{
    Lidar[i] ← Lidar[i] + ... //(iii)
}
}

```

Er zijn een paar gaten in deze implementatie, kan je ze opvullen?

Q4(e) [1 pt]	Wat is de uitdrukking (i) in bovenstaand algoritme?
Oplossing: $q - 1$	

Q4(f) [1 pt]	Wat is de uitdrukking (ii) in bovenstaand algoritme?
Oplossing: $right[i]$	

Q4(g) [1 pt]	Wat is de uitdrukking (iii) in bovenstaand algoritme?
Oplossing: $W[j]$	

Hier is een sneller algoritme dat maar een enkele passage over de waarden van $W[]$ en van $left[]$ en $right[]$ maakt. Het idee is om eerst een prefix tabel $P[]$ te berekenen zodat $P[0] = 0$ en $P[i+1] = W[0 \dots i]$ en dat vervolgens te gebruiken om $Lidar[]$ te berekenen. De inputs en output zijn hetzelfde als in het eerste algoritme.

```

Input : n, W[], q, left[], right[]
Output : Lidar[]

P[] ← [0,0,---,0] //n+1 "0".

for (i ← 0 to ... step 1) // (i)
{
    P[i + 1] ← ... // (ii)
}

for (i ← 0 to ... step 1) // (iii)
{
    Lidar[i] ← ... // (iv)
}

```

Je moet enkel nog een paar gaten opvullen.

Q4(h) [1 pt]	Wat is de uitdrukking (i) in bovenstaand algoritme?
Oplossing: $n - 1$	

Q4(i) [2 ptn]	Wat is de uitdrukking (ii) in bovenstaand algoritme?
Oplossing: $P[i] + W[i]$	

Q4(j) [1 pt]	Wat is de uitdrukking (iii) in bovenstaand algoritme?
Oplossing: $q - 1$	

Q4(k) [3 ptn]	Wat is de uitdrukking (iv) in bovenstaand algoritme?
----------------------	---

Oplossing: $P[right[i] + 1] - P[left[i]]$

Als we de $P[]$ tabel van de scans van het vorige algoritme konden berekenen, zouden we eenvoudig alle $W[i]$ waarden kunnen berekenen. Laat ons proberen dat efficiënt te doen. De inputs zijn het aantal segmenten n in de muur, het aantal scans q , de eindpunten van de scans in de tabellen $left[]$ en $right[]$, en de waarden van de scans in $Lidar[i] = W[left[i] \dots right[i]]$. **Belangrijk: voor dit algoritme veronderstellen we dat alle scans consistent zijn (dat is, er zijn geen contradicties), en dat de scans een unieke $P[]$ bepalen.**

We gebruiken de array $K[][]$ om alle metingen die we per eindpunt van elke scan weten, op te slaan. Bijvoorbeeld, voor een scan $W[3 \dots 7] = 61$ weten we dat $P[8] = W[0 \dots 7] = W[0 \dots 2] + W[3 \dots 7] = P[3] + 61$ en dat $P[3] = P[8] - 61$. Om daar rekening mee te houden registreren we de vereiste $(8, 61)$ in de lijst $K[3]$ en de vereiste $(3, -61)$ in de lijst $K[8]$. Let goed op de indices!

Merk op dat $K[][]$ een array van lijsten is en dat de elementen in elke lijst $K[i][]$ paren van getallen (*index, waarde*) zijn.

De lijn **foreach** $((index, scan) \text{ in } K[current])$ in de code begint een lus die een keer uitgevoerd wordt voor elk element in de lijst $K[current][]$. De waarden van *index* en *scan* kunnen in die lus gebruikt worden.

De bewerking **append** x to y in de code voegt x toe als nieuw element rechts van de lijst y . Bijvoorbeeld, als $x = 4$ en $y = [6, 2]$, dan wordt y nadien $[6, 2, 4]$.

De inverse bewerking is **remove last element**: dit verandert $y = [6, 2, 4]$ in $[6, 2]$ en wijst 4 toe aan de gekozen variabele.

```

Input : n, q, left[], right[], Lidar[]
Output : P[]

Initialize K[][] to n+1 empty arrays.

for (i ← 0 to q - 1 step 1)
{
    append (... , Lidar[i]) to K[left[i]] // (i)
    append (...) to K[...] // (ii), (iii)
}

Initialize processed[] to n values false.
P[0] ← 0
processed[0] ← true
Initialize todo[] to [0]

while (todo[] is not empty)
{
    current ← remove last element of todo[]
    foreach ((index, scan) in K[current][ ])
    {
        P[index] ← ... // (iv)
        if (not processed[index])
        {
            processed[index] ← true
            append index to todo
        }
    }
}

```

```

    }
}

```

Q4(l) [1 pt] Wat is de uitdrukking (i) in bovenstaand algoritme?

Oplossing: $right[i] + 1$

Q4(m) [3 ptn] Wat is de uitdrukking (ii) in bovenstaand algoritme?

Oplossing: $left[i], -Lidar[i]$

Q4(n) [1 pt] Wat is de uitdrukking (iii) in bovenstaand algoritme?

Oplossing: $right[i] + 1$

Q4(o) [5 ptn] Wat is de uitdrukking (iv) in bovenstaand algoritme?

Oplossing: $P[current] + scan$

De volgende scans van **een muur met 9 segmenten** zijn consistent en bepalen een unieke $P[]$ (je kan het vorige algoritme gebruiken) en $W[]$.

- $W[0 \dots 0] = 4$
- $W[0 \dots 3] = 18$
- $W[0 \dots 5] = 20$
- $W[1 \dots 4] = 14$
- $W[2 \dots 8] = 23$
- $W[3 \dots 4] = 1$
- $W[4 \dots 7] = 15$
- $W[5 \dots 7] = 15$
- $W[5 \dots 8] = 18$
- $W[7 \dots 8] = 7$
- $W[8 \dots 8] = 3$

Q4(p) [4 ptn] Wat zijn de hoogtes van de 9 segmenten in de muur $W[]$?

Oplossing: $[4, 9, 4, 1, 0, 2, 9, 4, 3]$

De volgende scans van **een muur met 7 segmenten** zijn consistent, maar bepalen geen unieke $W[]$.

- $W[0 \dots 4] = 24$
- $W[1 \dots 5] = 26$

- $W[2 \dots 5] = 17$
- $W[3 \dots 4] = 2$
- $W[3 \dots 5] = 9$
- $W[4 \dots 6] = 11$
- $W[5 \dots 5] = 7$

Q4(q) [4 ptn] Hoe veel muren $W[]$ zijn consistent met de gegeven scans?

Oplossing: 3, elke muur $[5, 9, 8, 2 - x, x, 7, 4 - x]$ waar x 0, 1 of 2 kan zijn, is mogelijk.

Oplossingen

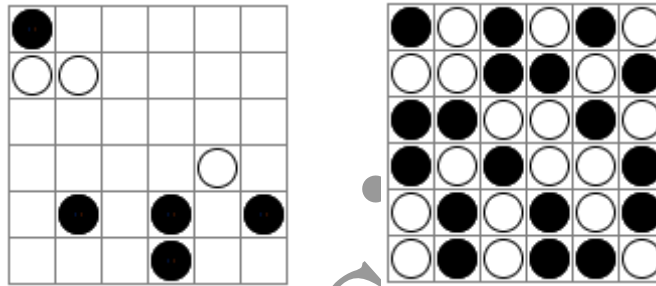
Vraag 5 – Binairo

Binairo (ook wel **Takuzu** genoemd) is een spel gebaseerd op logica met simpele regels, maar ingewikkelde oplossingen. Men speelt het op een vierkant veld van even lengte ($2n$ rijen van $2n$ cellen).

In het begin bevatten sommige cellen een pion (een zwarte pion of een witte pion). Alle andere cellen zijn leeg. De bedoeling is om in elke cel een pion te plaatsen, waarbij de volgende regels moeten worden gevolgd:

- **Regel 1:** Elke rij en kolom moet n witte pionnen en n zwarte pionnen bevatten.
- **Regel 2:** In elke rij en kolom mogen er twee opeenvolgende pionnen zijn van dezelfde kleur, maar geen drie.
- **Regel 3:** Elke rij en elke kolom is uniek (maar we zullen deze regel in het vervolg niet nodig hebben).

Dit is een voorbeeld van een 6×6 binairo veld (links) en de unieke oplossing ervan (rechts).



Dit zijn twee rijen die de regels niet volgen:



Regel 1 wordt niet gevolgd (er zijn meer witte pionnen dan zwarte pionnen).



Regel 2 wordt niet gevolgd (er zijn meer dan 2 opeenvolgende zwarte pionnen).

Vervolledig de volgende rijen **waarbij enkel de 2 eerste regels gevolgd moeten worden**.

Indien nodig, kan je hier je kladantwoorden noteren voordat je ze **op de antwoordbladen schrijft**.

Q5(a) [1 pt]	Vervolledig de rij	● ● □ □
		Oplossing: ● ● ○ ○

Q5(b) [1 pt]	Vervolledig de rij	○ □ □ ● ● □ □ ○
		Oplossing: ○ ● ○ ● ● ○ ● ○

Q5(c) [1 pt]	Vervolledig de rij	● □ ● □ □ □ □ ● ●
		Oplossing: ● ○ ● ○ ○ ● ○ ○ ● ●

Op hoe veel manieren kan men de volgende rijen vervolledigen **waarbij enkel de 2 eerste regels gevolgd moeten worden** ?




Q5(d) [1 pt]	Aantal manieren om dit te vervolledigen:	○ ● □ □
		Oplossing: 2

Q5(e) [1 pt]	Aantal manieren om dit te vervolledigen: <input type="radio"/> <input type="radio"/> <input type="checkbox"/> <input type="checkbox"/>
Oplossing: 1	
Q5(f) [1 pt]	Aantal manieren om dit te vervolledigen: <input type="checkbox"/> <input type="radio"/> <input type="checkbox"/> <input type="checkbox"/> <input checked="" type="radio"/>
Oplossing: 3	
Q5(g) [1 pt]	Aantal manieren om dit te vervolledigen: <input type="radio"/> <input type="checkbox"/> <input type="checkbox"/> <input checked="" type="radio"/> <input type="radio"/> <input type="checkbox"/> <input type="checkbox"/> <input checked="" type="radio"/>
Oplossing: 4	
Q5(h) [1 pt]	Aantal manieren om dit te vervolledigen: <input type="radio"/> <input type="radio"/> <input checked="" type="radio"/> <input type="radio"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/>
Oplossing: 2	
Q5(i) [1 pt]	Aantal manieren om dit te vervolledigen: <input type="radio"/> <input type="radio"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="radio"/> <input type="radio"/>
Oplossing: 2	
Q5(j) [1 pt]	Aantal manieren om dit te vervolledigen: <input type="radio"/> <input type="checkbox"/> <input type="radio"/> <input type="checkbox"/> <input type="checkbox"/> <input type="radio"/> <input type="checkbox"/>
Oplossing: 6	
Q5(k) [1 pt]	Aantal manieren om dit te vervolledigen: <input checked="" type="radio"/> <input type="radio"/> <input type="radio"/> <input checked="" type="radio"/> <input type="radio"/> <input type="radio"/> <input checked="" type="radio"/>
Oplossing: 0	
Q5(l) [1 pt]	Aantal manieren om dit te vervolledigen: <input type="checkbox"/> <input type="checkbox"/>
Oplossing: 2	
Q5(m) [1 pt]	Aantal manieren om dit te vervolledigen: <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/>
Oplossing: 6	
Q5(n) [1 pt]	Aantal manieren om dit te vervolledigen: <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/>
Oplossing: 14	

Programma 1 kijkt na of een rij van zwarte en witte pionnen de eerste 2 regels volgt. Sommige stukken code ontbreken en je moet deze aanvullen.

De rij die je moet nakijken wordt voorgesteld door een array *binairo*[] met gehele getallen. De witte pionnen worden voorgesteld door 1'en en zwarte pionnen worden voorgesteld door 2'en. Er wordt ook een getal *n* gegeven: de array *binairo*[] bevat $2n$ elementen (genummerd van 0 tot en met $2n - 1$) en, volgens **Regel 1**, moeten er even veel witte als zwarte pionnen zijn (n witte en n zwarte pionnen). Het programma moet **true** terug geven indien de eerste twee regels worden gevolgd en anders **false**.

Voorbeelden.

- Voor  geldt $n = 5$, *binairo* = [2, 1, 1, 2, 1, 2, 1, 2, 2, 1], en geeft het programma **true** terug.
- Voor  geldt $n = 3$, *binairo* = [1, 2, 2, 2, 1, 1], en geeft het programma **false** terug.
- Voor  geldt $n = 4$, *binairo* = [2, 1, 2, 2, 1, 2, 1, 2], en geeft het programma **false** terug.

Het programma loopt over *binairo*[] en gebruikt de variabele *t1* om het **totaal** aantal witte pionnen te tellen en de variabele *c1* om het aantal **opeenvolgende** witte pionnen (onder de laatst bekeken pionnen) te tellen. De variabelen *t2* en *c2* doen hetzelfde voor de zwarte pionnen.

Programma 1: De oplossingen volgen na de ...

```

Input  : n, binairo[]
Output : true or false
t1 ← 0; t2 ← 0; c1 ← 0; c2 ← 0
for (i ← 0 to 2*n-1 step 1){
  if (binairo[i] = 1) {
    t1 ← ...t1+1
    if (t1 > ...n) {return false}
    c1 ← ...c1+1
    if (c1 = ...3) {return false}
    c2 ← ...0
  }
  else {
    t2 ← ...t2+1
    if (t2 > ...n) {return false}
    c2 ← ...c2+1
    if (c2 = ...3) {return false}
    c1 ← ...0
  }
}
return ...true

```

Q5(o) [6 ptn]




Vervolledig de ... in programma 1.
Score tussen 0 en 6. Je verliest 1 punt per fout.

Oplossing: De oplossingen volgen na de ...

Programma 2 is een verbetering van programma 1. Dit programma kijkt na of een rij *waarbij eventueel bepaalde cellen nog leeg zijn* de 2 eerste regels volgt. Opgelet: het programma probeert niet na te kijken of het mogelijk is om de rij te vervolledigen zodat de regels gevolgd worden. Het kijkt enkel na of de eerste twee regels gevolgd worden *op dit moment, voor de pionnen die al geplaatst zijn op deze rij*.

De array `binairo[]` kan 0'en bevatten die lege cellen voorstellen (ter herinnering: 1'en en 2'en stellen pionnen voor). De variabelen `t1` en `t2` werden vervangen door een array `t[]` met 3 elementen. `t[0]` wordt niet gebruikt, `t[1]` telt het totaal aantal witte pionnen, `t[2]` telt het totaal aantal zwarte pionnen. De variabelen `c1` en `c2` werden vervangen door een array `c[]` met 3 elementen. `c[0]` wordt niet gebruikt, `c[1]` telt het aantal opeenvolgende witte pionnen, `c[2]` telt het aantal opeenvolgende zwarte pionnen.

Voorbeelden.

- Voor  geldt $n = 5$, `binairo = [2, 1, 0, 0, 1, 0, 0, 0, 0, 1]` en het programma geeft **true** terug.
- Voor  geldt $n = 3$, `binairo = [0, 2, 2, 2, 0, 1]`, en geeft het programma **false** terug, want er zijn meer dan 2 opeenvolgende zwarte pionnen.
- Voor  geldt $n = 4$, `binairo = [2, 0, 2, 2, 0, 2, 0, 2]`, en geeft het programma **false** terug want er zijn al te veel zwarte pionnen.

Programma 2: De oplossingen volgen na de ...

```

Input  : n, binairo[]
Output : true or false
t ← [0,0,0]
c ← [0,0,0]
for (i ← 0 to 2*n-1 step 1){
  j = binairo[i]
  if (...j ≠ 0) {
    t[j] ← ...t[j]+1
    if (...t[j] > n) {return false}
    c[j] ← ...c[j]+1
    if (...c[j] = 3) {return false}
    c[...3-j] ← ...0
  }
  else {
    c[1] ← 0
    c[2] ← 0
  }
}
return ...true

```

Q5(p) [6 ptn]

Vervolledig de ... in programma 2.
Score tussen 0 en 6. Je verliest 1 punt per fout.

Oplossing: De oplossingen volgen na de ...