be-OI 2013 **Halve Finale** 23 januari 2013	**Invullen in HOOFDLETTERS en LEESBAAR aub** VOORNAAM : ..... NAAM : ..... SCHOOL :	**Gereserveerd**

Belgische Informatica-olympiade (duur : 3u maximum)

Dit is de vragenlijst van de halve finale van de Belgische Informatica-olympiade 2013. Ze bevat negen vragen, en je krijgt **maximum 3u** de tijd om ze op te lossen. Naast elke vraag vind je een schatting van de benodigde tijd om ze op te lossen, en de maximale score die je kan behalen.

Algemene opmerkingen (lees dit aandachtig voor je begint)

1. Vul je voornaam, naam en school in, **alleen op het eerste blad**. Op alle andere bladzijden mag je enkel schrijven in de daarvoor **voorziene kaders**. Als je door een fout toch buiten de antwoordkaders moet schrijven, schrijf dan alleen verder op hetzelfde blad papier. Anders kunnen we je antwoord niet verbeteren.

2. Je mag alleen schrijfgerief bij je hebben. Rekentoestel, GSM, ... zijn **verboden**.

3. Schrijf je antwoorden met blauwe of zwarte **pen of balpen**. Laat geen antwoorden staan in potlood. Als je kladbladen wil, vraag ze dan aan een toezichthouder.

4. Voor alle opgaven werd **pseudo-code** gebruikt. Op de volgende bladzijde vind je een **beschrijving** van de pseudo-code die we hier gebruiken.

   Op open vragen mag je zelf antwoorden in **pseudo-code** of in één van de **toegestane programmeertalen** (Java, C, C++, Pascal, Python, PHP). We trekken geen punten af voor syntaxfouten. Tenzij het anders vermeld staat, mag je geen voorgedefinieerde functies gebruiken, met uitzondering van `max(a,b)`, `min(a,b)` en `pow(a,b)`, waarbij deze laatste a^b berekent.

5. Voor elke meerkeuzevraag (waarbij hokjes moeten worden aangekruist) doet een fout antwoord je evenveel punten *verliezen* als je met een correct antwoord zou winnen. Als je de vraag niet beantwoordt dan win je noch verlies je punten. Behaal je op die manier een negatieve score voor een volledige vraag, dan wordt de score teruggezet naar nul. Om op een meerkeuzevraag te antwoorden, kruis je het hokje aan (☒) dat met het juiste antwoord overeenkomt. Om een antwoord te annuleren, maak je het hokje volledig zwart (■).

6. Je mag **op geen enkel moment met iemand communiceren**, behalve met de toezichthouders. Je mag hen bijvoorbeeld wel om kladpapier vragen, maar je mag geen vragen stellen over de inhoud van de proef. Zo garanderen wij gelijke behandeling tussen deelnemers in alle regionale centra. Elke fout in de opgave moet je beschouwen als onderdeel van de proef.

7. Je mag in principe je **plaats niet verlaten** tijdens de proef. Moet je dringend naar het toilet, meldt dit dan aan een toezichthouder. Hij of zij kan dit toelaten of weigeren, afhankelijk van de mogelijkheid om je te laten vergezellen door een van de toezichters. Afhankelijk van het regionaal centrum kan het ook verboden zijn om te eten of te drinken in het lokaal.

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/juist (**`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 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 19.

```
geboortejaar ← 1993
leeftijd ← 2012 − 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*.

```
if (leeftijd ≥ 18)
{
  prijs ← 8
}
else
{
  prijs ← 6
}
```

Wanneer we in één variabele tegelijkertijd 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. (N noemen we de *lengte* van de array). 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 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]
}
```

En 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
}
```

Vraag 1 – Logisch denken (5 min – 4 ptn)

Bekijk het volgende programmafragment waarin `Schrijf(`x`)` **`true`** afdrukt als x *waar* is, en anders **`false`**.

```
Input : twee gehele waarden a en b

if (a = b and a ≠ 0)
{
   Schrijf(b = 0)
}
else
{
      if (b ≠ 0)
      {
        Schrijf(a = 0)
      }
      else
      {
        Schrijf(a ≠ b)
      }
}
```

Kruis bij elk van de volgende uitspraken aan of ze **correct** zijn of **niet correct** (incorrect).

	Correct	Incorrect	
Q1(a)	☐	☐	Als zowel a als b nul zijn ($= 0$), dan wordt er **`false`** afgedrukt.
Q1(b)	☐	☐	Als er **`true`** wordt afgedrukt, dan kunnen a en b niet gelijk zijn aan elkaar.
Q1(c)	☐	☐	Als noch a noch b nul zijn, dan wordt er **`true`** afgedrukt.
Q1(d)	☐	☐	Als a en b gelijk zijn, dan wordt er **`false`** afgedrukt, ongeacht de waarde van a en b.

Vraag 2 – Zoek een getal in een array (5 min – 4 ptn)

John heeft het volgende programmafragment geschreven om na te gaan of een getal n al dan niet voorkomt in een gegeven array arr van 20 elementen. (Ter herinnering: posities in arrays worden genummerd vanaf 0!)

```
Input : een array arr van 20 elementen en een geheel getal n
Output : drukt een bericht af dat aangeeft of arr het getal n bevat of niet

index ← 19
while (index > 0 and arr[index] ≠ n)
{
    index ← index − 1
}
if ( [...] )
{
    Schrijf("Het getal is gevonden")
}
else
{
    Schrijf("Het getal staat niet in de array")
}
```

John is echter niet helemaal zeker wat de correcte voorwaarde is die hij moet gebruiken in de **`if`**-instructie (op de zesde lijn). Hij ziet nog vier mogelijke opties (die we hieronder oplijsten).

Kruis bij elk van die opties aan of ze een geldige voorwaarde zijn voor de **`if`**-instructie (of niet). Met 'geldig' bedoelen we dat het programmafragment altijd het correcte resultaat moet geven, wat ook de inhoud van arr moge zijn.

	geldig	niet geldig	
Q2(a)	☐	☐	$index \leq 0$
Q2(b)	☐	☐	$arr[index] = n$
Q2(c)	☐	☐	$arr[index] = n$ **`and`** $index \leq 0$
Q2(d)	☐	☐	$arr[index] = n$ **`or`** $index \leq 0$

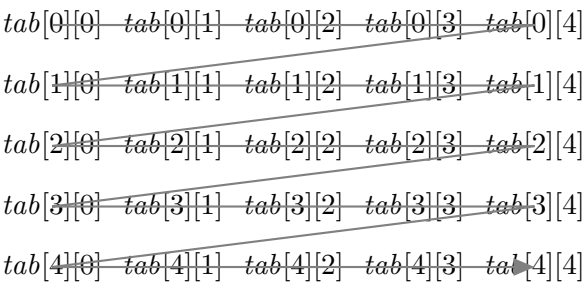
Vraag 3 – Tabellen doorlopen (20 min – 8 ptn)

Bij deze vraag zijn we geïnteresseerd in de verschillende manieren waarop we de elementen van een twee-dimensionale array (een matrix of tabel) kunnen doorlopen. Bij elk codefragment van deze opgave gebruiken we een matrix tab met N rijen en N kolommen. Het element op rij i en kolom j noteren we als $tab[i][j]$. De functie *DoeIets* is een functie waarvan je niet hoeft te weten wat ze precies doet.

Bekijk het volgende fragment:

```
for (i ← 0 to N − 1 step 1)
{
    for (j ← 0 to N − 1 step 1)
    {
        DoeIets(tab[i][j])
    }
}
```

Voor $N = 5$ zal deze code de elementen van de matrix doorlopen in de volgorde die hiernaast is geschetst:



Natuurlijk bestaan er nog andere manieren om alle elementen van een matrix te overlopen. Beschouw de volgende twee voorbeelden:

Q3(a)

```
for (i ← 0 to N − 1 step 1)
{
  if (i % 2 = 0)                    // met andere woorden, als i een even of paar getal is
  {
    for (j ← 0 to N − 1 step 1)
    {
      DoeIets(tab[i][j])
    }
  }
  else
  {
    for (j ← N − 1 to 0 step −1)
    {
      DoeIets(tab[i][j])
    }
  }
}
```

Q3(b)

```
richting ← 0
x ← 0
y ← −1
breedte ← N + 1
hoogte ← N
while (hoogte ≠ 0 and breedte ≠ 0)
{
  stapX ← richting % 2              // 0 als richting even is, anders 1
  stapY ← (richting + 1) % 2        // 1 als richting even is, anders 0
  if ((richting % 4) ≥ 2)
  {
    stapX ← −stapX
    stapY ← −stapY
  }
  aantal ← 0
  if (stapY = 0)
  {
    aantal ← hoogte
    hoogte ← hoogte − 1
  }
  else
  {
    aantal ← breedte
    breedte ← breedte − 1
  }
  for (i ← 1 to aantal − 1 step 1)
  {
    x ← x + stapX
    y ← y + stapY
    DoeIets(tab[x][y])
  }
  richting ← (richting + 1) % 4
}
```

Teken op de tabellen hieronder de **volgorde waarin de elementen worden doorlopen** bij de algoritmes hierboven. Vergeet niet om ook een **pijl te tekenen** die aangeeft in welke zin het traject wordt doorlopen (zoals in het voorbeeld).

Q3(a) met $N = 5$

$tab[0][0]$	$tab[0][1]$	$tab[0][2]$	$tab[0][3]$	$tab[0][4]$
$tab[1][0]$	$tab[1][1]$	$tab[1][2]$	$tab[1][3]$	$tab[1][4]$
$tab[2][0]$	$tab[2][1]$	$tab[2][2]$	$tab[2][3]$	$tab[2][4]$
$tab[3][0]$	$tab[3][1]$	$tab[3][2]$	$tab[3][3]$	$tab[3][4]$
$tab[4][0]$	$tab[4][1]$	$tab[4][2]$	$tab[4][3]$	$tab[4][4]$

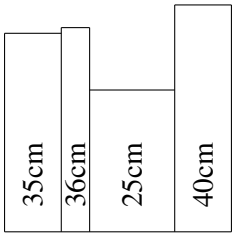
Q3(b) met $N = 6$

$tab[0][0]$	$tab[0][1]$	$tab[0][2]$	$tab[0][3]$	$tab[0][4]$	$tab[0][5]$
$tab[1][0]$	$tab[1][1]$	$tab[1][2]$	$tab[1][3]$	$tab[1][4]$	$tab[1][5]$
$tab[2][0]$	$tab[2][1]$	$tab[2][2]$	$tab[2][3]$	$tab[2][4]$	$tab[2][5]$
$tab[3][0]$	$tab[3][1]$	$tab[3][2]$	$tab[3][3]$	$tab[3][4]$	$tab[3][5]$
$tab[4][0]$	$tab[4][1]$	$tab[4][2]$	$tab[4][3]$	$tab[4][4]$	$tab[4][5]$
$tab[5][0]$	$tab[5][1]$	$tab[5][2]$	$tab[5][3]$	$tab[5][4]$	$tab[5][5]$

Vraag 4 – De bibliotheek (10 min – 8 ptn)

Je wenst de boeken in jouw kamer te rangschikken volgens grootte (hoogte), van klein naar groot. Een aantal boeken hebben ongeveer dezelfde grootte. Voor de zekerheid kies je ervoor om de boeken direct op de boekenplank te rangschikken door in elke stap slechts twee naast elkaar staande boeken om te wisselen.

Bijvoorbeeld, stel dat je de boeken hiernaast (FIGUUR 1) moet rangschikken, dan kan je het tweede en derde boek omwisselen, en daarna het eerste en tweede boek om al jouw boeken te ordenen (twee stappen).

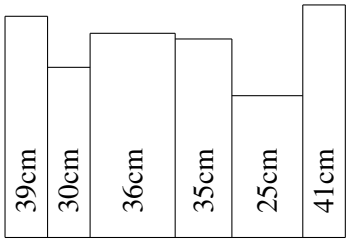


Figuur 1: Voorbeeld

Als je dezelfde regel blijft volgen, hoeveel stappen heb je dan minimaal nodig om de boekenplank hiernaast (FIGUUR 2) te ordenen?

Q4(a) [3 ptn] **een getal**

. . . . . .



Figuur 2: Q4(a)

Een vriend stelt je voor om eerst al jouw boeken te meten, dan het kleinste te selecteren en dit om te wisselen met het meest linkse boek. Daarna herhaal je dit proces: selecteer het tweede kleinste boek om het te verwisselen met het tweede meest linkse boek, enzovoort. Om te bewijzen dat dit werkt, is jouw vriend begonnen met het schrijven van het volgende algoritme:

```
Input  : arr, een array van lengte n met allemaal verschillende gehele getallen.
Output : De array arr is geordend in oplopende volgorde.

for (i ← 0 to n − 2 step 1)
{
  k ← i
  for (j ← i + 1 to n − 1 step 1)
  {
    if ( [...] )                    // (b)
    {
      k ← j
    }
  }
  if (k ≠ i)
  {
    swap(arr, i, k);              // Verwissel de boeken op posities i en k
  }
}
```

Vervolledig dit algoritme zodanig dat de array `arr` op een correcte manier geordend wordt.

Q4(b) [5 ptn] **een expressie**

. . . . . .

Vraag 5 – Recorder (10 min – 6 (+3) pt)

Om geen enkele van jouw favoriete uitzendingen te moeten missen tijdens jouw verblijf op de Internationale Olympiade in Australië, besluit je om een recorder te programmeren die toelaat om een aantal TV uitzendingen op te nemen op een harde schijf. De belangrijkste beperking van jouw systeem is dat het maar één enkele uitzending tegelijkertijd kan opnemen.

Een opname X wordt gedefinieerd door het beginuur (notatie: $X.begin$) en een einduur ($X.einde$). Een opname X conflicteert met een opname Y indien een gedeelte van X, of X in zijn geheel, overlapt met opname Y.

Schrijf een expressie die waar is enkel en alleen als twee opnames X en Y conflicteren. Voor deze expressie mag je enkel gebruik maken van de volgende stukjes expressie en van "en" (**and**), "of" (**or**) en haakjes.

A) $X.begin < Y.begin$	B) $X.begin > Y.begin$	C) $X.begin < Y.einde$	D) $X.begin > Y.einde$
E) $X.einde < Y.begin$	F) $X.einde > Y.begin$	G) $X.einde < Y.einde$	H) $X.einde > Y.einde$

Bijvoorbeeld, als je denkt dat het verifiëren dat X en Y conflicteren neerkomt op het verifiëren van de volgende expressie: $X.einde > Y.einde$ en ($X.begin < Y.einde$ of $X.einde < Y.begin$), antwoord dan "H **and** (C **or** E)".

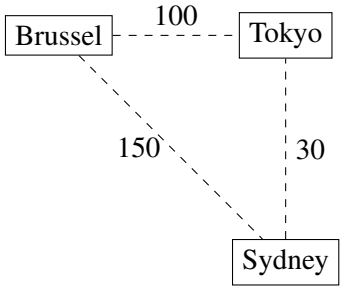
Bonus: je krijgt een bonus indien je de kortst mogelijke expressie vindt. (d.w.z. door gebruik te maken van het kleinst mogelijke aantal expressies)

Q5[6+3 ptn] **een expressie**

. . . . . .

Vraag 6 – Connecties (15 min – 14 ptn)

Een telecom-operator wenst een nieuw netwerk met een hoge snelheid op te zetten. De operator heeft een aantal steden gekozen die op het netwerk aangesloten moeten worden, en heeft de kostprijs bepaald van verschillende mogelijke (tweerichtings-) verbindingen tussen deze steden. Hij heeft dit alles als volgt in kaart gebracht:

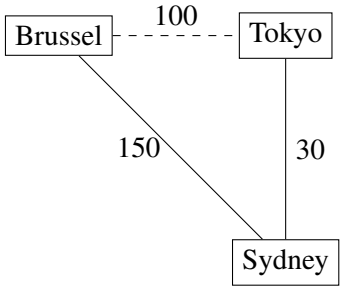


Zoals aangeduid op bovenstaande figuur zou de kost om Brussel met Sydney te verbinden 150 zijn, terwijl de verbinding tussen Tokyo en Sydney 30 kost.

De operator wil enkele van deze mogelijke verbindingen selecteren zodanig dat:

- elke stad verbonden is, ofwel direct ofwel indirect, met alle andere steden;
- de totale kost van alle geselecteerde verbindingen zo laag mogelijk is.

In het onderstaande voorbeeld stelt de operator voor om de verbindingen Brussel – Sydney en Sydney – Tokyo te selecteren en dit voor een totale kost van $150 + 30 = 180$. Merk op dat Brussel ook verbonden is met Tokyo maar op een indirecte manier, namelijk *via* Sydney.

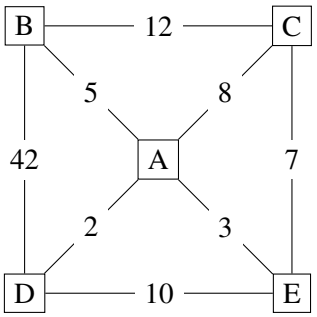


Kan de operator het beter doen? Indien ja, wat is de resulterende minimale kost?

Q6(a) [2 ptn] **« neen » ofwel een getal**

. . . . . .

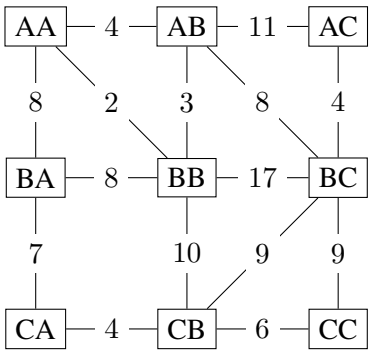
De operator overweegt nu een nieuwe kaart (zie hieronder). Wat is nu de verzameling van verbindingen die nodig zijn om een minimale kost te garanderen en eveneens te garanderen dat elke stad verbonden is, direct of indirect, met alle andere steden?



Q6(b) [4 ptn] **een lijst van verbindingen in het volgend formaat: « A–B, B–C,… »**

. . . . . .

Wat is de optimale kost die een operator kan bereiken op deze nieuwe kaart?



Q6(c) [5 ptn] **een getal**

. . . . . .

In het algemeen, indien er n steden verbonden moeten worden, wat zal het *aantal connecties* zijn dat gemaakt moet worden om zeker te zijn dat alle steden verbonden zijn, direct of indirect, aan de minimum kost?

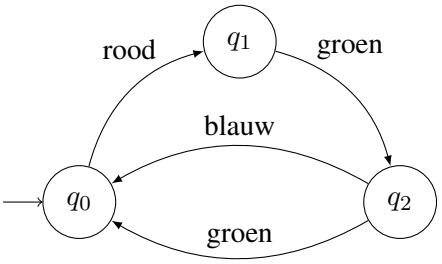
Q6(d) [3 ptn] **een expressie**

. . . . . .

Vraag 7 – De Australische afwas (15 min – 16 ptn)

In een restaurant in het Australische niemandsland wil Joe Bloggs, een keukenhulp die de ondankbare opdracht gekregen heeft om de afwas te doen, op gelijk welke manier wraak nemen. De borden van het restaurant zijn rood, groen of blauw en Joe heeft beslist dat hij enkel nog de afwas zal doen indien de borden geordend bij hem aankomen. De volgorde stelt hij elke morgen op. Elk bord dat niet in de juiste volgorde aankomt, wordt systematisch gebroken.

Om de volgorde op een precieze manier uit te leggen, gebruikt Joe diagrammen die bestaan uit cirkels, genaamd *toestanden*, en pijlen, genaamd *transities*. Elke transitie heeft als naam de kleur van een bord. Dit is een voorbeeld van zo een diagram:



Deze diagrammen worden op de volgende manier geïnterpreteerd. De toestanden laten toe om te weten welke kleuren Joe accepteert, dit zijn de kleuren op de *uitgaande* transities, dit zijn transities die een toestand *verlaten*. In het begin van zijn dag bevindt Joe zich in de *begintoestand* (q_0 in het voorbeeld), aangeduid door een korte horizontale pijl. Elke keer als Joe een bord binnenkrijgt, kijkt hij of er een transitie bestaat die de kleur van het bord draagt en die vertrekt vanuit zijn huidige toestand. Zo ja, dan wast hij het bord af en verandert hij van toestand: zijn nieuwe toestand wordt diegene aan het andere eind van de transitie. Zo nee, dan breekt hij het bord en wacht op het volgende bord (zijn toestand verandert niet).

Bijvoorbeeld, als we het bovenstaande diagram beschouwen, zal Joe de volgende reeks van borden afwassen: rood, groen, groen, rood, groen, blauw. Maar hij breekt het tweede en derde bord van de volgende sequentie: rood, blauw, rood, groen.

Als we het bovenstaande diagram beschouwen, hoeveel borden van de volgende reeks zal Joe dan breken: rood, blauw, groen, groen, blauw, blauw, rood?

Q7(a) [2 ptn] **een getal**

. . . . . .

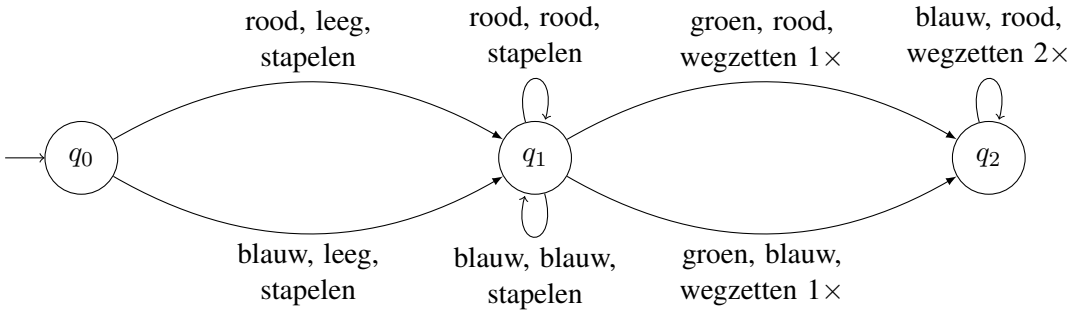
Gebruikmakende van het bovenstaande diagram, geef een reeks van 8 borden die Joe niet zal breken en die geen blauw bord bevat.

Q7(b) [4 ptn] **een reeks van acht kleuren (gebruik R voor rood en G voor groen)**

. . . . . .

Na één week hebben de obers van het restaurant begrepen hoe ze de diagrammen moeten lezen en geven ze hun borden in de juiste volgorde aan Joe met als gevolg dat hij geen enkel bord meer breekt. Maar Joe is sluw en besluit om de zaken wat complexer te maken: hij zal niet meer automatisch de gewassen borden wegzetten. Af en toe plaatst hij een gewassen bord op een stapel naast zijn gootsteen, en wil hij enkel nog bepaalde borden afwassen op voorwaarde dat het bovenste bord van de stapel ook de juiste kleur heeft!

De diagrammen zien er nu als volgt uit:



Elke transitie draagt nu *3 stukken informatie*: c_1, c_2 en a. Het *eerste stuk info* is een bepaalde kleur (rood, groen, blauw) en representeert de kleur van het bord dat Joe moet ontvangen om deze transitie uit te voeren (net zoals in het eenvoudige geval in het eerste stuk van deze vraag). Het *tweede stuk info* is ofwel een kleur (rood, groen, blauw) ofwel « leeg » en duidt aan wat Joe verwacht van de bordenstapel vooraleer hij de transitie uitvoert: indien het gaat om een kleur, dan wil Joe die kleur zien op de top van de stapel, indien het gaat om « leeg » dan moet de stapel leeg zijn. Het *derde stuk info* is een *actie* en duidt aan wat Joe met de borden zal doen. Er zijn drie mogelijke acties (voor een transitie die de info c_1, c_2, a draagt):

Actie a	**Effect**
stapelen	Joe wast het bord met kleur c_1 dat hij ontvangen heeft en plaatst het op de stapel.
wegzetten 1×	Joe wast het bord met kleur c_1 dat hij ontvangen heeft en zet het weg (dit verandert de stapel dus niet).
wegzetten 2×	Joe wast het bord met kleur c_1 dat hij ontvangen heeft, zet het weg, en hij neemt ook het bord met kleur c_2 van de stapel en zet dat ook weg (de top van de stapel is dus gewijzigd).

Zoals gewoonlijk, indien Joe zich in een bepaalde toestand bevindt en hij vindt geen uitgaande transitie (dus uitgaande uit zijn toestand) die aan al zijn voorwaarden voldoet, dan breekt hij het bord dat hij ontvangen heeft. In het voorbeeld hierboven betekent dit dat als men aan Joe de volgende reeks van borden geeft: blauw, groen, blauw, dat Joe het eerste bord zal afwassen en op de stapel zal plaatsen, hij zal het groene bord wassen en wegzetten maar hij zal het volgende, blauwe bord breken omdat hij een rood bord eist bovenaan de stapel.

Gebruikmakende van dit laatste diagram, hoeveel borden zal Joe breken in de volgende reeks: rood, blauw, groen, blauw, blauw?

Q7(c) [2 ptn] **een getal**

. . . . . .

Zelfde vraag voor de volgende reeks: rood, rood, rood, groen, blauw, blauw, blauw?

Q7(d) [2 ptn] **een getal**

. . . . . .

Zelfde vraag voor de reeks: rood, rood, groen, blauw, blauw, blauw?

Q7(e) [2 ptn] **een getal**

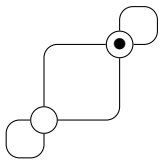
. . . . . .

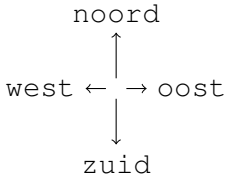
Veronderstel nu dat Joe eerst n rode borden ontvangt, daarna *één* groen bord en daarna m blauwe borden. Nog steeds gebruikmakend van het bovenstaande diagram, duid aan voor elk van de hieronder vermelde voorwaarden, welke ervoor zorgen dat Joe geen enkel bord zal breken (wat ook de waarden van n en m zijn die voldoen aan de voorwaarde):

	Garandeert dat Joe niks breekt	Garandeert *niet* dat Joe niks zal breken	Voorwaarde
Q7(f) [1pt]	☐	☐	$n < m$
Q7(g) [1pt]	☐	☐	$n > m$
Q7(h) [1pt]	☐	☐	$n = m$ en $n > 0$
Q7(i) [1pt]	☐	☐	$n + m$ is even

Vraag 8 – Onder de zuidpool (25 min – 9+4 ptn)

Eind 2012, zoals voorspeld door de Maya's, ontdekten Australische wetenschappers tekenen van buitenaards leven op aarde. Om precies te zijn, vonden ze verschillende kaarten van oeroude ondergrondse kamers en gangenstelsels onder de zuidpool. De eerste kaart werd ontdekt op kerstavond, in *Queen Mary Land*. De kaart lijkt op het onderstaande schema, en toont twee cirkelvormige vertrekken met elkaar verbonden door vier tunnels:





Bovendien hielp het toeval de wetenschappers een handje: ze slaagden er namelijk in om met behulp van sonar één van de ondergrondse kamers te lokaliseren en er zelfs een tunnel naartoe te boren. Nu zouden ze graag een kleine robot neerlaten in deze kamer.

Omdat de tunnels zich echter heel diep onder de grond bevinden, moet deze robot reeds zijn instructies krijgen vóór hij wordt neergelaten. Eenmaal in de kamer kan hij namelijk niet meer gecontacteerd worden, en enkel op het einde van zijn missie kan hij nog een korte video uitzenden naar de wetenschappers boven op de begane grond.

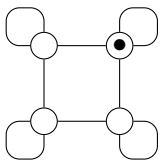
De robot begrijpt slechts vier eenvoudige opdrachten: `noord`, `oost`, `zuid` en `west`. Wanneer hij bijvoorbeeld de opdracht `noord` ontvangt, dan neemt hij de noordelijke uitgang van de kamer waarin hij op dat moment is, volgt daar de tunnel en blijft doorrijden totdat hij in de volgende kamer is aangekomen. En dan voert hij de volgende opdracht uit. (De andere drie opdrachten werken gelijkaardig, maar dan met een andere windrichting.)

De wetenschappers zijn vooral geïnteresseerd in de kamer die op de kaart is aangegeven met een zwarte stip. Jammer genoeg weten ze niet in welke van de kamers op de kaart de robot zal worden neergelaten. Welke opdrachten moeten ze geven zodat de robot met zekerheid eindigt in de juiste kamer?

Australische wetenschappers zijn niet dom, en heel snel realiseren ze zich dat er een eenvoudige oplossing bestaat voor hun probleem. Gebruiken ze (voor bovenstaande kaart) bijvoorbeeld de reeks opdrachten '`west noord`' dan zal de robot altijd in de juiste kamer eindigen, onafhankelijk van waar hij is gestart. Er zijn trouwens nog andere goeie oplossingen: de kortste zijn de (enkelvoudige) opdracht '`noord`', of de (enkelvoudige) opdracht '`oost`').

Na Kerstdag hebben ze nog verschillende andere buitenaardse kaarten gevonden en in drie gevallen zijn ze er ook in geslaagd om telkens één van de ondergrondse kamers terug te vinden. De nieuwe kaarten zijn echter wel een beetje ingewikkelder dan de eerste. Kan je hen helpen om de robot op de correcte manier te programmeren?

MacRobertson Land

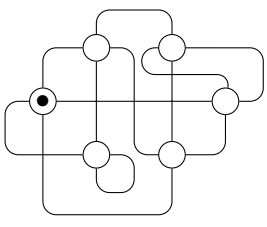


Schrijf de opdrachten neer die je aan de robot moet geven voor de kaart hiernaast. Na de laatste opdracht moet de robot zijn aangekomen in de kamer die met een zwarte stip is aangeduid, ongeacht in welke kamer hij is gestart.

Q8(a) [3 ptn] **een reeks opdrachten**

. . . . . .

Prince Charles Mountains

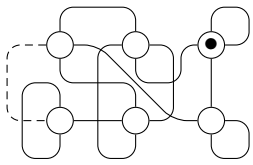


Zelfde opgave voor deze kaart. Je krijgt een bonus als jouw reeks opdrachten zo kort is als mogelijk. (Waar lijnen op de kaart elkaar snijden, zullen in de realiteit de tunnels boven of onder elkaar liggen.)

Q8(b) [3+2 ptn] **een reeks opdrachten**

. . . . . .

Wilhelm II Land



Zelfde opgave voor deze kaart. Je krijgt opnieuw een bonus als jouw oplossing de kortst mogelijke is.

Let op. We hebben geen vertrouwen in de meest linkse tunnel (aangegeven met een streepjeslijn). De robot mag daarom deze tunnel *in geen geval* gebruiken!

Q8(c) [3+2 ptn] **een reeks opdrachten**

. . . . . .

Vraag 9 – Antipersoonsmijnen (15 min – 12 ptn)

In het kader van een 'search and destroy'-missie tegen antipersoonsmijnen, roept Defensie jouw hulp in. De ontmijningsdienst van Defensie beschikt over gesofisticeerd detectiemateriaal voor mijnen, de MINE SWEEPER T-1000. Dit materiaal kan kaarten produceren met daarop de vermoedelijke plaatsen waar de mijnen zich bevinden. De T-1000 verdeelt de te analyseren ruimte in rechthoekige zones en duidt voor elke zone aan of deze een mijn bevat of niet. De mijnen zijn zeer gevoelig zijn voor trillingen, dus het is gevaarlijk om zich te verplaatsen in de zones nabij tenminste één mijn. Defensie vertrouwt je de taak toe om automatisch de gevaarlijke zones te bepalen, door te berekenen hoeveel mijnen in de nabijheid van elke zone liggen.

De T-1000 geeft je in elektronisch formaat een kaart met daarop de plaatsen waar de mijnen zich bevinden. Deze kaart is een twee-dimensionale array (een tabel) `mijnen` die M rijen en N kolommen bevat. Elke cel komt overeen met een zone. De cel op rij i en kolom j van de tabel bevat de waarde 1 indien een mijn gedetecteerd is binnen deze zone (ter herinnering, de rijen en kolommen zijn geïndexeerd vanaf 0). Indien er geen mijn gedetecteerd is, bevat de cel de waarde 0. Hieronder bevindt zich een voorbeeld van de tabel `mijnen`, met $M = 2$ rijen en $N = 3$ kolommen:

0	0	**1**
0	**1**	0

Het hieronder beschreven programma berekent het aantal mijnen in de nabijheid van elke zone. De output van het programma is een tabel `mijnen_cnt` met dezelfde grootte als de inputtabel `mijnen`. Elke cel `mijnen_cnt[`i`][`j`]` van deze nieuwe tabel moet het totaal aantal mijnen bevatten dat zich bevindt in `mijnen[`i`][`j`]` en in de acht aangrenzende cellen (of minder indien de cel zich situeert aan de grens van het terrein). Bijvoorbeeld, de tabel hieronder is de tabel `mijnen_cnt` die het programma moet berekenen op basis van de tabel `mijnen` die hierboven is gegeven.

1	2	2
1	2	2

Er wordt aan jou gevraagd om de ontbrekende delen van de code aan te vullen zodanig dat het programma de correcte inhoud van `mijnen_cnt` berekent.

```
Input  : mijnen, een tabel van M rijen en N kolommen waarvan de cellen enkel
         0 (geen mijn) of 1 (wel een mijn) bevatten, waarbij M en N ≥ 2
Output : mijnen_cnt, een tabel van M rijen en N kolommen
         waarvan alle cellen op 0 geïnitialiseerd zijn.

for (i ← 0 to M − 1 step 1)
{
  for (di ← −1 to 1 step 1)
  {
    for (dj ← 0 to 1 step 1)
    {
      if ( [...] )  // (a)
      {
        mijnen_cnt[i][0] ← mijnen_cnt[i][0] + mijnen[i+di][dj] ;
        mijnen_cnt[i][N − 1] ← mijnen_cnt[i][N − 1] + mijnen[...][...] ; // (b)
      }
    }
  }
}

for (j ← 1 to N − 2 step 1)
{
  for (di ← 0 to 1 step 1)
  {
    for (dj ← −1 to 1 step 1)
    {
      mijnen_cnt[0][j] ← mijnen_cnt[0][j] + mijnen[di][j+dj] ;
      mijnen_cnt[M − 1][j] ← mijnen_cnt[M − 1][j] + mijnen[...][...] ; // (c)
    }
  }
}

for (i ← 1 to M − 2 step 1)
{
  for (j ← 1 to N − 2 step 1)
  {
    for (di ← −1 to 1 step 1)
    {
      for (dj ← −1 to 1 step 1)
      {
        mijnen_cnt[i+di][j+dj] ← mijnen_cnt[i+di][j+dj] + mijnen[i+di][j+dj] ;
      }
    }
  }
}
```

Q9(a) [4 pts] **een expressie**

. . . . . .

Q9(b) [4 pts] **de indices van de cel**

. . . . . .

Q9(c) [4 pts] **de indices van de cel**

. . . . . .