be-OI 2020 **Finale - JUNIOR** zaterdag 7 maart 2020	**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)

1. 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.
2. Vul duidelijk je voornaam, naam en school in, **alleen op dit eerste blad**.
3. Jouw antwoorden moet je invullen op de daarop voorziene antwoordbladen, die je achteraan vindt.
4. Als je door een fout buiten de antwoordkaders moet schrijven, schrijf dan alleen verder op hetzelfde blad papier (desnoods op de achterkant).
5. Schrijf **duidelijk leesbaar** met blauwe of zwarte **pen of balpen**.
6. Je mag alleen schrijfgerief bij je hebben. Rekentoestel, GSM, . . . zijn **verboden**.
7. Je mag altijd extra kladpapier vragen aan de toezichthouder of leerkracht.
8. 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.
9. 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.
10. 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!

Oplossingen

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

©2020 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/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 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 ← 2003
leeftijd ← 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 ≥ 18)
{
  prijs ← 8    // Dit is een stukje commentaar
}
else
{
  prijs ← 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 ≥ 18)
{
    prijs ← 8 // Prijs voor een volwassene.
}
else if (leeftijd ≥ 6)
{
    prijs ← 6 // Prijs voor een kind van 6 of ouder.
}
else
{
    prijs ← 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.

Oplossingen

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
```

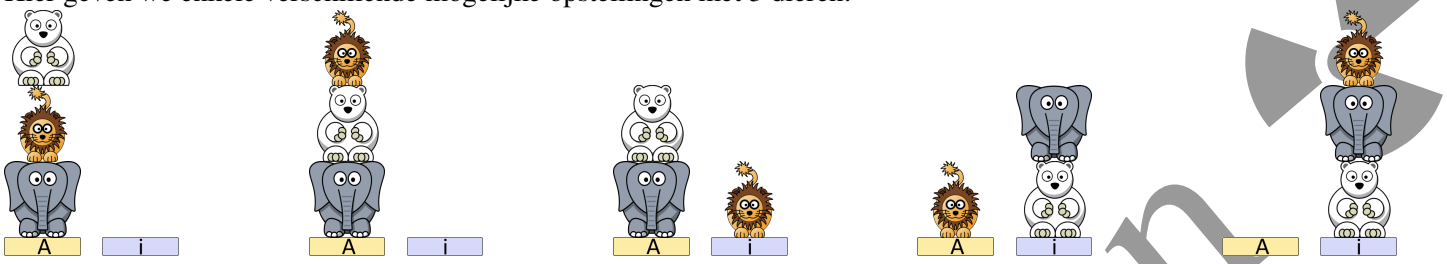
Opmerking: in dit laatste voorbeeld wordt de variabele i enkel gebruikt om de tel bij te houden van de **for**-lus. Er is dus geen uitleg voor nodig bij **Input** of **Output**, en de waarde ervan wordt niet teruggegeven.

Oplossingen

Vraag 1 – Circus

Professor Zarbi heeft een aantal circus acts opgesteld met zijn robot-diertjes. Hij gebruikt twee verhoogjes A en i (genoteerd met `A` en `i`) waarop de dieren kunnen staan, op elkaar gestapeld tot op willekeurig hoogte. De dieren kunnen zich op elke manier opstellen, het kan dat één verhoogje leeg is en dat alle dieren gestapeld zijn op het andere.

Hier geven we enkele verschillende mogelijke opstellingen met 3 dieren:



Q1(a) [2 ptn]	**Op hoeveel verschillende manieren kunnen we 2 dieren opstellen op de verhoogjes?**
Oplossing: 6	

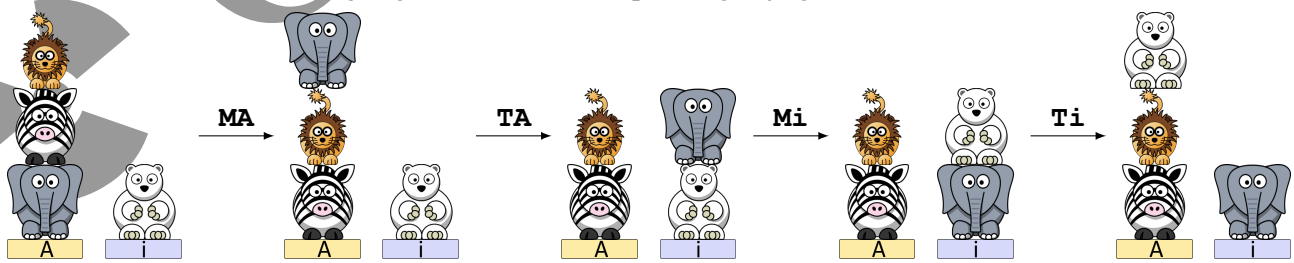
Q1(b) [4 ptn]	**Op hoeveel verschillende manieren kunnen we 3 dieren opstellen op de verhoogjes?**
Oplossing: 24	

Tijdens een opvoering geeft professor Zarbi instructies die de dieren doen veranderen van plaats.

Instructies om te “klimmen”.
`MA`: Dit horende, klimt het onderste dier van verhoogje `A` naar de top van de stapel op dat verhoogje.
`Mi`: Dit horende, klimt het onderste dier van verhoogje `i` naar de top van de stapel op dat verhoogje.
(Er gebeurt niets als er minder dan 2 dieren bevinden op het verhoogje van toepassing.)

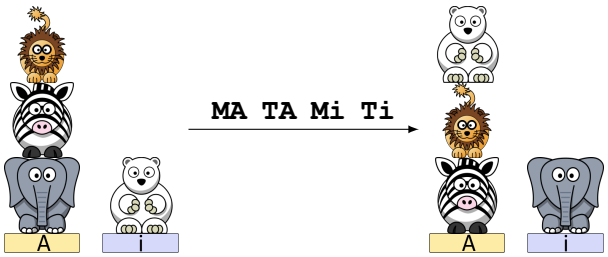
Instructies om te “springen”.
`TA`: Dit horende, springt het bovenste dier van verhoogje `A` en landt het bovenaan op de stapel van verhoogje `i`.
`Ti`: Dit horende, springt het bovenste dier van verhoogje `i` en landt het bovenaan op de stapel van verhoogje `A`.
(Er gebeurt niets als er zich geen dier bevindt op het verhoogje van toepassing.)

Bijvoorbeeld: 4 dieren staan opgesteld zoals in de linkerafbeelding en de professor geeft achtereenvolgens de instructies **`MA TA Mi Ti`**. De afbeeldingen geven weer hoe de opstelling wijzigt na elke instructie.



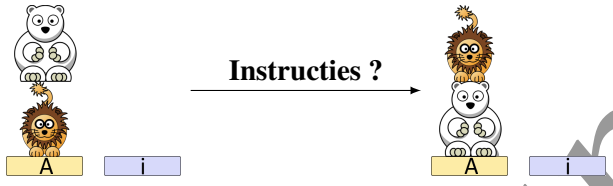
Oplossingen

De beweging die het geheel aan instructies omvat stellen we door één enkele pijl als volgt voor:



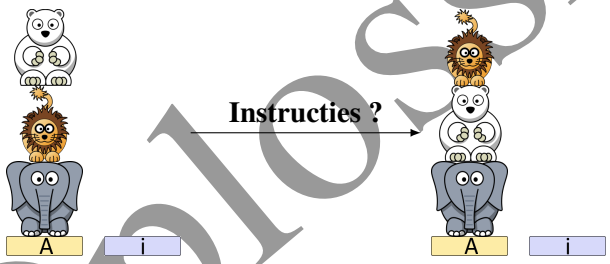
De lengte van een lijst instructies is het aantal instructies in die lijst. Bijvoorbeeld: de lijst instructies **`MA TA Mi Ti`** heeft lengte 4.
In de volgende vragen moet je een lijst van instructies uit {**`MA`**, **`Mi`**, **`TA`**, **`Ti`**} vinden die de opstelling van links doet veranderen naar die van rechts. **Jouw lijst moet zo kort mogelijk zijn.** Als jouw lijst niet de kortst mogelijke lengte heeft, krijg je slechts de helft van de punten van die vraag.

Q1(c) [2 ptn]



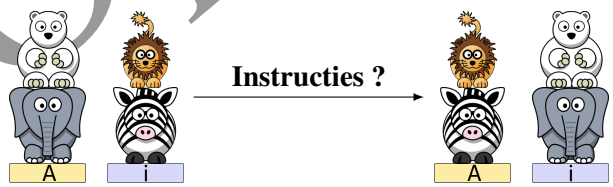
Oplossing: **`MA`**

Q1(d) [2 ptn]



Oplossing: **`TA MA Ti MA`**

Q1(e) [2 ptn]



Oplossing: bijvoorbeeld **`MA TA TA Mi Ti Mi Ti`**

Oplossingen

Q1(f) [4 ptn]

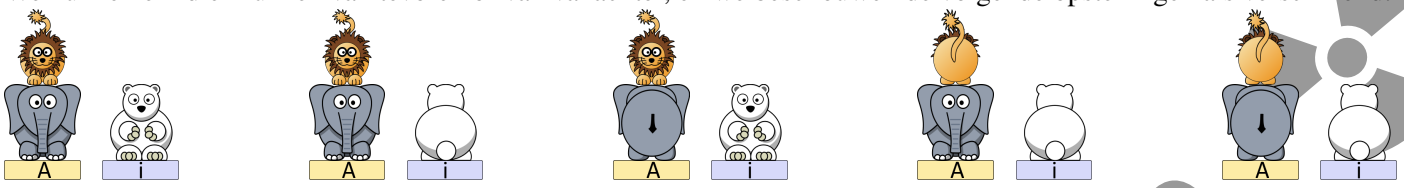
Twee dieren moeten zo snel mogelijk alle opstellingen aannemen. Beginnende bij de opstelling afgebeeld links, moet men één enkele keer alle andere opstellingen aangenomen hebben alvorens terug te komen naar de begin-opstelling.

Oplossing: **`MA TA TA Mi Ti Ti`** ou **`TA TA Mi Ti Ti MA`**

Oplossingen

Laten we nieuwe instructies toevoegen om te "draaien".
`RA`: Dit horende, draait het bovenste dier op verhoogje `A` 180° om zichzelf heen (het draait zich om).
`Ri`: Dit horende, draait het bovenste dier op verhoogje `i` 180° om zichzelf heen (het draait zich om).
(Er gebeurt niets als erg zich geen dier bevindt op het verhoogje van toepassing.)

We kunnen elk dier nu zien van tevoren of van vanachter, en we beschouwen de volgende opstellingen als verschillend:



Q1(g) [2 ptn]	**Ermee rekening houdend dat we elk diertje kunnen bekijken langs voor alsook langs achter, hoeveel verschillende opstellingen op de verhoogjes zijn er dan voor twee dieren ?**
Oplossing: $6 \cdot 2^2 = 24$	

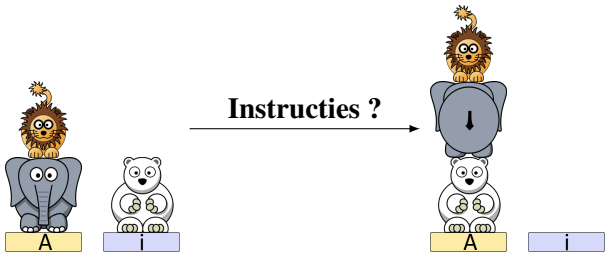
Q1(h) [4 ptn]	**Ermee rekening houdend dat we elk diertje kunnen bekijken langs voor alsook langs achter, hoeveel verschillende opstellingen op de verhoogjes zijn er dan voor drie dieren ?**
Oplossing: $24 \cdot 2^3 = 192$	

Wanneer een dier "klimt" of "springt", dan verandert zijn oriëntatie niet: of we de "voor- of achterkant" van het dier zien.
In de volgende vragen moet je een lijst instructies uit {**`MA`**, **`Mi`**, **`TA`**, **`Ti`**, **`RA`**, **`Ri`**} kiezen die de opstelling van links doet veranderen naar die van rechts.
Jouw lijst moet zo kort mogelijk zijn. Als jouw lijst niet de kortst mogelijke lengte heeft, krijg je slechts de helft van de punten van die vraag.

Q1(i) [2 ptn]

Oplossing: **`TA RA Ti`** of **`MA RA MA`**

Q1(j) [2 ptn]



Oplossing: **`Ti MA RA MA`**

Oplossingen

Q1(k) [2 ptn]

Oplossing: Bijvoorbeeld: **`Ri Mi Ri Ti MA TA MA Mi`**

Oplossingen

Vraag 2 – De poort openen

Een poort is beveiligd met een geheime code van 4 cijfers.
De poort opent onmiddellijk van zodra dat de laatste 4 ingedrukte cijfers overeen komen met de geheime code.
Als we opeenvolgend de cijfers 6,4,5,5,0,8,6,7 ingeven, dan zal de poort zich openen als de geheime code 6455, 4550, 5508, 5086 of 0867 is.
We kunnen nu een robot programmeren gebruik makende van het commando `press(a)`, dat de robot laat drukken op het knopje [a] (`a` is natuurlijk een cijfer van 0 tot en met 9).

Programma 1 :

```
for (a ← 0 to 9 step 1) {
  for (b ← 0 to 9 step 1) {
    for (c ← 0 to 9 step 1) {
      for (d ← 0 to 9 step 1) {
        press(a)
        press(b)
        press(c)
        press(d)
      }
    }
  }
}
```

In de volgende vragen, dien je enkel rekening te houden met de ingedrukte toetsen vanaf de uitvoering van Programma 1.

Q2(a) [1 pt]	**Hoeveel keer drukt de robot op een knopje doorheen de uitvoering van Programma 1 ?**
Oplossing: $4 \cdot 10^4 = 40000$	

Q2(b) [1 pt]	**Hoeveel keer zal het knopje [5] ingedrukt worden ?**
Oplossing: 4000	

Q2(c) [2 ptn]	**Hoeveel combinaties van 4 cijfers worden er getest door Programma 1 ? Als een combinatie meerdere keren getest wordt, dan moet je die pogingen allemaal meetellen.**
Oplossing: $40000 - 3 = 39997$	

Q2(d) [1 pt]	**Hoeveel verschillende combinaties van 4 cijfers worden er getest door Programma 1 ? Als een combinatie meerdere keren getest wordt, dan tel je die maar 1 keer.**
Oplossing: 10000	

Q2(e) [1 pt]	**Zal de poort zich steeds openen doorheen de uitvoering van Programma 1 ongeacht wat de geheime code is ?**
Oplossing: JA	

Laten we de ingedrukte cijfers door de robot nummeren vanaf **0** (vergis je niet, we beginnen te tellen vanaf **nul**). De eerste twintig cijfers die worden ingedrukt (genummerd van 0 19) zijn dus 0,0,0,0, 0,0,0,1, 0,0,0,2, 0,0,0,3, 0,0,0,4. Merk

Oplossingen

op dat de cijfers waarvan de positie een veelvoud van 4 zijn altijd overeenkomen met `press(a)` in het programma, en nooit `press(b)`, noch `press(c)`, noch `press(d)`.

Q2(f) [1 pt]	**Welk cijfer wordt er ingedrukt door de robot bij nummer 680 ?**
Oplossing: 0	

Q2(g) [1 pt]	**Welk cijfer wordt er ingedrukt door de robot bij nummer 682 ?**
Oplossing: 7	

Q2(h) [1 pt]	**Geef de 8 opeenvolgende cijfers die ingedrukt worden door robot beginnende bij nummer 732.**
Oplossing: 0183 0184	

Een "**mooie opeenvolging**" is een opeenvolging van 8 cijfers die ingedrukt worden door de robot gedurende de uitvoering van Programma 1 en **begint steeds met een toets genummerd met een veelvoud van 4**, of anders gezegd, beginnend met een cijfer dat ingedrukt wordt als gevolg van `press(a)` in de tekst van het programma.
Hier zijn enkele voorbeelden van "mooie opeenvolgingen": 0000 0001, 2307 2308, 6499 6500 (de spatie is hier louter voor de leesbaarheid).
Merk op dat een mooie opeenvolging altijd bestaat uit twee opeenvolgende getallen van 4 cijfers.
We zeggen dat een mooie opeenvolging **een combinatie test** als deze combinatie voorkomt in de mooie opeenvolging.
Bijvoorbeeld: de mooie opeenvolging **2307 2308** test de combinaties 2307, 3072, 0723, 7230 en 2308.
Een ander voorbeeld: mooie opeenvolging **7171 7172** test de combinaties 7171 en 1717, nog eens 7171 en 1717, en uiteindelijk 7172.
Merk op dat het twee keer testen van 7171 niet plaats neemt op hetzelfde ogenblik; het gaat dus over eenzelfde combinatie die twee keer getest wordt (hetzelfde voor 1717).

Q2(i) [5 ptn]	**Vind 5 mooie opeenvolgingen die de volgende combinatie test: 3108. Tip: je kan een combinatie "opdelen" in verschillende mogelijkheden (|3108, 3|108, 31|08, . . .).**
Oplossing: 08**31 08**32, 108**3 108**4, 3107 **3108**, 3108 3109 et 8**310 8**311	

Q2(j) [1 pt]	**Hoeveel keer drukt de robot opeenvolgend op de knoppen 3,1,0,8 ? Anders gezegd, hoeveel keer wordt de combinatie 3108 getest ?**
Oplossing: 4 (de mooie opeenvolgingen 3107 3108 en 3108 3109 doen dezelfde test)	

Q2(k) [3 ptn]	**Vind 3 mooie opeenvolgingen die de combinatie 6464 testen.**
Oplossing: 4**646 464**7, 6463 6464 en **6464 64**65	

Q2(l) [1 pt]	**Hoeveel keer drukt de robot opeenvolgend op de knopjes 6,4,6,4 ? Anders gezegd, hoeveel keer wordt de combinatie 6464 getest ?**
Oplossing: 4 (let op dat je elke test maar één keer meetelt.)	

Oplossingen

Q2(m) [4 ptn]	**Vind de vier mooie opeenvolgingen die de combinatie 9990 testen**
Oplossing: 89**99 90**00, 989**9 990**0, 9989 **9990** en 9990 9991	

Q2(n) [1 pt]	**Hoeveel keer drukt de robot opeenvolgend op de knopjes 9,9,9,0 ? Anders gezegd, hoeveel keer wordt de combinatie 9990 getest ?**
Oplossing: 3	

Q2(o) [1 pt]	**Hoeveel keer drukt de robot opeenvolgend op de knopjes 8,4,5 ? Help: Als X een cijfer is, dan zijn bijvoorbeeld 845X, 45X8 en 5X84 verschillende geheime codes.**
Oplossing: $4 \cdot 10 = 40$	

Oplossingen

We beschouwen nu een niet-deterministisch programma, hetgeen betekent dat voor sommige toetsen, willekeurige waarden zullen gegenereerd worden.
In het programma hebben we de functie `random()` die een random cijfer genereert van 0 t.e.m. 9.
Merk op dat de tweede loop is aangepast: het gaat niet om een loop van 0 t.e.m. 9 maar van 0 t.e.m. **a**.

Programma 2 :

```
for (a ← 0 to 9 step 1) {
  for (d ← 0 to a step 1) {
        press(a)
        press(random())
        press(random())
        press(d)
  }
}
```

In de vragen die volgen, moet je enkel rekening houden met knopjes die ingedrukt worden vanaf de uitvoering van Programma 2.

Q2(p) [3 ptn]	**Hoeveel combinaties van 4 cijfers worden er getest door Programma 2 ? Het kan zijn dat sommige combinaties meerdere keren getest worden, in dat geval tel je ze allemaal.**
Oplossing: $(1+2+3+4+5+6+7+8+9+10)\cdot 4-3=217$	

Q2(q) [2 ptn]	**Stel dat de geheime code 7431 is. Is het mogelijk dat de poort zich zal openen doorheen de uitvoering van Programma 2 ?**
Oplossing: JA	

Q2(r) [2 ptn]	**Stel dat de geheime code 4444 is. Is het mogelijk dat de poort zich zal openen doorheen de uitvoering van Programma 2?**
Oplossing: JA	

Q2(s) [2 ptn]	**Stel dat de geheime code 2345 is. Is het mogelijk dat de poort zich zal openen doorheen de uitvoering van Programma 2 ?**
Oplossing: JA	

*Uitleg: als we bijvoorbeeld de ingedrukte knopjes neerschrijven beginnende vanaf het moment **a** waarde 3 heeft in de eerste loop en **d** waarde 2 heeft in de tweede loop, dan genereert het programma 3**__2 3__**3 waarbij * een random cijfer voorstelt en de laatste 2 kunnen bijvoorbeeld 4 en 5 zijn.*

Het is mogelijk om een optimaal programma te schrijven dat een rij van cijfers (toetsen) genereert zodat elke 4-cijferige combinatie maar exact één keer getest wordt. In de vragen die volgen, moet je enkel rekening houden met de drukken gegenereerd door de uitvoering van zo'n optimaal programma.

Q2(t) [4 ptn]	**Hoeveel keer worden knopjes ingedrukt doorheen de de uitvoering van zo'n optimaal programma ?**
Oplossing: $10^4+3=10003$.	

Uitleg: Er zijn 10000 combinaties om uit te testen, elke toets moet beginnen met een unieke toets, en men mag de laatste 3

Oplossingen

cijfers van de laatste combinatie niet vergeten. Er zijn meerdere mogelijkheden om zo'n optimaal programma te schrijven, maar hoe dan ook zijn de laatste 3 gegenereerde cijfers altijd dezelfde als de eerste drie.
Bijvoorbeeld, als zo'n optimale rij begint met 123, eindigt die ook met 123.

Q2(u) [2 ptn]	**Als een programma Optimaal123 een rij genereert die begint met 123, hoeveel keer zal de robot dan opeenvolgend de knopjes 8,4,5 indrukken ?**
Oplossing: 10	

Q2(v) [2 ptn]	**Als we het optimale programma Optimaal123 uitvoeren dat een rij genereert beginnend met 123, hoeveel keer zullen dan de knopjes 1,2,3 opeenvolgend worden ingedrukt ?**
Oplossing: 11	

Oplossingen

Vraag 3 – Verstoppertje

Alice en Bob spelen verstoppertje. Ze spelen in een huis met $n \geq 2$ kamers, genummerd van 0 tot $n-1$. Alice gaat zich verstoppen, en Bob gaat haar proberen te vinden. Aangezien Bob heel voorspelbaar is, weet Alice reeds van tevoren in welke kamers en in welke volgorde Bob gaat zoeken. Bob gaat $m \geq 1$ bezoekjes brengen, waarvan sommige eventueel aan dezelfde kamer. Als Bob een kamer bezoekt en Alice zich daar op dat moment bevindt, dan wint Bob. Alice wilt natuurlijk voorkomen dat Bob wint, en om dat te doen, kan ze kiezen waar ze zich in het begin verstopt, en kan ze zich ook verplaatsen van de ene kamer naar de andere tussen twee bezoekjes van Bob. Maar elke verplaatsing is ook riskant (ze kan gezien worden), en dus probeert ze zo haar totaal aantal verplaatsingen tot een minimum te beperken.

We kunnen er bijvoorbeeld van uitgaan dat het huis $n = 3$ kamers bevat, genummerd 0, 1 en 2, en dat Bob de volgende rij (met lengte $m = 4$) van bezoekjes gaat uitvoeren:

$$\texttt{bob[]} = \{1, 2, 0, 2\}.$$

Een mogelijke oplossing voor Alice is om haar eerst in kamer 2 te verstoppen, dan zich te verplaatsen naar kamer 0 voordat Bob kamer 2 bezoekt, en zich uiteindelijk naar kamer 1 te verplaatsten alvorens Bob kamer 0 bezoekt. Alice verplaatst zich dan 2 keer en haar positie doorheen de tijd wordt beschreven door de volgende rij:

$$\texttt{alice[]} = \{2, 0, 1, 1\}.$$

Maar Alice kan beter doen: na kamer 2 verlaten te hebben, kan ze zich direct naar kamer 1 verplaatsen en daar blijven tot het einde van het spel. Ze zou zich zo maar één enkele keer moeten verplaatsten en haar positie doorheen de tijd wordt dan beschreven door de volgende rij:

$$\texttt{alice[]} = \{2, 1, 1, 1\}.$$

Ze kan echter niet beter doen; ze moet zich minstens één keer verplaatsen. Alice kan doorheen het volledige spel niet in éénzelfde kamer blijven, want Bob bezoekt alle kamers minstens één keer. Dus, voor $n = 3$ en $\texttt{bob[]} = \{1, 2, 0, 2\}$, is het minimum aantal verplaatsingen voor Alice 1.

Q3(a) [2 ptn]	**Wat is het minimum aantal verplaatsingen voor Alice als $n = 2$ en Bob de kamers bezoekt als volgt: `bob[]` $= \{0, 1, 0, 1\}$?**
Oplossing: 3 ($\texttt{alice[]} = \{1, 0, 1, 0\}$)	

Q3(b) [2 ptn]	**Wat is het minimum aantal verplaatsingen voor Alice als $n = 3$ en Bob de kamers bezoekt als volgt: `bob[]` $= \{0, 1, 0, 1\}$?**
Oplossing: 0 ($\texttt{alice[]} = \{2, 2, 2, 2\}$)	

Q3(c) [2 ptn]	**Wat is het minimum aantal verplaatsingen voor Alice als $n = 3$ en Bob de kamers bezoekt als volgt: `bob[]` $= \{2, 1, 2, 0\}$?**
Oplossing: 1 (bijvoorbeeld, $\texttt{alice[]} = \{0, 0, 1, 1\}$)	

Q3(d) [2 ptn]	**Wat is het minimum aantal verplaatsingen voor Alice als $n = 3$ en Bob de kamers bezoekt als volgt: `bob[]` $= \{0, 1, 2, 0, 2, 0\}$?**
Oplossing: 1 ($\texttt{alice[]} = \{2, 2, 1, 1, 1, 1\}$)	

Q3(e) [2 ptn]	**Wat is het minimum aantal verplaatsingen voor Alice als $n = 4$ en Bob de kamers bezoekt als volgt: `bob[]` $= \{0, 1, 2, 3, 0\}$?**
Oplossing: 1 (bijvoorbeeld, $\texttt{alice[]} = \{3, 3, 3, 2, 2\}$)	

Oplossingen

Q3(f) [2 ptn]	**Wat is het minimum aantal verplaatsingen voor Alice als $n = 4$ en Bob de kamers bezoekt als volgt: `bob[]` $= \{0, 1, 2, 3, 0, 1\}$?**
Oplossing: 1 (`alice[]` $= \{3, 3, 3, 2, 2, 2\}$)	

Q3(g) [3 ptn]	**Wat is het minimum aantal verplaatsingen voor Alice als $n = 4$ en Bob de kamers bezoekt als volgt: `bob[]` $= \{0, 1, 2, 3, 0, 1, 2\}$?**
Oplossing: 2 (bijvoorbeeld, `alice[]` $= \{3, 3, 3, 2, 2, 3, 3\}$)	

Q3(h) [4 ptn]	**In een huis met n kamers, wat is het minimum aantal bezoekjes m dat Bob moet uitvoeren om Alice minstens k keer te laten verplaatsen?**
Oplossing: $k(n-1)+1$	

Alles wordt fijner met een beetje code

Alice beslist om een programma te schrijven dat haar laat weten in welke kamer ze zich moet verstoppen doorheen elk van Bob's m bezoekjes.

De invoer van het programma is als volgt:

- `n` : het aantal kamers.
- `m` : het aantal bezoekjes dat Bob zal uitvoeren.
- `bob[]` (een array van lengte `m`) : de rij van bezoekjes die Bob zal uitvoeren.

Het programma moet `alice[]` (een array van lengte `m`) berekenen : een lijst van `m` kamers waar Alice zich moet bevinden om te voorkomen dat ze gevonden wordt door Bob, en waarbij ze zich zo weinig mogelijk moet verplaatsen.
Er kunnen meerdere oplossingen zijn die het minimum aantal verplaatsingen geven: als dat het geval is, dan is elk van die oplossingen aanvaardbaar en het programma van Alice geeft er zo maar één.

De code van het programma staat op de volgende pagina.
Hier heb je een beetje documentatie om je te helpen het te begrijpen.

- `cnt` : dient om het aantal verschillende kamers te berekenen dat Bob wilt bezoeken na de laatste verplaatsing van Alice.
- `t[]` (een array van lengte `n`) : als `t[v]`=**`false`**, dan wil dit zeggen dat Bob kamer `v` wenst te bezoeken na de laatste verplaatsing van Alice.
- De eerste **`for`**-loop berekent de verstopplekken van Alice, behalve de laatste.
- De rest van het programma (vanaf `a` $\leftarrow$ `-1`) berekent de laatste schuilplaats van Alice.

De code is niet volledig en we vragen jou om de missende expressies aan te vullen.

Q3(i) [2 ptn]	**Wat is de expressie (i) in het algoritme ?**
Oplossing: `1`	

Q3(j) [4 ptn]	**Wat is de expressie (j) in het algoritme ?**
Oplossing: `bob[i]`	

Oplossingen

```
Input  : n, m, bob[]
Output : alice[]

alice[] ← {-1, ..., -1} (een array van lengte m, geïnitialiseerd met -1'tjes)
t[] ← {true, ..., true} (un tableau de longueur n, geïnitialiseerd met true'tjes)
cnt ← 0
j ← 0
for (i ← 0 to m-1 step 1)
{
    if (t[bob[i]])
    {
        cnt ← cnt + 1
        if (cnt = n)
        {
            cnt ← ...            // (i)
            while (j < i)
            {
                alice[j] ← ...  // (j)
                j ← j + 1
            }
            for (v ← 0 to n-1 step 1)
            {
                t[v] ← true
            }
        }
        t[bob[i]] ← false
    }
}
a ← -1
for (v ← 0 to n-1 step 1)
{
    if (t[v])
    {
        a ← v
    }
}
while (j < m)
{
    alice[j] ← a
    j ← j + 1
}
return alice[]
```

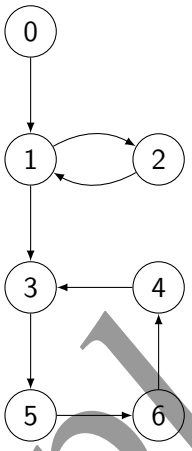
Oplossingen

Vraag 4 – Al Capone

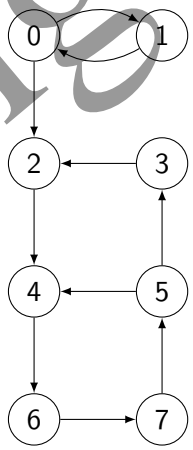
De befaamde gangster Al Capone heeft besloten om zijn netwerk van illegale kroegen te herorganiseren. Hij wil verschillende kroegen herstructureren in *wijken*, en elke wijk komt onder leiding van een bendeleider. Om zijn medeplichtigen te kunnen laten vluchten in het geval van politie-invallen, wil hij dat de volgende conditie geldt: *twee kroegen zijn in dezelfde wijk als en slechts als er in beide richtingen minstens één traject bestaat waarlangs ze met de auto kunnen vluchten van de ene kroeg naar de andere (mogelijks passerend langs andere kroegen).* Een wijk kan over een ongelimiteerd aantal kroegen beschikken zodat de conditie hierboven geldig is voor elke combinatie van twee kroegen. Bijvoorbeeld, de kroegen 0 en 1 en 2 zijn in dezelfde **wijk** als het mogelijk is om met de auto te rijden van 0 naar 1, van 1 naar 0, van 1 naar 2, van 2 naar 1, van 0 naar 2 en van 2 naar 0.

Hier heb je twee kaarten die Al Capone's kroegen in Chicago voorstellen. Elke kroeg is voorgesteld door een cirkel (met het nummer van kroeg) en de pijlen geven de routes (in één richting) aan die gevolgd kunnen worden om *onmiddellijk (d.w.z. zonder te passeren langs andere kroegen)* met de auto van een kroeg naar de andere te rijden.

Kaart 1:



Kaart 2:



Bijvoorbeeld, met **Kaart 1**, kan men (onmiddellijk) met de auto van 3 naar 5 rijden. Men kan ook met de auto van 5 naar 3 rijden maar dan moet men passeren langs 6 en 4 (want de onmiddellijke route van 5 naar 3 is verboden in die richting).

Geef aan, voor elk van de volgende stellingen, of ze waar of niet waar zijn met betrekking tot **Kaart 1**.

	Waar	Niet Waar	Stelling met betrekking tot **Kaart 1**.
Q4(a) [1 pt]	☐	☒	Men kan met de auto rijden van 1 naar 0.
Q4(b) [1 pt]	☐	☒	Men kan met de auto rijden van 6 naar 1.
Q4(c) [1 pt]	☒	☐	Men kan met de auto rijden van 6 naar 5.
Q4(d) [1 pt]	☒	☐	1 en 2 kunnen zich in dezelfde wijk bevinden.
Q4(e) [1 pt]	☐	☒	6 en 2 kunnen zich in dezelfde wijk bevinden.
Q4(f) [1 pt]	☒	☐	3, 4 en 6 kunnen zich in dezelfde wijk bevinden.
Q4(g) [1 pt]	☒	☐	3, 4, 5 en 6 kunnen zich in dezelfde wijk bevinden.
Q4(h) [1 pt]	☐	☒	1, 2, 3, 4, 5 en 6 kunnen zich in dezelfde wijk bevinden.

Oplossingen

Geef aan, voor elk van de volgende stellingen, of ze waar of niet waar zijn met betrekking tot **Kaart 2**.

	Waar	Niet Waar	Stelling met betrekking tot **Kaart 2**.
Q4(i) [1 pt]	☒	☐	Men kan met de auto rijden van 1 naar 0.
Q4(j) [1 pt]	☒	☐	Men kan met de auto rijden van 2 naar 7.
Q4(k) [1 pt]	☐	☒	Men kan met de auto rijden van 7 naar 1.
Q4(l) [1 pt]	☒	☐	2 en 5 kunnen zich in dezelfde wijk bevinden.
Q4(m) [1 pt]	☒	☐	2, 3 en 4 kunnen zich in dezelfde wijk bevinden.
Q4(n) [1 pt]	☒	☐	4, 5, 6 en 7 kunnen zich in dezelfde wijk bevinden.
Q4(o) [1 pt]	☐	☒	4, 5, 6 en 7 kunnen zich in dezelfde wijk bevinden en er bestaat geen grotere wijk die deze 4 kroegen bevat.
Q4(p) [1 pt]	☒	☐	2, 3, 4, 5, 6 en 7 kunnen zich in dezelfde wijk bevinden.
Q4(q) [1 pt]	☒	☐	2, 3, 4, 5, 6 en 7 kunnen zich in dezelfde wijk bevinden en er bestaat geen grotere wijk die deze 6 kroegen bevat.

Spijtig genoeg beschikt Capone niet over kaarten van zijn kroegen zo precies als deze, en dus zendt hij zijn medeplichtige Joe te voet (dat is discreter) om de straten van Chicago door te nemen. Capone legt aan Joe uit hoe hij het moet aanpakken om de stad door te nemen: « Ik geef je het adres van de eerste kroeg. Je volgt alle routes die vertrekken vanuit die kroeg, het éénrichtingsverkeer nalevend: eerst degene in het westen (links op de kaart), dan degene in het zuiden (onder), dan degene in het oosten (rechts), en dan degene in het noorden (boven) als die bestaan. Als je aan het einde van een route **een kroeg waar je nog nooit bent langs geweest** tegenkomt, dan begin je, aangekomen op die nieuwe kroeg, opnieuw met dezelfde procedure, zoniet, dan keer je terug op je stappen naar de vorige kroeg. Wanneer je alle routes vertrekkende vanuit een kroeg hebt verkend, dan keer je ook terug op je stappen. »

Omdat Joe niet zo slim is, geeft Capone hem het algoritme `Explore` (de code vind je een beetje verder) om letterlijk op te volgen. Dit algoritme drukt zich uit met behulp van een functie, die opgeroepen wordt met een parameter van de eerste kroeg gegeven door Capone. De functie genereert een array `order` waarvan de index het nummer is van een kroeg (van `0` tot `N-1`), en die de volgorde (van `1` tot `N`) geeft in dewelke de kroegen bezocht dienen te worden.
Bijvoorbeeld, als `order[i]=1`, dan betekent dit dat de kroeg met nummer `i` de eerste is die Capone heeft opgegeven, etc.

Als voorbeeld, met betrekking tot **Kaart 1** en ervan uitgaande dat kroeg 0 de eerste is, dan bezoekt het algoritme `Explore` als eerste de kroeg 0, dan 1 (de enige kroeg bereikbaar vertrekkende vanuit 0), dan de kroeg met nummer 3 (men bezoekt eerst de kroeg in het zuiden alvorens die in het oosten), dan 5, 6, 4 en uiteindelijk 2 (na terug te zijn gekeerd op zijn stappen).

Q4(r) [3 ptn]	**Geef de inhoud van `order` op het einde van de uitvoering van `Explore` op Kaart 1, ervan uitgaande dat de eerste kroeg die met nummer 0 is.**
Oplossing: `[1,2,7,3,6,4,5]`	

Q4(s) [3 ptn]	**Geef de volgorde in welke de kroegen bezocht worden door het algoritme `Explore` op Kaart 2, vertrekkende van kroeg 0.**
Oplossing: 0, 2, 4, 6, 7, 5, 3, 1.	

Oplossingen

```
Input: N, het aantal kroegen
       I, het nummer van de eerste kroeg (tussen 0 en N-1)

volgorde[] ← {-1, ..., -1} (een array van lengte N, geïnitialiseerd met -1'tjes)
cpt ← 1 (een teller geïnitialiseerd met 1)

Explore(kroeg X)
{
  if (volgorde[X] = -1)
  {
    volgorde[X] ← cpt
    cpt ← cpt+1

    for (d = west, zuid, oost, noord)
    {
      if (er is een route in richting d vertrekkende vanuit X)
      {
        Y ← de kroeg aan het einde van de route in richting d vertrekkende vanuit X
        Explore(Y)
      }
    }
  }
}
Explore(I) (voltooit de array volgorde[])
           voor een verkenning vertrekkende van kroeg met nummer I)
```

Q4(t) [3 ptn]	**Geef de inhoud van `order` op het einde van de uitvoering van `Explore` op Kaart 2, ervan uitgaande dat de eerste kroeg die met nummer 0 is.**
Oplossing: `[1,8,2,7,3,6,4,5]`	

Oplossingen

Nu dat het lijkt dat Joe dit eerste algoritme heeft begrepen, verrijkt Capone de functie `Explore` om het mogelijk te maken om de kaart op te delen in wijken. Allereerst vraagt hij Joe om een notitieboekje mee te nemen, waarmee hij een lijst kan bijhouden van de kroegen die hij reeds heeft bezocht: elke keer dat hij aankomt in een nieuwe kroeg, voegt hij die toe onderaan zijn lijstje. Bijvoorbeeld, op **Kaart 1**, wanneer Joe aankomt op de kroeg met nummer 5 (wanneer `Explore(5)wordt op geroepen`, dan bevat de lijst: 0, 1, 3.

Q4(u) [3 ptn]	**Wat is de inhoud van de Joe's lijstje wanneer hij aankomt op de kroeg met nummer 3 op Kaart 2; met andere woorden: op het moment van het oproepen van `Explore(3)` ?**
Oplossing: 0, 2, 4, 6, 7, 5.	

Capone voegt later ook een tweede array `low[]` toe aan de functie `Explore`. Die laat Joe toe om voor elke kroeg *het kleinste volgorde-nummer* (gegeven in de array `order`) te noteren waartoe men toegang heeft vanuit die kroeg. Capone heeft zich dus gerealiseerd dat al zoekend naar kroegen X waarvoor `order[X]=low[X]`, men op basis van de lijst bijgehouden door Joe, de wijken kan afleiden. Dit alles is in detail uitgelegd in het nieuwe algoritme (de code vind je een beetje verder).

Bij het uitvoeren van `Explore(I)`, gaat Joe niet enkel de arrays `order[]` en `low[]` invullen, maar hij gaat ook de wijken afbakenen zodanig dat (ter herinnering) voor elke wijk, elke kroeg bereikbaar is met de auto vanuit eender welke andere kroeg uit die wijk.

Q4(v) [3 ptn]	**Wat is de inhoud van `low[]` op het einde van de uitvoering van het algoritme op Kaart 1 (met initiële kroeg 0) ?**
Oplossing: `[1,2,2,3,3,3,3]`	

Q4(w) [3 ptn]	**Wat is, met betrekking tot Kaart 2 (initiële kroeg 0), de waarde van `low[5]` op het moment dat `Explore(3)` wordt opgeroepen ?**
Oplossing: 3	

Q4(x) [3 ptn]	**Met betrekking tot Kaart 2 (initiële kroeg 0), wat is de inhoud van Joe's lijstje op het moment dat `Explore(1)` wordt opgeroepen ?**
Oplossing: 0	

Q4(y) [3 ptn]	**Met betrekking tot Kaart 1 (initiële kroeg 0), wat zijn de wijken berekend door het algoritme van Capone ? Geef elke wijk aan als een verzameling van kroeg-nummers, bijvoorbeeld $\{0, 1, 4, 5\}$, $\{2, 3, 6\}$.**
Oplossing: $\{0\}$, $\{1, 2\}$, $\{3, 4, 5, 6\}$	

Q4(z) [3 ptn]	**Met betrekking tot Kaart 2 (initiële kroeg 0), wat zijn de wijken berekend door het algoritme van Capone ? Geef elke wijk aan als een verzameling van kroeg-nummers.**
Oplossing: $\{0, 1\}$, $\{2, 3, 4, 5, 6, 7\}$	

Oplossingen

```
Input: I, de eerste kroeg
       N, het aantal kroegen

volgorde[] ← {-1, ..., -1} (een array van lengte N, geïnitialiseerd met -1'tjes)
low[] ← {-1, ..., -1} (een array van lengte N, geïnitialiseerd met -1'tjes)
cpt ← 1 (een teller geïnitialiseerd met 1)

Explore(kroeg X)
{
  order[X] ← cpt
  low[X] ← cpt
  Voeg X toe vanonder aan de lijst
  cpt ← cpt+1
  for (d = west, zuid, oost, noord)
  {
    if (er is een route in richting d vertrekkende vanuit X)
    {
      Y ← de kroeg aan het einde van de route in richting d vertrekkende vanuit X
      if (order[Y] = -1)
      {
        Explore(Y)
        low[X] ← min(low[X],low[Y])
      }
      else if (Y is in de lijst)
      {
        low[X] ← min(low[X],order[Y])
      }
    }
  }

  if (low[X] = order[X])
  {
    Maak een nieuwe wijk aan die alle kroegen bevat die zich in de lijst bevinden
      vertrekkende van X (X inbegrepen), en verwijder ze uit de array
  }
}

Explore(I)
```

Oplossingen

Vraag 5 – Administratieve Oorlogen

In een verre, héél verre wereld, nemen meerdere landen het tegen elkaar op om te kunnen regeren over hun planeet. Zonder bloedvergieten, is hun manier van oorlog voeren verschillend aan de onze; Administratieve magistraten vanuit de landen nemen het tegen elkaar op in ongeremde verbale sparpartijen (zoals we zullen zien, is het in feite de kracht van de administratieve diensten die telt). Het is op dit moment waarschijnlijk nuttig om uit te leggen wat een **entiteit** is. Een land kan onderverdeeld zijn in provincies, ieder van deze provincies kan onderverdeeld zijn in sub-provincies, ieder van deze sub-provincies kan ook onderverdeeld zijn in sub-sub-provincies, en ga zo maar door. Elk land, provincie, sub-provincie, sub-sub-provincie, sub-sub-sub-provincie,... is een **entiteit**.

Laten we kijken naar het voorbeeldland van Heldor :

Fig 1: Landkaart Heldor

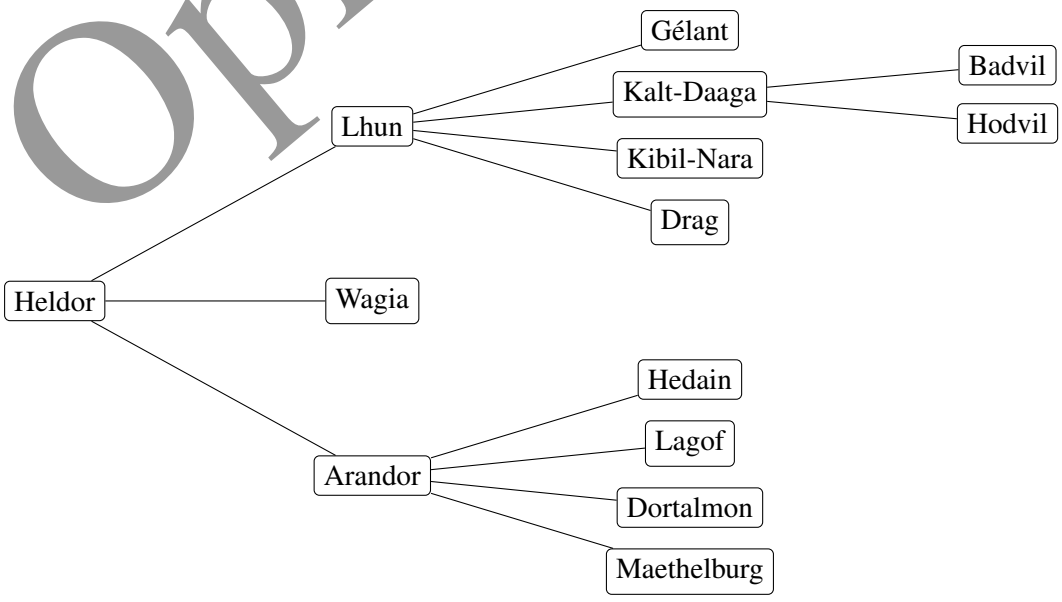


Fig 2: Kaart: Administratieve **entiteiten** van Heldor

Elke **entiteit** beschikt over zijn eigen bureaucratie die zijn eigen bevoegdheden/macht heeft. Uiteraard heeft die complexe administratie als gevolg dat hoe meer onderliggende entiteiten een entiteit beheert, hoe minder efficiënt ze is (bureaucratie is een probleem in elk land).
Bijvoorbeeld, Lhun beheert 7 entiteiten: zichzelf en 6 onderliggende entiteiten. We zeggen dat Lhun beheert over 7 **sub-entiteiten** (we moeten de centrale bureaucratie van Lhun meetellen als zijnde een speciale sub-entiteit !).

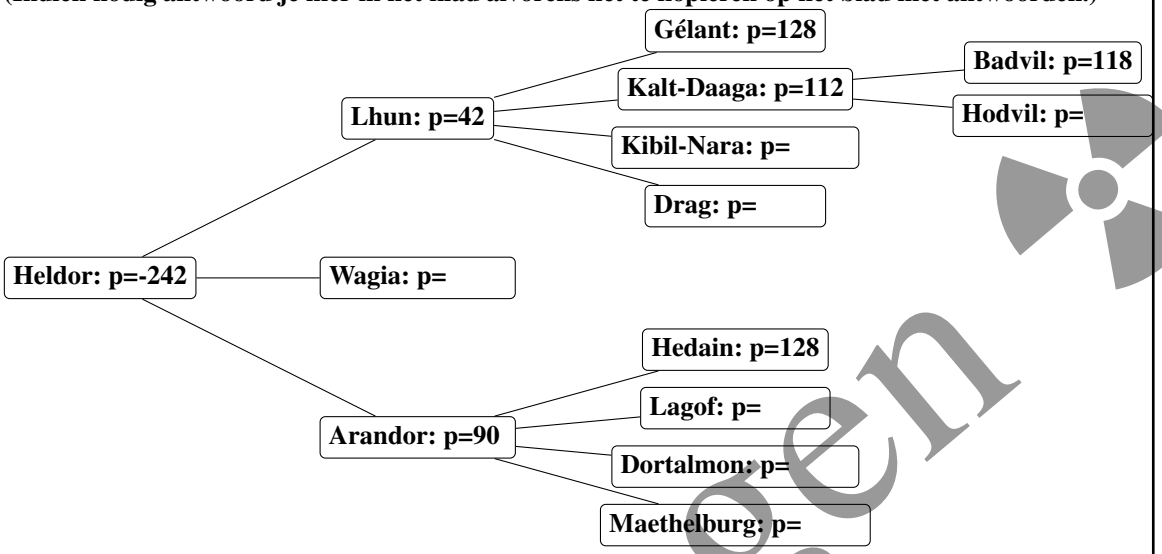
Bovendien hebben we ook de eigenschap dat hoe kleiner een entiteit is in verhouding tot het land, hoe minder geld en macht haar administratie heeft.

Een beetje specifieker; als **X** een **entiteit** is van rang n (0 voor een land, 1 voor een provincie, 2 voor een sub-provincie, 3 voor een sub-sub-provincie,...) en als s het aantal sub-entiteiten van **X** is, dan bedraagt het **administratieve vermogen** van **X**, p, $\boxed{p = 150 - 10n - 2s^2}$.

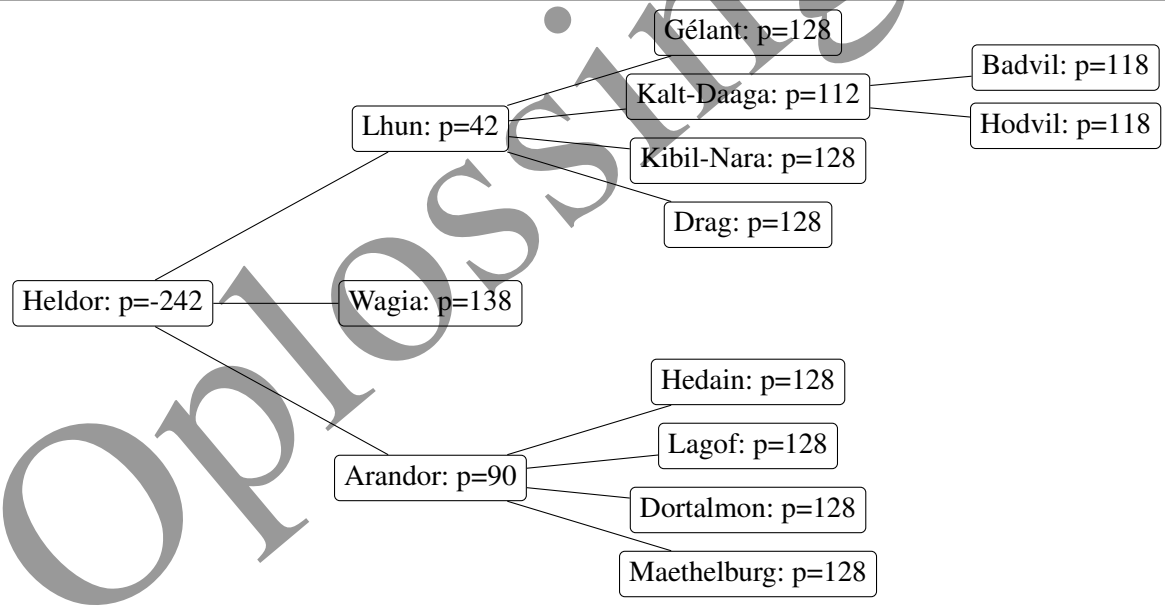
- Bijvoorbeeld de sub-sub-provincie van Badvil: $n = 3$ en $s = 1$, dus $p = 150 - 10 \cdot 3 - 2 \cdot 1^2 = 150 - 30 - 2 = 118$.
- Bijvoorbeeld de provincie van Lhun: $n = 1$ en $s = 7$, dus $p = 150 - 10 \cdot 1 - 2 \cdot 7^2 = 150 - 10 - 98 = 42$.
- Bijvoorbeeld het land Heldor: $n = 0$ en $s = 14$, dus $p = 150 - 10 \cdot 0 - 2 \cdot 14^2 = 150 - 392 = -242$.

Oplossingen

Q5(a) [7 ptn] **Vervolledig de administratieve kaart van Heldor door de administratieve vermogens aan te vullen.**
(Indien nodig antwoord je hier in het klad alvorens het te kopiëren op het blad met antwoorden.)



Oplossing:



Oplossingen

Je merkt op dat Heldor bedolven is onder administratief werk en dat haar administratief vermogen negatief is, d.w.z. dat Heldor resources verbruikt om haar administratie overeind te houden!
Op dezelfde manier, kan een sub-sub-sub…provincie een negatief administratief vermogen hebben dat negatief of nul is, zelfs als ze geen andere (buiten haar eigen) sub-entiteiten beheert. Maar hoeveel "sub" is er nodig ?

Q5(b) [2 ptn]	**Hoeveel "sub" is er nodig om voor het woord "provincie" te zetten zodat het administratief vermogen sowieso nul of negatief is** (≤ 0) **?**
Oplossing: 14 (want $150 - 10n - 2 \leq 0 \iff n \geq 14,8$ en er zijn 14 keren "sub" op rang 15)	

Wat eigenlijk echt van belang is, is het **totale vermogen**. Het **totale vermogen** van een entiteit **X** wordt berekend door de som van de vermogens van alle sub-entiteiten van **X** (vergeet dus niet **X** zelf). Bijvoorbeeld, het **totale vermogen** van Badvil is gelijk aan enkel haar vermogen (118), aangezien ze geen sub-entiteiten heeft. Het **totale vermogen** van Lhun

is daarentegen een som van 7 termen (help: de waarde is 774).

Q5(c) [1 pt]	**Wat is het totale vermogen van Hodvil ?**
Oplossing: 118	

Q5(d) [1 pt]	**Wat is het totale vermogen van Wagia ?**
Oplossing: 138	

Q5(e) [2 ptn]	**Wat is het totale vermogen van Arandor ?**
Oplossing: $90 + 4 \cdot 128 = 602$	

Q5(f) [2 ptn]	**Wat is het totale vermogen van Heldor ?**
Oplossing: $-242 + 774 + 138 + 602 = 1272$	

Wanneer twee landen het tegen elkaar opnemen, dan wint die met het grootste **totale vermogen**. Bij een gelijke stand, geraken de administraties verstrikt in een procedure-fout. Veronderstel een functie `TOTAAL_VERMOGEN(X, n)` waarmee je het **totaal vermogen** van een staat `X` met rang `n` kunt berekenen.

Q5(g) [2 ptn]	**Geef een uitdrukking die aangeeft of een land `A` wint in een gevecht tegen land `B`. Herinner je dat landen een rang `n=0` hebben en dat er geen winnaar is in het geval van een gelijke stand.**
Oplossing: `TOTAAL_VERMOGEN(A, 0)> TOTAAL_VERMOGEN(B, 0)`	

Elke entiteit van rang n beschikt over de lijst van haar sub-entiteiten van rang $n+1$ die men haar **kinderen** noemt. Die rare naam komt van grafentheorie, dat men gebruikt om administratieve kaarten op te stellen. Op een administratieve kaart, verbindt een streepje elke entiteit met haar kinderen (zie Fig 2). De kinderen van een land zijn provincies, de kinderen van een provincie zijn sub-provincies,...

Om het administratief vermogen van een entiteit `X` te berekenen, moet men de naam kennen van haar sub-entiteiten (X inbegrepen). Hier heb je onvolledige pseudo-code (er mankeert een lijntje) van de functie `BEREKEN_S(X)` die dat doet.

```
functie BEREKEN_S(X) {
  nb_entiteiten ← 1
  voor elk kind E van X   {
    ...   //(i)
  }
  return nb_entiteiten
}
```

Merk op dat de regel "`voor elk kind E van X`" een loop opstart die één keer uitgevoerd zal worden voor elk kind van `X`, en dat in elke loop het kind aangewezen zal worden met de variabele `E`. Bijvoorbeeld, als `X=Heldor`, dan wordt de loop 3 keer uitgevoerd, één keer met `E=Lhun`, één keer met `E=Wagia`, en één keer met `E=Arandor`. Als `X` geen kinderen heeft, dan wordt de loop niet uitgevoerd (ze wordt 0 keer uitgevoerd omdat ze 0 kinderen heeft.).

Oplossingen

Q5(h) [2 ptn]		**Wat is de regel `(i)` in de pseudo-code van de functie `BEREKEN_S()` ?**
	☐	`nb_entiteiten ← BEREKEN_S(E)+ 1`
	☐	`nb_entiteiten ← nb_entiteiten + 1`
	☒	`nb_entiteiten ← nb_entiteiten + BEREKEN_S(E)`
	☐	`nb_entiteiten ← BEREKEN_S(E)`
	☐	`nb_entiteiten ← nb_entiteiten + BEREKEN_S(E)+ 1`

Oplossingen

Vul de pseudo-code van de functie `TOTAAL_VERMOGEN(X, n)` aan, waarmee (zoals reeds uitgelegd) je het **totaal vermogen** van een entiteit `X` van rang `n` kunt berekenen.

```
functie TOTAAL_VERMOGEN(X, n) {
    vermogen_van_X ← ...                //(i)
    vermogen_van_de_kinderen ← 0
    voor elk kind E van X   {
      vermogen_van_de_kinderen ← vermogen_van_de_kinderen + ...     //(j)
    }
    return ...                          //(k)
}
```

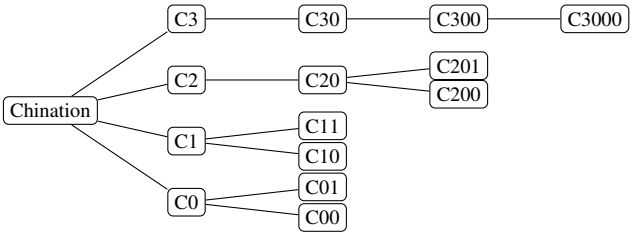
Merk op: je kan en zult gebruik moeten maken van `BEREKEN_S(X)`. Ga ervan uit dat die functie correct geïmplementeerd is, zelfs als je niet hebt kunnen antwoorden op de voorgaande vraag. Om het kwadraat van een getal (`r`) te berekenen, vermenigvuldig je het met zichzelf (`r*r`) of doe je het tot de 2-de macht (`r^2`).

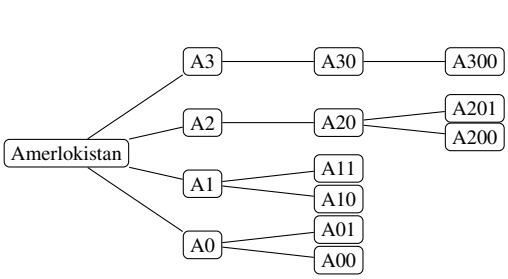
Q5(i) [2 ptn]	**Wat is de formulering voor `(j)` in de pseudo-code van de functie `TOTAAL_VERMOGEN(X, n)` ?**
Oplossing: `150 - 10*n - 2*BEREKEN_S(X)*BEREKEN_S(X)`	

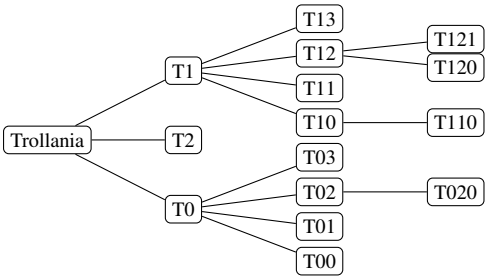
Q5(j) [2 ptn]	**Wat is de formulering voor `(k)` in de pseudo-code van de functie `TOTAAL_VERMOGEN(X, n)` ?**
Oplossing: `TOTAAL_VERMOGEN(E, n+1)`	

Q5(k) [2 ptn]	**Wat is de formulering voor `(l)` in de pseudo-code van de functie `TOTAAL_VERMOGEN(X, n)` ?**
Oplossing: `vermogen_van_X + vermogen_van_de_kinderen`	

Wanneer twee landen oorlog voeren, dan wordt het verliezende land opgeslorpt door het winnende land en wordt het één van zijn provincies. Dat verandert natuurlijk het administratief vermogen van het winnende land ! Laten we een voorbeeld beschouwen. Hier heb je de administratieve kaarten van drie landen (we gebruiken postcodes in plaats van namen; dit is gewoon een beetje korter).







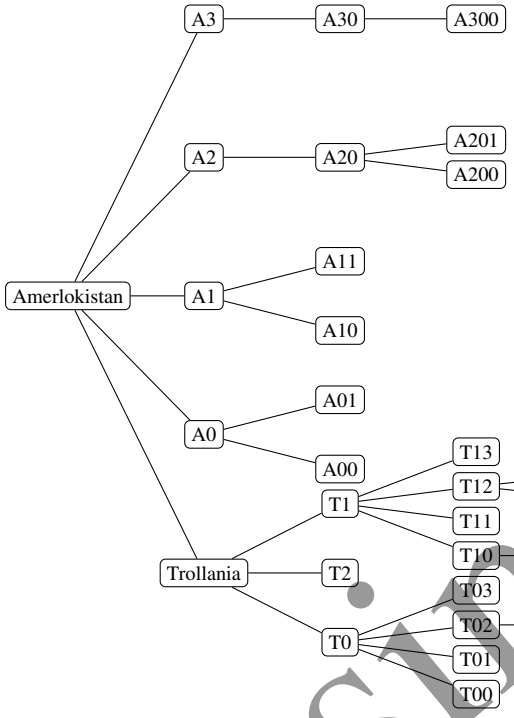
Oplossingen

Onze spionnnen hebben de **totale vermogens** van de drie landen voor jullie berekend :

Chination: **1352**	Amerlokistan: **1332**	Trollania: **1324**

Oplossingen

Als Amerlokistan en Trollania een oorlog aangaan, dan slorpt Amerlokistan Trollania op.
Hieronder zie je dan hoe de administratieve kaart de Amerlokistan er zou uitzien na die overwinning :



Het nieuwe **totaal vermogen** van Amerlokistan na de oorlog zou dan gelijk zijn aan **1088**.

Er bestaat een formule om het **totale vermogen** van de winnaar van een oorlog te bereken.

Q5(l) [5 ptn]		**Zij `X1` een land met `totaal vermogen t1` en dat `s1` sub-entiteiten (`X1` bevat). Zij `X2` een land met `totaal vermogen t2` en dat `s2` sub-entiteiten (`X2` bevat). Als `t1`>`t2`, welke formule geeft dan het nieuwe totaal vermogen van de winnaar tussen de confrontatie tussen `X1` en `X2` ?**
	☐	`t1 + t2`
	☐	`t1 + t2 - 10*s2`
	☐	`t1 + t2 - 2*s2*s2`
	☐	`t1 + t2 + 2*s1*s1 - 2*(s1+s2)*(s1+s2)`
	☐	`t1 + t2 + 2*s2*s2 - 2*(s1+s2)*(s1+s2)`
	☒	`t1 + t2 - 10*s2 + 2*s1*s1 - 2*(s1+s2)*(s1+s2)`
	☐	`t1 + t2 - 10*s2 + 2*s2*s2 - 2*(s1+s2)*(s1+s2)`

Oplossingen

Om alles met de hand te doen is nogal saai, is het niet ? Het is hoog tijd dat we een procedure gaan implementeren.

Alle landen worden bijgehouden in een lijst: `lijst_landen`. De lijst ondersteunt drie operaties :

- `lijst_landen.lengte()` die het aantal landen in de lijst terug geeft,
- `lijst_landen.grootste_admin_macht()` die het land uit de lijst met het grootste totaal vermogen terug geeft **en** verwijdert uit de lijst (als meerdere landen hetzelfde grootste aantal vermogen hebben, dan wordt er één van de twee "at random" gekozen).
- `lijst_landen.toevoegen(X)` die het land `X` toevoegt aan de lijst.

De variabelen voor een "land" bevatten bovendien een beschrijving van de administratieve kaart van dat land; Als `A` en `B` twee variabelen zijn voor een "land", dan verandert de instructie `A.provincie_toevoegen(B)` de variabele `A` door er een nieuwe provincie aan toe te voegen, overeenkomstig met het land `B`.

We vragen jou om de functie `OORLOGEN` aan te vullen die de opeenvolgende oorlogen in de landen in de lijst simuleert, totdat er slechts één (of geen) land overblijft in de lijst.

```
functie OORLOGEN() {
    while (...)                              //(e1)
    {
        A ← ...                             //(e2)
        B ← ...                             //(e2)
        p_tot_A ← TOTAAL_VERMOGEN(A, 0)
        p_tot_B ← TOTAAL_VERMOGEN(B, 0)
        if (p_tot_A > p_tot_B)
        {
            A.provincie_toevoegen(B)
            ...                              //(e3)
        }
    }
}
```

Q5(m) [2 ptn]	**Wat is de expressie `(e1)` in de functie `OORLOGEN()` ?**
Oplossing: `lijst_landen.lengte()> 1`	

Q5(n) [2 ptn]	**Wat is de expressie `(e2)` die zich twee keer bevindt in de functie `OORLOGEN()` ?**
Oplossing: `lijst_landen.grootste_admin_macht()`	

Q5(o) [2 ptn]	**Wat is de expressie `(e3)` in de functie `OORLOGEN()` ?**
Oplossing: `lijst_landen.toevoegen(A)`	

Oplossingen