



**INFORMATIK-BIBER SCHWEIZ
 CASTOR INFORMATIQUE SUISSE
 CASTORO INFORMATICO SVIZZERA**

Exercices et solutions 2022

Années HarmoS 11/12

<https://www.castor-informatique.ch/>

Éditeurs :

Susanne Datzko, Nora A. Escherle,
 Elsa Pellet, Jean-Philippe Pellet

010100110101011001001001
 010000010010110101010011
 010100110100100101000101
 001011010101001101010011
 010010010100100100100001

SS!E

www.svia-ssie-ssii.ch
 schweizerischerverein für informatik in
 erausbildung // société suisse pour l'infor
 matique dans l'enseignement // società sviz
 zera per l'informatica nell'insegnamento



Ont collaboré au Castor Informatique 2022

Masiar Babazadeh, Susanne Datzko, Jean-Philippe Pellet, Giovanni Serafini, Bernadette Spieler

Cheffe de projet : Nora A. Escherle

Nous adressons nos remerciements pour le travail de développement des exercices du concours à :
Juraj Hromkovič, Christian Datzko, Jens Gallenbacher, Regula Lacher : ETH Zurich, Ausbildungs- und Beratungszentrum für Informatikunterricht

Tobias Berner : Pädagogische Hochschule Zürich

Waël Almoman : Collège Voltaire

Le choix des exercices a été fait en collaboration avec les organisateurs de Bebras en Allemagne, Autriche, Hongrie, Slovaquie et Lituanie. Nous remercions en particulier :

Valentina Dagienė, Tomas Šiaulyš, Vaidotas Kinčius : Bebras.org

Wolfgang Pohl, Hannes Endreß, Ulrich Kiesmüller, Kirsten Schlüter, Michael Weigend : Bundesweite Informatikwettbewerbe (BWINF), Allemagne

Wilfried Baumann, Liam Baumann, Anoki Eischer, Thomas Galler, Benjamin Hirsch, Martin Kandlhofer, Katharina Resch-Schobel : Österreichische Computer Gesellschaft

Gerald Futschek, Florentina Voboril : Technische Universität Wien

Zsuzsa Pluhár : ELTE Informatikai Kar, Hongrie

Michal Winzcer : Université Comenius de Bratislava, Slovaquie

La version en ligne du concours a été réalisée sur l'infrastructure cuttle.org. Nous remercions pour la bonne collaboration :

Eljakim Schrijvers, Justina Dauksaite, Dave Oostendorp, Alieke Stijf, Kyra Willekes, Jo-Ann Bolten : cuttle.org, Pays-Bas

Chris Roffey : UK Bebras Administrator, Royaume-Uni

Pour le support pendant les semaines du concours, nous remercions en plus :

Hanspeter Erni : Direction, école secondaire de Rickenbach

Christoph Frei : Chragokyberneticks (Logo Castor Informatique Suisse)

Dr. Andrea Leu, Maggie Winter, Lena Frölich : Senarclens Leu + Partner AG

La version allemande des exercices a également été utilisée en Allemagne et en Autriche.

L'adaptation française a été réalisée par Elsa Pellet et l'adaptation italienne par Christian Giang.



INFORMATIK-BIBER SCHWEIZ
CASTOR INFORMATIQUE SUISSE
CASTORO INFORMATICO SVIZZERA

Le Castor Informatique 2022 a été réalisé par la Société Suisse pour l'Informatique dans l'Enseignement (SSIE) et soutenu de manière déterminante par la Fondation Hasler. Les sponsors du concours sont l'Office de l'économie et du travail du canton de Zurich et l'UBS.

Cette brochure a été produite le 22 novembre 2023 avec le système de composition de documents \LaTeX . Nous remercions Christian Datzko pour le développement et maintien de la structure de génération des 36 versions de cette brochure (selon les langues et les degrés). La structure actuelle a été mise en place de manière similaire à la structure précédente, qui a été développée conjointement avec Ivo Blöchliger dès 2014. Nous remercions aussi Jean-Philippe Pellet pour le développement de la série d'outils `bebras`, qui est utilisée depuis 2020 pour la conversion des documents source depuis les formats Markdown et YAML.

Tous les liens dans les tâches ci-après ont été vérifiés le 1^{er} décembre 2022.



Les exercices sont protégés par une licence Creative Commons Paternité – Pas d'Utilisation Commerciale – Partage dans les Mêmes Conditions 4.0 International. Les auteur·e·s sont cité·e·s en p. 70.



Préambule

Très bien établi dans différents pays européens et plus largement à l'échelle mondiale depuis plusieurs années, le concours « Castor Informatique » a pour but d'éveiller l'intérêt des enfants et des jeunes pour l'informatique. En Suisse, le concours est organisé en allemand, en français et en italien par la SSIE, la Société Suisse pour l'Informatique dans l'Enseignement, et soutenu par la Fondation Hasler.

Le Castor Informatique est le partenaire suisse du concours « Bebras International Contest on Informatics and Computer Fluency » (<https://www.bebas.org/>), initié en Lituanie.

Le concours a été organisé pour la première fois en Suisse en 2010. Le Petit Castor (années HarmoS 5 et 6) a été organisé pour la première fois en 2012.

Le Castor Informatique vise à motiver les élèves à apprendre l'informatique. Il souhaite lever les réticences et susciter l'intérêt quant à l'enseignement de l'informatique à l'école. Le concours ne suppose aucun prérequis quant à l'utilisation des ordinateurs, sauf de savoir naviguer sur Internet, car le concours s'effectue en ligne. Pour répondre, il faut structurer sa pensée, faire preuve de logique mais aussi de fantaisie. Les exercices sont expressément conçus pour développer un intérêt durable pour l'informatique, au-delà de la durée du concours.

Le concours Castor Informatique 2022 a été fait pour cinq tranches d'âge, basées sur les années scolaires :

- Années HarmoS 5 et 6 (Petit Castor)
- Années HarmoS 7 et 8
- Années HarmoS 9 et 10
- Années HarmoS 11 et 12
- Années HarmoS 13 à 15

Chaque tranche d'âge avait des exercices classés en trois niveaux de difficulté : facile, moyen et difficile. Les élèves des années HarmoS 5 et 6 avaient 9 exercices à résoudre : 3 faciles, 3 moyens, 3 difficiles. Les élèves des années HarmoS 7 et 8 avaient, quant à eux, 12 exercices à résoudre (4 de chaque niveau de difficulté). Finalement, chaque autre tranche d'âge devait résoudre 15 exercices (5 de chaque niveau de difficulté).

Chaque réponse correcte donnait des points, chaque réponse fautive réduisait le total des points. Ne pas répondre à une question n'avait aucune incidence sur le nombre de points. Le nombre de points de chaque exercice était fixé en fonction du degré de difficulté :

	Facile	Moyen	Difficile
Réponse correcte	6 points	9 points	12 points
Réponse fautive	-2 points	-3 points	-4 points

Utilisé au niveau international, ce système de distribution des points est conçu pour limiter le succès en cas de réponses données au hasard.



Chaque participant·e obtenait initialement 45 points (ou 27 pour la tranche d'âge «Petit Castor», et 36 pour les années HarmoS 7 et 8).

Le nombre de points maximal était ainsi de 180 (ou 108 pour la tranche d'âge «Petit Castor», et 144 pour les années HarmoS 7 et 8). Le nombre de points minimal était zéro.

Les réponses de nombreux exercices étaient affichées dans un ordre établi au hasard. Certains exercices ont été traités par plusieurs tranches d'âge (en étant classés différemment dans les niveaux de difficulté).

Certains exercices sont indiqués comme «bonus» pour certaines catégories d'âge : ils ne comptent pas dans le total des points, mais servent à départager plusieurs scores identiques en cas de qualification pour les éventuels tours suivants.

Pour de plus amples informations :

SVIA-SSIE-SSII Société Suisse pour l'Informatique dans l'Enseignement
Castor Informatique
Jean-Philippe Pellet

<https://www.castor-informatique.ch/fr/kontaktieren/>
<https://www.castor-informatique.ch/>



Table des matières

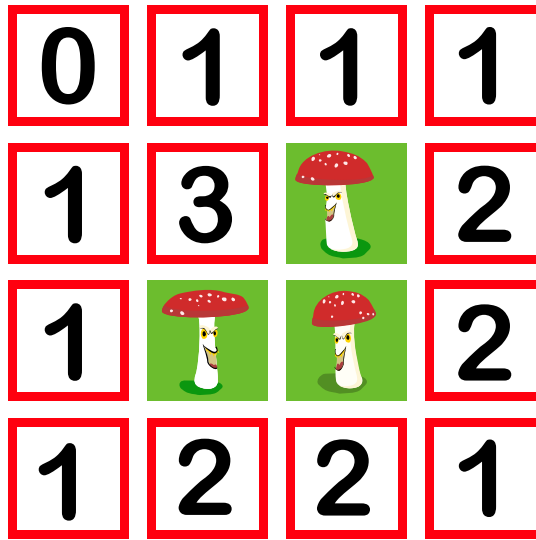
Ont collaboré au Castor Informatique 2022	i
Préambule	iii
Table des matières	v
1. Déchampignonneur	1
2. Boulons et écrous	5
3. Que la lumière soit !	9
4. Code 8	15
5. Les voisins de Lili	19
6. Poste robotisée	23
7. Séquences	27
8. Hangar tournant	31
9. Soirée ciné	35
10. Morpion	39
11. Pierres précieuses	43
12. Galets et coquillages	47
13. Coffre au trésor	51
14. Empaquetage	55
15. Labyrinthe	59
16. Virus	63
17. Carrelage	67
A. Auteur-e-s des exercices	70
B. Sponsoring: Concours 2022	72
C. Offres supplémentaires	73



1. Déchampignonneur

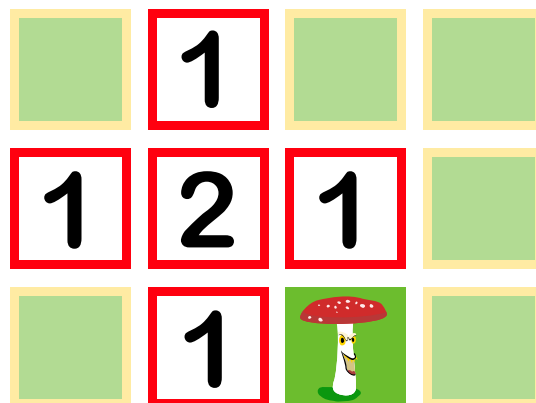
Au début de jeu du «déchampignonneur», un seul champignon et quelques cases contenant des chiffres sont visibles. Toutes les autres cases du jeu sont cachées. Lorsque tu découvres une case, un champignon ou le nombre de champignons présents dans les cases voisines apparaît. Tu gagnes le jeu si tu arrives à découvrir uniquement toutes les cases sans champignon.

Voici un exemple de jeu complètement découvert :



Tu commences un nouveau jeu – regarde en dessous.

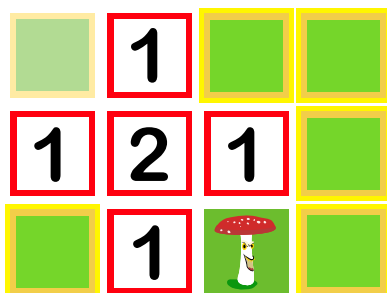
Sur quelles cases ne peut-il pas y avoir de champignon ?





Solution

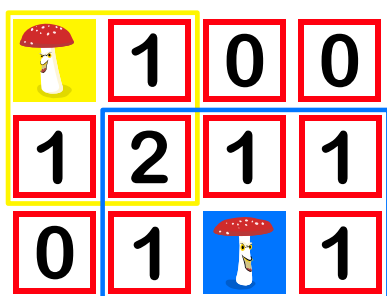
Voici la bonne solution :



Afin d'expliquer la bonne réponse, nous assignons des lettres aux cases couvertes. Nous disons qu'un chiffre N sur une case est épuisé lorsque que nous savons quelles N cases voisines contiennent un champignon ; il ne peut alors plus y avoir de champignon sur les autres cases voisines.



- Il n'y a pas de champignon sur la case D, parce que le chiffre 1 à côté est épuisé.
- Il n'y a pas de champignon sur les cases B, C, E et F, parce que le chiffre 1 sur leur case voisine commune est épuisé.
- Il y a un champignon sur la case A, car les chiffres 1, 2 et 1 sur les cases voisines n'indiqueraient sinon pas le bon nombre de champignon sur les cases voisines.



Il y a donc un champignon caché sur la case A. Les cases B, C, D, E et F peuvent être découvertes.

C'est de l'informatique !

Comment avons-nous résolu cet exercice ? Parfois, il faut commencer avec une supposition et continuer logiquement. Si l'on se trouve face à une contradiction, on revient en arrière et continue avec la supposition suivante. Il s'agit alors d'une recherche « ciblée » et non pas aléatoire.



Comment un ordinateur résoudre-t-il cet exercice ? Si au moins une case avec un champignon est découverte, de simples règles peuvent être énoncées. Par exemple, s'il y a déjà une case avec un champignon découvert à côté d'une case avec le chiffre 1, il ne peut plus y avoir de champignon sur les autres cases voisines. Si l'on formule cette règle précisément pour tous les chiffres, un ordinateur peut l'utiliser comme *instruction* à exécuter pas à pas. Nous obtenons ainsi un *algorithme* que nous pouvons « simplement » exécuter pour gagner le jeu (si au moins un champignon est déjà découvert).



Mots clés et sites web

- Démineur : [https://fr.wikipedia.org/wiki/Démineur_\(genre_de_jeu_vidéo\)](https://fr.wikipedia.org/wiki/Démineur_(genre_de_jeu_vidéo))
- Instruction : https://fr.wikipedia.org/wiki/Instruction_informatique
- Algorithme : <https://fr.wikipedia.org/wiki/Algorithme>





2. Boulons et écrous

Ben assemble des pièces sur une ligne de montage : des écrous  et des boulons .



Ben applique strictement la méthode suivante :

- Ben prend la pièce suivante sur la ligne de montage.
- Si c'est un écrou, il le met dans le seau.
- Si c'est un boulon, il prend un écrou dans le seau, le visse sur le boulon, et met la pièce terminée dans la boîte.

Deux erreurs peuvent se produire avec cette méthode :

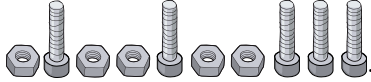
1. Ben prend un boulon sur la ligne de montage, mais il n'y a pas d'écrou à visser dessus dans le seau.
2. Ben a pris toutes les pièces sur la ligne de montage, mais il reste des écrous dans le seau.

Le seau pour les écrous est assez grand et est vide au départ. Laquelle des séquences suivantes Ben peut-il assembler de gauche à droite sans aucune erreur ?

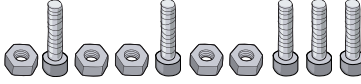

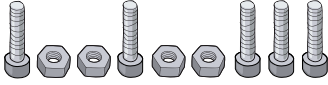

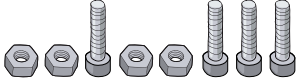


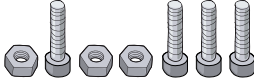


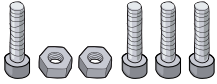


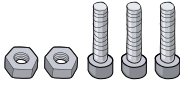


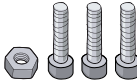



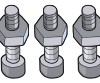


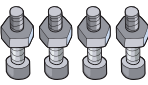


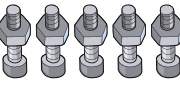
- A)
- B)
- C)
- D)




Solution

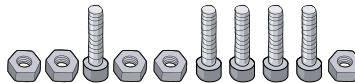
La bonne réponse est C) : 

La table ci-dessous montre le contenu de la boîte pour les pièces terminées, du seau pour les écrous et de la ligne de montage.

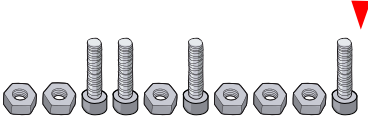
Boîte	Seau	Ligne de montage
<i>vide</i>	<i>vide</i>	
<i>vide</i>		
	<i>vide</i>	
		
		
		
		
		
		
		
	<i>vide</i>	<i>vide</i>

Pourquoi les autres réponses sont-elles fausses ?

A) La séquence  cause une erreur à la position indiquée. Ben a pris un boulon, mais il n'y a plus d'écrou dans le seau.

B) La séquence  cause une erreur à la position indiquée. Ben a vissé quatre écrous sur quatre boulons, le seau est donc vide lorsqu'il prend le dernier boulon pour laquelle il n'a plus d'écrou.



D) La séquence  cause une erreur une fois la ligne de montage vide. En effet, quatre écrous ont été vissés sur quatre boulons et il reste encore deux écrous.

C'est de l'informatique !

Ben travaille avec des pièces qui sont livrées les unes après les autres sur la ligne de montage. Pour cela, il utilise un grand seau pour stocker les écrous. En *informatique théorique*, une séquence similaire est utilisée comme modèle pour les *algorithmes* qui peuvent résoudre un certain type de problèmes : les *automates à pile*.

Un automate à pile traite des données (des chiffres ou symboles) qu'il reçoit en entrée les unes après les autres. Il possède un seul espace de stockage infini, une pile. Au contraire du seau de l'exercice, les éléments de la pile ont un ordre précis, et seul le dernier élément ajouté à la pile peut en être ressorti (« last in, first out », LIFO). Un automate à pile peut être utilisé pour reconnaître un *langage non contextuel*.

En informatique, un langage est constitué d'un ensemble de chaînes de symboles qui ont été assemblé d'après certaines règles. Les langages non contextuels appartiennent à une classe de langages simples. Par exemple, toutes les expressions entre parenthèses bien formulées sont un langage non contextuel. Une expression entre parenthèses bien formulée commence toujours par une parenthèse ouverte qui est ensuite refermée. Par exemple, les expressions $((()))$ et $((()()))$ sont bien formulées. Par contre, les expressions $((((($ et $((()(($ ne le sont pas. On peut se représenter les boulons et les écrous dans l'exercice comme des parenthèses ouvrantes et fermantes ; Ben traite alors une séquence de pièces sans erreurs seulement lorsqu'elle représente une expression entre parenthèses bien formulée. La vérification des parenthèses est une tâche importante pour un compilateur qui traduit des textes de programmes en programmes exécutables. En effet, dans la plupart des langages de programmation, les textes de programmes contiennent des appels de fonctions emboîtés et des expressions arithmétiques entre parenthèses.

Mots clés et sites web

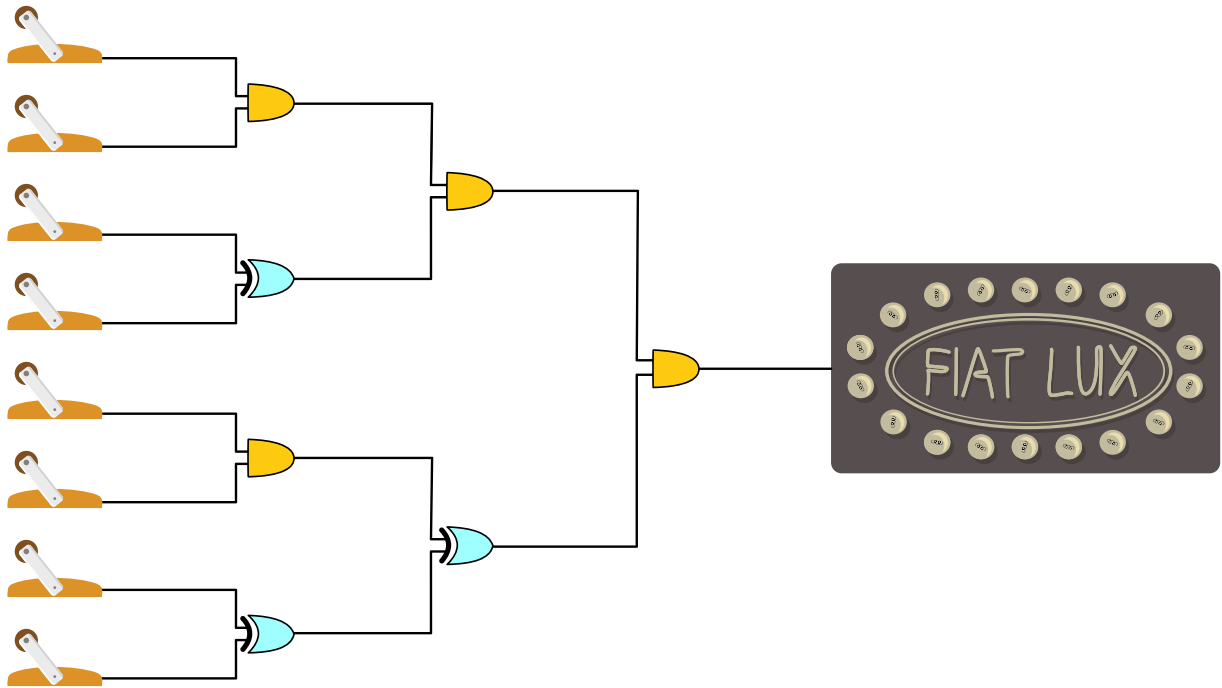
- Informatique théorique : https://fr.wikipedia.org/wiki/Informatique_théorique
- Automate à pile : https://fr.wikipedia.org/wiki/Automate_à_pile
- Langage non contextuel : https://fr.wikipedia.org/wiki/Langage_algébrique







3. Que la lumière soit !

Le jeu « Que la lumière soit ! » est composé de 8 interrupteurs pouvant être actifs ou inactifs. Des fils relient ces interrupteurs à un panneau publicitaire lumineux en passant par différents composants.



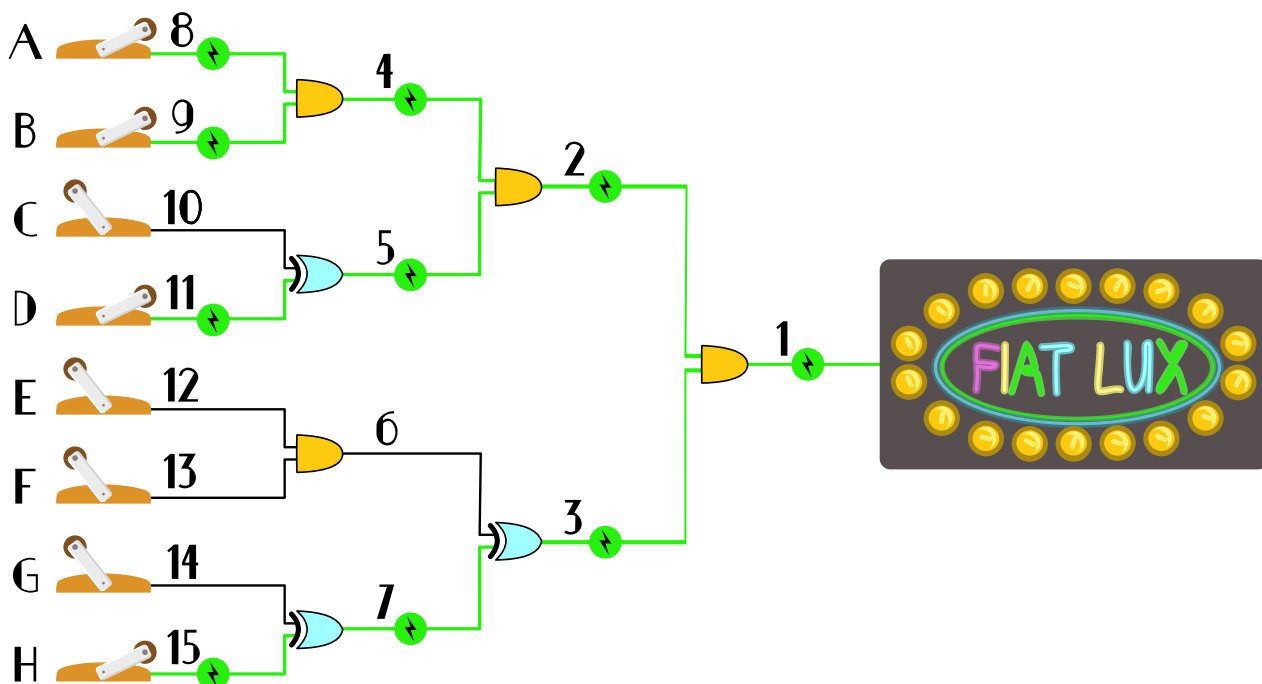
Les fils sortant des interrupteurs sont actifs lorsque l'interrupteur correspondant est allumé. La sortie du composant  est active seulement lorsque les deux fils entrants sont actifs. La sortie du composant  est active seulement lorsqu'un seul des deux fils entrants est actif.


Quels interrupteurs doivent être allumés  afin d'allumer le panneau publicitaire ?















Solution





Une des solutions possibles est la suivante :



On arrive facilement à la solution en commençant à résoudre le problème par la fin. Le fil 1 est relié à un composant . Pour que la sortie de ce composant soit *active*, les deux fils entrants, 2 et 3, doivent être *actifs*.

- Le fil 2 est relié à un composant . Pour que sa sortie soit *active*, les deux fils entrants, 3 et 4, doivent être *actifs*.
- Le fil 3 est relié à un composant . Pour que sa sortie soit *active*, seul l'un des deux fils entrants peut être *actif*, par exemple le fil 7. Le fil 6 doit alors être *inactif*.
- Le fil 4 est relié à un composant . Pour que sa sortie soit *active*, les deux fils entrants, 8 et 9, doivent être *actifs* ; les deux interrupteurs A et B doivent donc aussi être *allumés* : .
- Le fil 5 est relié à un composant . Pour que sa sortie soit *active*, seul l'un des deux fils entrants peut être *actif*, par exemple le fil 11. Le fil 10 doit alors être *inactif*. L'interrupteur C doit donc être *éteint*  et l'interrupteur D *allumé* .
- Le fil 6 est relié à un composant . Pour que sa sortie soit *inactif*, au moins l'un des deux fils entrants, 12 et 13, doit être *inactif* ; les deux interrupteurs E et F peuvent donc être les deux *éteints* : .
- Le fil 7 est relié à un composant . Pour que sa sortie soit *active*, seul l'un des deux fils entrants peut être *actif*, par exemple le fil 15. Le fil 14 doit alors être *inactif*. L'interrupteur G doit donc être *éteint*  et l'interrupteur H *allumé* .





Les composants  laissent des alternatives, car ils permettent de décider lequel des deux fils entrants est *actif*. De plus, pour le composant  avec fil 6 sortant, on peut décider si aucun des deux fils entrants ou seul l'un des deux est *inactif* afin que la sortie reste *inactive*. Si la sortie du composant  est *active*, les deux fils entrants doivent également être *actifs*, les deux entrées du composant  avec le fil 7 sortant doivent être soit les deux *actifs*, soit les deux *inactifs*. Cela donne au total 16 différentes combinaisons possibles :

Interrupteur								Fil	
A	B	C	D	E	F	G	H	6	7
Toujours allumés		Un seul allumé		Les deux allumés si fil 6 <i>actif</i> , sinon au plus un allumé		Un seul allumé si fil 7 <i>actif</i> , sinon les deux allumés ou éteints		Un seul actif	
allumé	allumé	allumé	éteint	allumé	allumé	allumé	allumé	actif	inactif
allumé	allumé	éteint	allumé	allumé	allumé	allumé	allumé	actif	inactif
allumé	allumé	allumé	éteint	allumé	allumé	éteint	éteint	actif	inactif
allumé	allumé	éteint	allumé	allumé	allumé	éteint	éteint	actif	inactif
allumé	allumé	allumé	éteint	allumé	éteint	allumé	éteint	inactif	actif
allumé	allumé	éteint	allumé	allumé	éteint	allumé	éteint	inactif	actif
allumé	allumé	allumé	éteint	allumé	éteint	éteint	allumé	inactif	actif
allumé	allumé	éteint	allumé	allumé	éteint	éteint	allumé	inactif	actif
allumé	allumé	allumé	éteint	éteint	allumé	allumé	éteint	inactif	actif
allumé	allumé	éteint	allumé	éteint	allumé	allumé	éteint	inactif	actif
allumé	allumé	allumé	éteint	éteint	éteint	allumé	éteint	inactif	actif
allumé	allumé	éteint	allumé	éteint	éteint	allumé	éteint	inactif	actif
allumé	allumé	allumé	éteint	éteint	éteint	éteint	allumé	inactif	actif
allumé	allumé	éteint	allumé	éteint	éteint	éteint	allumé	inactif	actif

C'est de l'informatique !

Le courant peut passer ou non à travers les fils de cet exercice, les interrupteurs sont donc soit éteints, soit allumés. En informatique, de tels états représentent les valeurs de *variables booléennes*, qui sont souvent nommés *VRAI* et *FAUX* ou *1* et *0*.

Les ordinateurs actuels fonctionnent en règle générale uniquement avec ces variables, comme le jeu de cet exercice. Cela vient entre autre du fait que des milliards de *transistors* dont l'état est aussi *actif* ou *inactif* sont présents au cœur de l'ordinateur.

On peut construire des portes logiques avec plusieurs transistors. Deux de ces portes sont présentes dans cet exercice : le composant  est une *porte ET* dont la sortie est VRAI lorsque les deux entrées sont VRAI. Le composant  est une *porte OU exclusif* dont la sortie est VRAI lorsqu'exactlyement une des deux entrées est VRAI. On peut aussi les représenter dans une *table de vérité* :



Entrées		Porte ET		Porte OU exclusif	
Entrée A	Entrée B	Image	Sortie C	Image	Sortie C
VRAI	VRAI		VRAI		FAUX
VRAI	FAUX		FAUX		VRAI
FAUX	VRAI		FAUX		VRAI
FAUX	FAUX		FAUX		FAUX

D'autres portes répandues sont la *porte OU*, dont la sortie est VRAI lorsqu'au moins l'un des deux entrées est VRAI, et la *porte NON*, dont la sortie est VRAI lorsque l'entrée est FAUX. Souvent, on utilise des combinaisons de portes ET et de porte NON dans la construction de circuits, car cela demande peu de transistors. Les tables de vérité sont :

Entrée A	Entrée B	Sortie porte OU	Sortie porte NON-ET
VRAI	VRAI	VRAI	FAUX
VRAI	FAUX	VRAI	VRAI
FAUX	VRAI	VRAI	VRAI
FAUX	FAUX	FAUX	VRAI

Entrée	Sortie porte NON
VRAI	FAUX
FAUX	VRAI

Un ordinateur peut effectuer des calculs compliqués très rapidement en utilisant des combinaisons de *portes logiques* adaptées.

À un plus haut niveau, les portes logiques sont aussi utilisées en programmation : lorsque l'exécution d'une partie de programme dépend de plusieurs conditions, ces conditions peuvent être décrites par des combinaisons d'*opérateurs logiques* qui fonctionnent de la même manière que les portes logiques. On les trouve dans les programmes informatiques ; parfois, un ordinateur doit décider quelle action exécuter sur la base de ce qui a eu lieu précédemment.



Mots clés et sites web

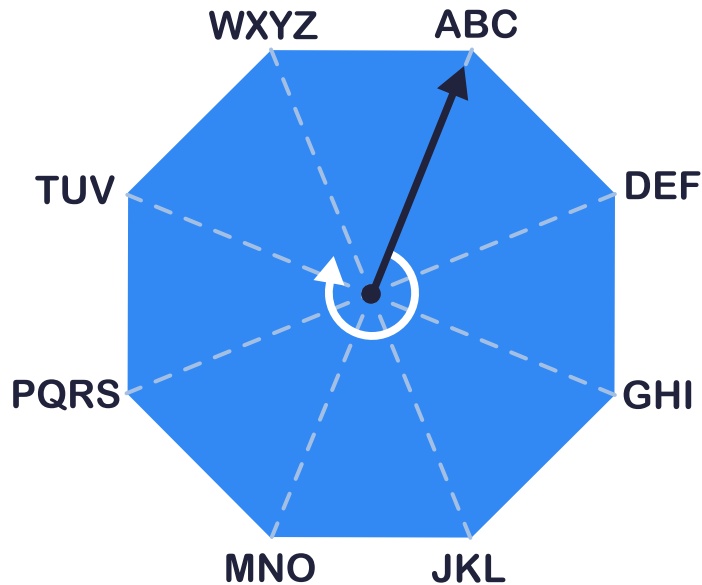
- Variable booléenne : <https://fr.wikipedia.org/wiki/Booléen>
- Transistor : <https://fr.wikipedia.org/wiki/Transistor>
- Électronique numérique : https://fr.wikipedia.org/wiki/Électronique_numérique
- Porte ET : https://fr.wikipedia.org/wiki/Fonction_ET
- Porte OU exclusif : https://fr.wikipedia.org/wiki/Fonction_NON-ET
- Table de vérité : https://fr.wikipedia.org/wiki/Table_de_vérité
- Porte OU : https://fr.wikipedia.org/wiki/Fonction_OU
- Porte NON : https://fr.wikipedia.org/wiki/Fonction_NON
- Porte logique : https://fr.wikipedia.org/wiki/Fonction_logique





4. Code 8

Des textes en clair peuvent être chiffrés grâce au disque suivant :



Au départ, l'aiguille pointe sur « ABC ».

Chaque lettre est chiffrée individuellement. Pour cela, deux chiffres sont déterminés :

- Le premier chiffre indique de combien de positions l'aiguille doit être tournée dans le sens des aiguilles d'une montre pour qu'elle pointe le bloc contenant la lettre à chiffrer.
- Le deuxième chiffre indique la position de la lettre à chiffrer dans le bloc pointé.

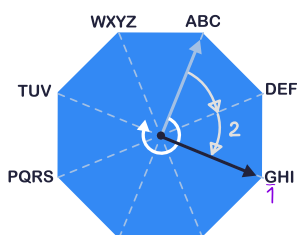
Le cryptogramme du mot « CHAT », par exemple, est 03 – 22 – 61 – 61.

Que signifie le cryptogramme 21-72-32-14 ?

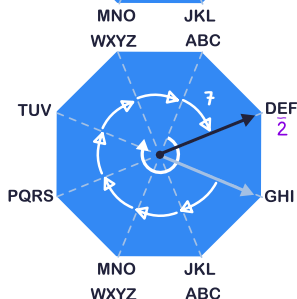
- A) GARS
- B) GENS
- C) GEMIR
- D) GELS
- E) GENE



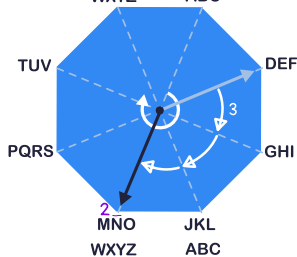
Solution



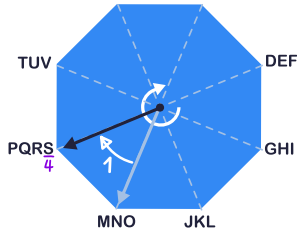
21 signifie que l'aiguille se déplace du bloc « ABC » au bloc « GHI » et que la première lettre, « G », est sélectionnée (le deuxième chiffre est 1).



72 signifie que l'aiguille se déplace du bloc « GHI » au bloc « DEF » et que la deuxième lettre, « E », est sélectionnée (le deuxième chiffre est 2).



32 signifie que l'aiguille se déplace du bloc « DEF » au bloc « MNO » et que la deuxième lettre, « N », est sélectionnée (le deuxième chiffre est 2).



14 signifie que l'aiguille se déplace du bloc « MNO » au bloc « PQRS » et que la quatrième lettre, « S », est sélectionnée (le deuxième chiffre est 4).

La réponse B) GENS est donc correcte.

Il existe aussi un moyen plus rapide de trouver la bonne réponse : la réponse C) GEMIR n'entre pas en question, car elle est composée de cinq lettres et le cryptogramme n'en contient que quatre. Le deuxième chiffre pour la quatrième lettre étant un 4, celle-ci ne peut être qu'un « S » ou un « Z ». Seules les réponses A), B) et D) remplissent cette condition. La lettre précédente doit venir du bloc situé à une position dans le sens inverse des aiguilles d'une montre du bloc « PQRS », donc du bloc « MNO ». Cela ne peut donc être que la réponse B) GENS.

C'est de l'informatique !

Depuis des milliers d'années, l'être humain cherche à cacher des informations afin que seul le destinataire ne puisse les déchiffrer. Ce qui commença avec des bouts de papier enroulés autour d'un bâton (scytale) se développa en la *cryptographie à clé publique* moderne (comme « GnuPG », qui utilise entre autres le chiffrement RSA) en passant par le chiffrement par transposition comme le « chiffre de César » et les *chiffrements polyalphabétiques* (comme le « chiffre de Vigenère »).



Le chiffrement de cet exercice est un chiffrement polyalphabétique, car une lettre n'est pas forcément toujours chiffrée la même chose : la lettre « A », par exemple, est chiffrée par 01 au début du texte en clair, mais par 31 si elle suit un « S ». Ces chiffrements peuvent tous être décryptés rapidement de nos jours à l'aide d'ordinateurs.

Dans ce cas, le déchiffrement est spécialement simple : il n'existe qu'une seule clé pour chiffrer un texte. Même si la position de départ de l'aiguille n'était pas toujours le bloc « ABC » mais un bloc au hasard, il n'y aurait que huit clés possibles. . . Même le chiffre de César, qui a plus de 2000 ans, est plus « sûr » que celui-ci ! On pourrait encore argumenter que le secret n'est pas la clé elle-même, mais le chiffrement. Mais le *principe de Kerckhoffs*, formulé par Auguste Kerckhoffs (1835 à 1903) en 1883 et encore valable aujourd'hui, montre que la sécurité d'un *cryptosystème* ne devrait pas se fonder sur la confidentialité d'une méthode de chiffrement, car celle-ci pourrait trop facilement être découvert par d'autres personnes.

Mots clés et sites web

- Chiffre de César : https://fr.wikipedia.org/wiki/Chiffrement_par_décalage
- Substitution polyalphabétique/chiffre de Vigenère :
https://fr.wikipedia.org/wiki/Chiffre_de_Vigenère
- Système de chiffrement : <https://fr.wikipedia.org/wiki/Chiffrement>
- Cryptographie à clé publique :
https://fr.wikipedia.org/wiki/Cryptographie_asymétrique
- GnuPG : https://fr.wikipedia.org/wiki/GNU_Privacy_Guard
- Chiffrement RSA : https://fr.wikipedia.org/wiki/Chiffrement_RSA
- Principe de Kerckhoffs : https://fr.wikipedia.org/wiki/Principe_de_Kerckhoffs
- Auguste Kerckhoffs : https://fr.wikipedia.org/wiki/Auguste_Kerckhoffs
- Cryptosystème : <https://fr.wikipedia.org/wiki/Cryptosystème>
- Cryptographie : <https://fr.wikipedia.org/wiki/Cryptographie>



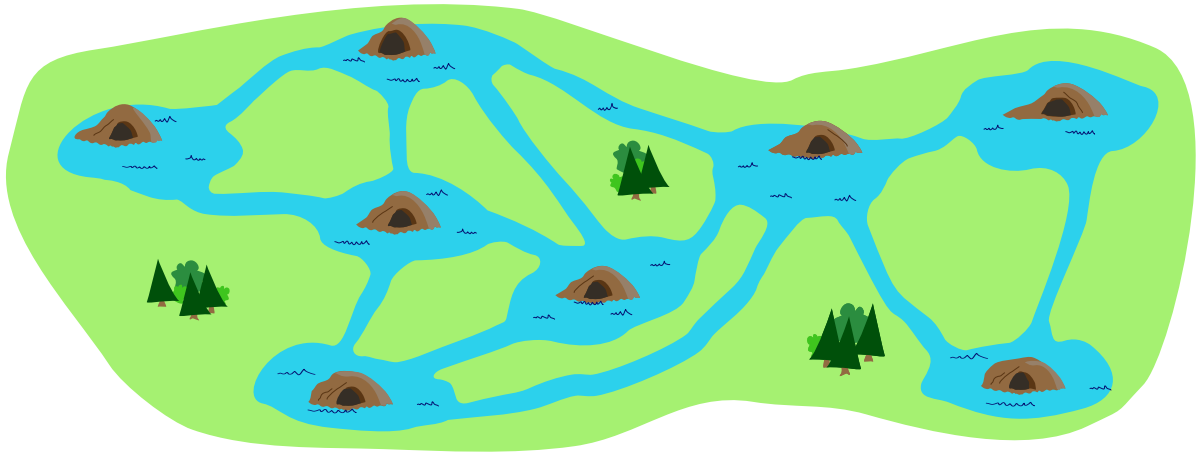


5. Les voisins de Lili

Tu vois sur la carte les huttes de huit castors. Deux castors sont voisins lorsqu'un canal relie leurs huttes.

- Lili, Simon, et Pierre ont quatre voisins chacun.
- Simon et Pierre sont les seuls voisins de Nina.

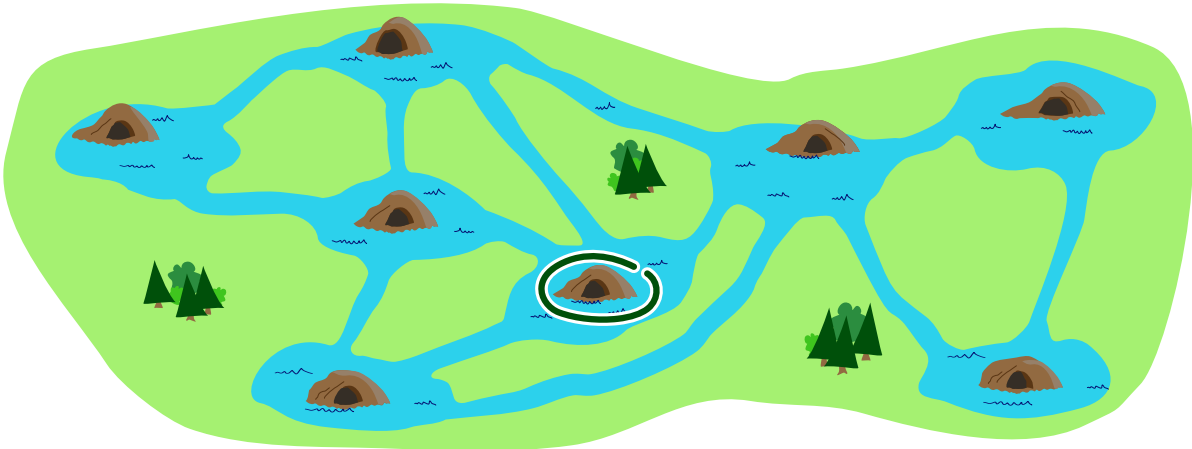
Dans quelle hutte Lili habite-t-elle ?



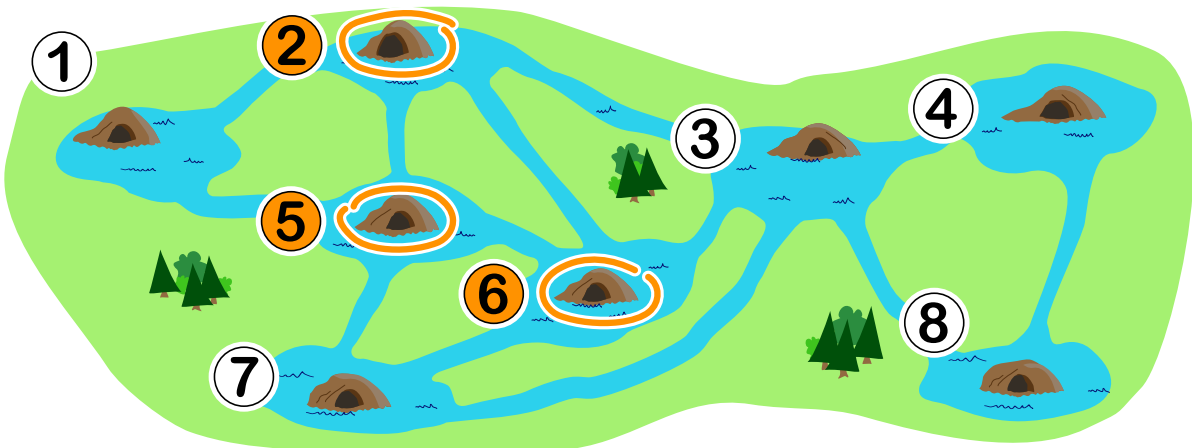


Solution

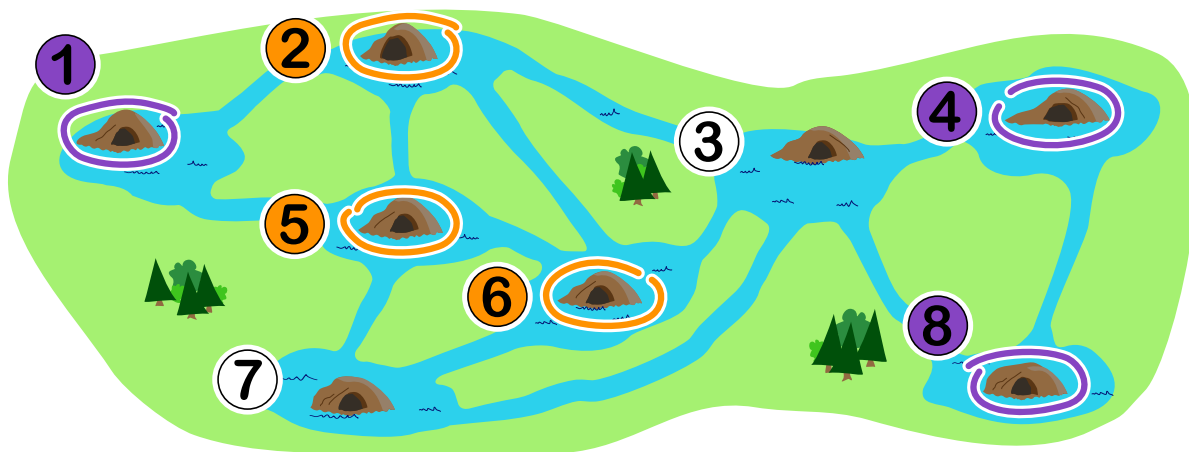
La bonne réponse est :



Pour résoudre le problème, il faut se concentrer sur les canaux entre les fortifications. Nous devons identifier les huttes dans lesquelles peuvent habiter Lili, Pierre et Simon. Comme ils ont chacun quatre voisins, exactement quatre canaux doivent rejoindre chacune de leurs huttes. Il y a trois huttes qui remplissent ces conditions : les huttes 2, 5 et 6.



Lili, Pierre et Simon habitent donc chacun dans une de ces trois huttes. Nous devons maintenant déterminer dans laquelle des trois vit Lili. Les deux autres informations concernent la hutte de Nina, et indiquent qu'exactly deux canaux rejoignent sa hutte. Nina vit donc dans une de ces trois huttes : 1, 4 et 8.



Comme nous savons que Simon et Pierre sont les deux voisins de Nina, nous pouvons en déduire que :

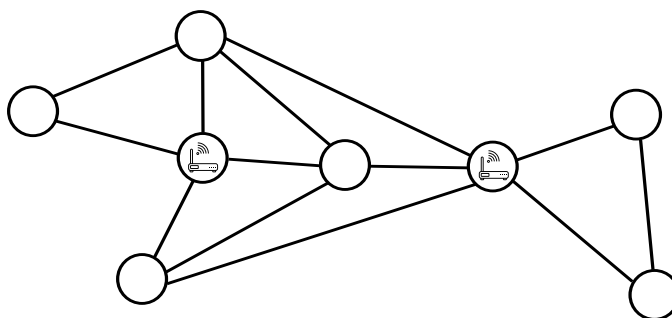
- Nina vit dans la hutte 1.
- Simon et Pierre vivent dans les huttes 5 et 7 (ou l'inverse).

Il ne reste donc qu'une seule hutte de laquelle partent quatre canaux et qui peut être celle de Lili. C'est la hutte 6.

C'est de l'informatique !

Dans cet exercice, deux huttes de castor sont reliées par un canal. L'ensemble des huttes et des canaux forme un réseau qui représente les *relations* entre toutes les huttes. Un tel réseau de relations entre des objets est appelé un *graphe* en informatique et en mathématiques. Un graphe peut être considéré comme un *ensemble* de *nœuds* qui sont reliés par des *arêtes*. Dans cet exercice, les nœuds sont les huttes et les arêtes sont les canaux.

La *théorie des graphes* peut être utilisée pour modéliser les relations entre des paires d'objets. Les graphes sont des modèles mathématiques de structures techniques ou naturelles, par exemple des structures sociales, des réseaux informatiques, des réseaux routiers, des circuits électriques, des réseaux d'approvisionnement ou des molécules. Les graphes peuvent être utiles pour décrire et résoudre des *problèmes de réseaux*, par exemple lorsqu'il s'agit de trouver un bon emplacement pour un routeur dans un bâtiment ou de s'assurer qu'un réseau Wi-Fi atteigne chaque pièce d'une maison.






Mots clés et sites web


- Graphe: [https://fr.wikipedia.org/wiki/Graphe_\(mathématiques_discrètes\)](https://fr.wikipedia.org/wiki/Graphe_(mathématiques_discrètes))
- Théorie des graphes: https://fr.wikipedia.org/wiki/Théorie_des_graphes
- Ensemble: <https://fr.wikipedia.org/wiki/Ensemble>
- Nœud: [https://fr.wikipedia.org/wiki/Sommet_\(théorie_des_graphes\)](https://fr.wikipedia.org/wiki/Sommet_(théorie_des_graphes))
- Arête: [https://fr.wikipedia.org/wiki/Arête_\(théorie_des_graphes\)](https://fr.wikipedia.org/wiki/Arête_(théorie_des_graphes))





























6. Poste robotisée

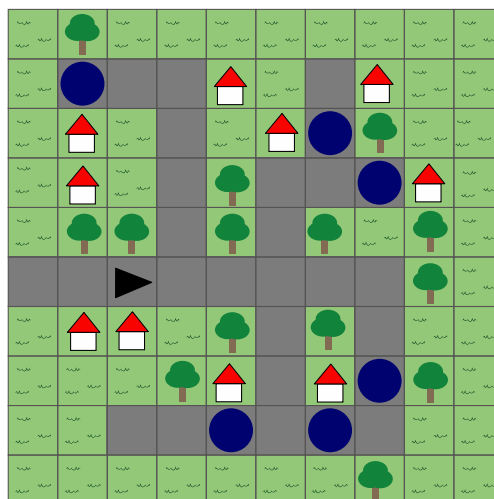
Tina le robot livre le courrier. Pour cela, elle utilise une carte du quartier, qui est divisée en cases. Tina se déplace le long de la rue  de case en case en allant vers la droite, la gauche ou l'avant (pas en diagonale).

Tina a trois capteurs pour naviguer. Dès qu'elle arrive sur une case (et avant qu'elle ne puisse se tourner), les capteurs reconnaissent ce qui se trouve sur les cases à la droite, à la gauche et devant Tina.

Le contenu des cases reconnu par les capteurs de Tina sur son chemin est enregistré dans la table ci-dessous. Tina a commencé sur la case  dans le sens de la flèche.

	gauche	devant	droite
			
			
			
			
			
			
			
			

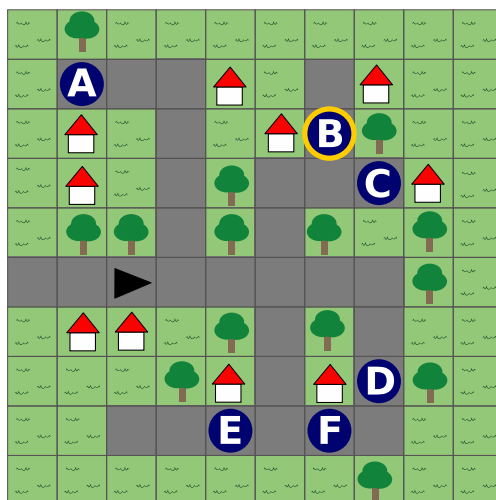
Sur quel point bleu  Tina se trouve-t-elle à la fin de son chemin ?





Solution

La bonne réponse est le point B.

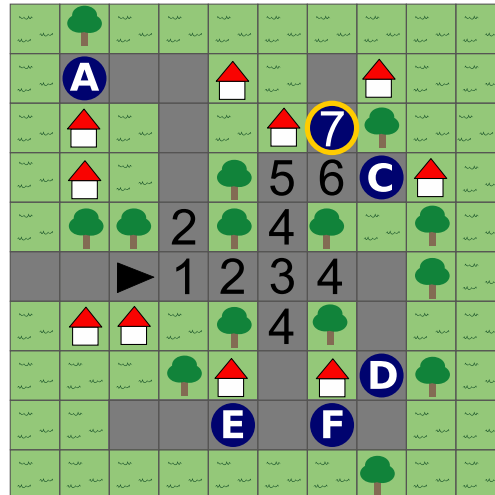


Étape	gauche	devant	droite
1			
2			
3			
4			
5			
6			
7			

Pour cet exercice, une méthode efficace consiste à se concentrer sur les six points d'arrivée et à vérifier si les données des capteurs pour l'étape 7 « » pourraient y correspondre. On peut ainsi exclure les réponses C, E et F. Les données des capteurs pour l'étape 6 « » permettent d'exclure les réponses A et D.

On pourrait également essayer de suivre le chemin enregistré dans la table. Les données ne correspondent qu'au chemin menant au point B.

On ne peut pas toujours décider tout de suite quel chemin Tina a suivi en suivant les informations des capteurs. À l'étape 4, Tina verrait des arbres à gauche et à droite quel que soit la direction dans laquelle elle est allée. Dans ce genre de situation, il faut prendre en compte les données des capteurs de l'étape suivante pour pouvoir déterminer le chemin exact de Tina.



C'est de l'informatique !

Dans cet exercice, nous rencontrons Tina le *robot*. Les robots sont des ordinateurs spécialement équipés qui utilisent des *capteurs* pour obtenir des informations sur leur environnement, traitent ces informations de manière automatisée (c'est-à-dire à l'aide d'un programme) et se basent sur leur environnement pour agir de manière autonome à l'aide d'*actionneurs*. Les capteurs de Tina recueillent le contenu des cases à sa gauche, à sa droite et devant elle. De manière concrète, on peut s'imaginer que les capteurs prennent des photos et que des données géométriques que l'ordinateur peut attribuer à une arbre, une maison ou une route sont extraites lors d'une analyse automatique. Les roues de Tina, ses actionneurs, peuvent donc être dirigés de manière à éviter les cases contenant des arbres ou une maison.

Les véhicules autonomes sont un exemple de tels robots. Ils sont équipés de nombreux capteurs qui ne mesurent pas seulement la vitesse et la position, mais aussi la distance au bord de la route, détectent les objets sur ou au bord de la route et encore beaucoup, beaucoup d'autres choses. Ces informations sont traitées à l'aide de programmes très complexes, qui peuvent par exemple reconnaître des enfants qui pourraient traverser la route et les distinguer de panneaux de circulation. Dans beaucoup de tels scénarios, l'*apprentissage automatique* est une technologie cruciale. Dans le cas des véhicules autonomes, les ordinateurs apprennent à l'aide de nombreux exemples donnés comment distinguer des enfants de panneaux de circulation. Les actionneurs sont par exemple les freins qui sont activés sans intervention humaine.

Mots clés et sites web

- Robot : <https://fr.wikipedia.org/wiki/Robot>
- Capteur : <https://fr.wikipedia.org/wiki/Capteur>
- Actionneur : <https://fr.wikipedia.org/wiki/Actionneur>
- Apprentissage automatique :
https://fr.wikipedia.org/wiki/Apprentissage_automatique





7. Séquences

Tu vois ici une séquence de chiffres appelée X. Les chiffres 5, 4, 3, 2, 1 occupent les positions 1 à 5 de la séquence X.

	1	2	3	4	5
X	5	3	2	4	1

Le chiffre occupant une certaine position d'une séquence est désigné en utilisant le nom de la séquence et le numéro de la position entre parenthèses. Par exemple, le chiffre en deuxième position de la séquence X est désigné par (X 2). Actuellement, (X 2) = 3.

Un chiffre désigné ainsi peut lui-même être une position, par exemple (X (X 2)) = (X 3) = 2.

Voici trois autres séquences : A, B et C.

A	3	2	4	1	5
B	5	4	1	3	2
C	2	5	4	3	1

Quel est le chiffre désigné par (A (B (C 3))) ?

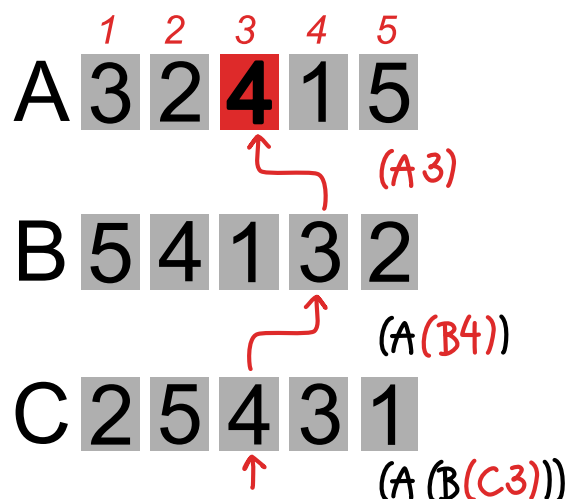
- A) 1
- B) 2
- C) 3
- D) 4
- E) 5



Solution

La bonne réponse est D) 4.

La désignation (A (B (C 3))) décrit le chiffre de la séquence A en position (B (C 3)); la position du chiffre est donc en position (C 3) dans la séquence B, position qui est elle-même en position 3 de la séquence C. C'est compliqué.



C'est plus facile de considérer la désignation « de l'intérieur vers l'extérieur », comme une expression arithmétique, et de procéder comme montré dans la description de l'exercice : (A (B (C 3))) = (A (B 4)) = (A 3) = 4.

C'est de l'informatique !

Il n'y a pas si longtemps que le travail des ordinateurs était qualifié de *traitement de données*, et pour cause : les ordinateurs traitent toute sortes de données comme des nombres, des textes, des images, des sons, et ainsi de suite. La plupart des données intéressantes enregistrées dans des ordinateurs sont complexes et structurées : les températures mesurées au cours de la journée par une station météorologique, par exemple, peuvent être représentées comme des paires de nombres composées chacune de l'heure de la mesure et de la température mesurée. Il y a dans ce cas une structure par paire et une structure par séquence.

Les données peuvent avoir beaucoup de structures différentes, et les informaticiennes et informaticiens ont développé beaucoup de *structures de données* différentes afin de pouvoir enregistrer des données de manière pratique, et, ce qui est tout aussi important, y accéder de manière efficace. Un *tableau* est une structure de données simple qui est au centre de cet exercice. Dans un tableau, un nombre fixe de données (par exemple des chiffres) peut être enregistré dans des positions adjacentes. Ces positions donnent aux données du tableau une structure séquentielle – ce qui serait donc adapté aux paires température/heure mentionnées plus haut. Les tableaux font partie des structures de données *statiques* à cause de leur taille fixe. Il existe aussi des structures de données *dynamiques* pour les séquences, comme les *listes*, dont la taille peut être changée selon les besoins.



Qu'elles soient statiques ou dynamiques, si une structure de données séquentielle contient des chiffres, ces chiffres peuvent indiquer des positions dans cette structure de données ou dans une autre structure de données. Ceci est souvent utilisé pour l'adressage indirect en informatique : l'adresse (ou position) dans une séquence n'est pas indiquée directement par chiffre, mais indirectement par une valeur dans une séquence, qui peut elle-même être adressée indirectement, et ainsi de suite.



Mots clés et sites web

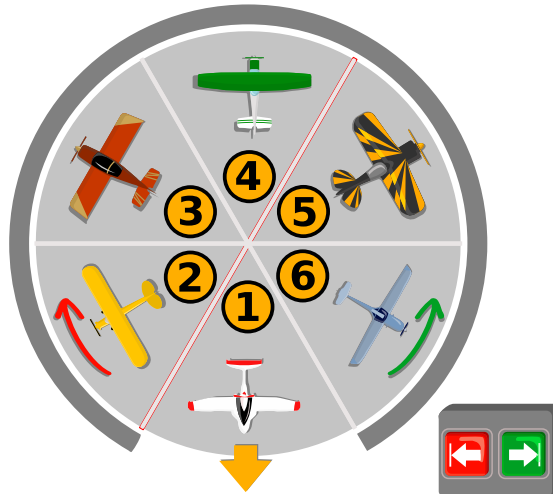
- Traitement de données : https://fr.wikipedia.org/wiki/Traitement_de_données
- Structure de données : https://fr.wikipedia.org/wiki/Structure_de_données
- Tableau : [https://fr.wikipedia.org/wiki/Tableau_\(structure_de_données\)](https://fr.wikipedia.org/wiki/Tableau_(structure_de_données))
- Adressage : https://fr.wikipedia.org/wiki/Mode_d'adressage






8. Hangar tournant

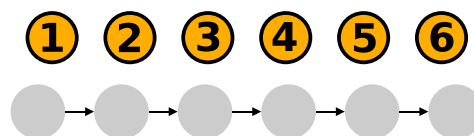
Six avions se parquent à l'aéroport de Castorville. Ils sont parqués dans un hangar, sur une plaque tournante qui a six places de parc. À l'extérieur se trouvent deux boutons fléchés  . En actionnant un bouton, on peut faire tourner la plaque d'exactly une place de parc vers la gauche ou la droite.



Lorsque les pilotes viennent chercher leurs avions le matin, la place de parc 1 est toujours en face de la porte du hangar et l'avion parqué dessus peut tout de suite sortir. Dans le meilleur des cas, il ne faut appuyer que cinq fois sur les boutons fléchés pour permettre à tous les autres avions de sortir. Par exemple, si les pilotes veulent faire sortir les avions dans l'ordre 1, 6, 5, 4, 3, 2, il suffit d'appuyer cinq fois sur la touche .

Mais quel est le pire des cas ? Quelle séquence de sortie les avions doivent-ils avoir pour qu'il faille appuyer le plus souvent sur les boutons ?

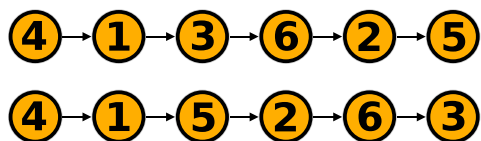
Donne un exemple d'une telle séquence.





Solution

Il y a deux bonnes réponses possibles :



Pour arriver à une solution, il faut toujours choisir l'avion parké sur la place la plus éloignée de la porte du hangar.

« 4 » signifie que l'avion parké sur la place 4 sort du hangar après que l'on a appuyé trois fois sur le bouton fléché .

4 1 3 6 2 5 :



4 1 5 2 6 3 :



16 étapes sont nécessaires dans les deux cas.

Il ne peut pas y avoir plus de 16 étapes, car il n'est possible d'avoir deux séries de trois étapes qu'au début ; ensuite, il peut y avoir tout au plus des séries de deux et de trois étapes en alternance.

C'est de l'informatique !

Le hangar tournant a l'avantage de permettre de parker des avions en économisant de la place, mais cela prend en général plus de temps d'aller chercher les avions.

L'*efficacité* et la *complexité* d'un processus forment un thème central de l'informatique, parce que c'est un critère d'évaluation des *algorithmes* important. Souvent, l'efficacité concerne le *temps d'exécution* de l'algorithme, mais ce n'est pas toujours le cas. La définition générale de la complexité d'un algorithme concerne toutes les ressources utilisées, par exemple l'*espace de stockage* nécessaire.

Comme dans notre exemple du hangar, économiser une ressource cause un besoin plus grand d'autres ressources. Quelle ressource a la plus grande importance dépend de la relation concrète entre les ressources et de leurs différentes disponibilités.

Par exemple, les algorithmes *Timsort* et du *tri à bulles* sont deux algorithmes permettant de trier une liste d'objets. Le tri à bulles trie les objets en un temps proportionnel à leur nombre au carré ($\mathcal{O}(n^2)$), mais n'a besoin que de peu d'espace supplémentaire et cet espace n'augmente pas avec le nombre d'objets.

Timsort trie beaucoup plus rapidement que le tri à bulles ($\mathcal{O}(n \log n)$), mais l'espace dont il a besoin est proportionnel à la taille de la liste. Si une utilisation concrète de tri demande de trier de longues



listes rapidement, Timsort est la meilleur méthode ; par contre, s'il est plus important de minimiser l'espace de stockage nécessaire, le tri à bulles est une meilleure méthode.

Mots clés et sites web

- Complexité (efficacité) :
https://fr.wikipedia.org/wiki/Analyse_de_la_complexité_des_algorithmes
- Algorithme : <https://de.wikipedia.org/wiki/Algorithmus>
- Tri à bulles : https://fr.wikipedia.org/wiki/Tri_à_bulles
- Timsort : <https://fr.wikipedia.org/wiki/Timsort>
- Notation de Landau : https://fr.wikipedia.org/wiki/Comparaison_asymptotique#La_-famille_de_notations_de_Landau





9. Soirée ciné

Quelques amis veulent regarder un film ensemble. Ils ont le choix entre sept films. Pour prendre une décision, chaque personne évalue chaque film : bon 😊, moyen 😐 ou mauvais 😞.

Tu vois le résultat ci-dessous. Malheureusement, il n’y a pas de favori pour la soirée.

Un film est un « favori » quand il reçoit de chaque personne la meilleure évaluation parmi les films. Par exemple, le film 1 n’est pas un favori parce que Niklaus a donné sa meilleure évaluation à un autre film, le film 4.

Ada aimerait devoir convaincre le moins de personnes possible de changer leurs évaluations pour avoir un favori malgré tout.

Aide Ada en changeant le moins d’évaluations possible pour qu’il y ait un favori.

	1	2	3	4	5	6	7
Ada	😊	😊	😊	😊	😊	😊	😊
Nancy	😐	😊	😊	😐	😐	😊	😊
Niklaus	😞	😞	😞	😐	😞	😞	😞
Grace	😞	😐	😐	😐	😞	😐	😞
Edsger	😊	😐	😞	😞	😐	😊	😊
Rozsa	😐	😞	😐	😞	😊	😐	😐



Solution

Il n’y a pas de favori au départ. Pour chaque film, nous trouvons des amis qui ont mieux évalué d’autres films.

Film Amis qui ont mieux évalué d’autres films

1	4: Nancy, Niklaus, Grace et Rozsa
2	3: Niklaus, Edsger et Rozsa
3	3: Niklaus, Edsger et Rozsa
4	3: Nancy, Edsger et Rozsa
5	3: Nancy, Grace et Edsger
6	2: Niklaus et Rozsa
7	3: Niklaus, Grace et Rozsa

Il n’y a que deux amis qui ont mieux évalué d’autres films que le film 6. Tous les autres films ont été moins bien évalués que d’autres par plus de deux des amis. Il ne suffit donc pas qu’un seul ami change son évaluation pour qu’il y ait un favori. Ada doit convaincre Niklaus et Rozsa de faire du film 6 leur film préféré, et elle obtient alors un favori après deux changements.

Quelles évaluations Niklaus et Rosza peuvent-ils changer afin que le film 6 ait leur meilleure évaluation ? Chacun a trois possibilités :

- Niklaus peut changer son évaluation du film 6 en 😞 ou 😊, ou alors changer son évaluation du film 4 en 😞. Dans les trois cas, le film 6 obtient sa meilleure évaluation.
- Rozsa peut changer son évaluation du film 6 en 😊, ou alors changer son évaluation du film 5 en 😞 ou 😞. Dans les trois cas, le film 6 obtient sa meilleure évaluation.

Chacune des trois possibilités de Niklaus et Rosza peuvent être combinées de n’importe quelle manière. Il existe en tout $3 \times 3 = 9$ possibilités de ne changer que deux évaluations afin d’obtenir un favori.

C’est de l’informatique !

Comment procédons-nous pour résoudre cet exercice ? Une idée est de vérifier pour chaque film et chaque personne s’il y a un autre film que cette personne a mieux évalué. Dans notre cas, on obtient ainsi la table plus haut. Cette table nous aide à trouver quelles personnes doivent changer leurs évaluations pour que nous obtenions un favori avec le moins de changements possible.

Ada peut utiliser cet algorithme pour résoudre son problème. Mais cet algorithme est-il *efficace* ? Ada pourrait-elle résoudre le problème encore plus vite ?



Dans ce qui suit, nous dénotons le nombre de film par M et le nombre d'amis par F . Ada doit considérer les $M \times F$ entrées individuellement et doit prendre en compte les $M - 1$ autres évaluations de la même personne pour chaque entrée. Ada doit donc considérer $M \times (M - 1) \times F$ évaluations en tout.

Pour déterminer si une évaluation est problématique, Ada ne doit la comparer qu'à la meilleure évaluation de la même personne. Si cette personne a mieux évalué un autre film, le film considéré par Ada ne peut pas être un favori.

Autrement dit, Si Ada commence par déterminer la meilleure évaluation de chaque personne (en considérant toutes les $M \times F$ évaluations), elle peut déterminer pour chacune des $M \times F$ évaluations si elle est moins bonne que la meilleure évaluation de la même personne.

Cet algorithme alternatif qui commence par rechercher la meilleure évaluation permet à Ada de considérer $2 \times M \times F$ évaluations en tout. Pour $M = 7$ et $F = 6$, il s'agit de 84 comparaisons, alors que le premier algorithme avait besoin de 252 comparaisons. Le deuxième algorithme résout aussi correctement le problème d'Ada, mais beaucoup plus efficacement que le premier.

Une des tâches les plus importantes en informatique est de résoudre des problèmes non seulement de manière correcte, mais aussi le plus efficacement possible. On arrive plus rapidement à des solutions avec des ordinateurs plus rapides, mais si aucun algorithme efficace n'est connu, même les ordinateurs les plus rapides peuvent atteindre leurs limites.

Mots clés et sites web

- Algorithme : <https://fr.wikipedia.org/wiki/Algorithme>
- Complexité (efficacité) :
https://fr.wikipedia.org/wiki/Analyse_de_la_complexité_des_algorithmes



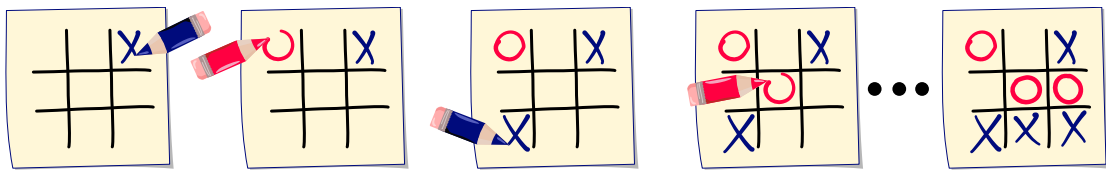


10. Morpion

Le morpion est un jeu pour deux personnes.

Les deux joueurs remplissent une grille de 3 × 3 cases à tour de rôle : un joueur dessine des X et l'autre des O dans les cases. Le premier joueur qui place ses trois symboles dans trois cases alignées dans une colonne, une ligne ou une diagonale a gagné et le jeu est terminé. Si personne n'a gagné et que toutes les cases sont remplies, c'est un match nul.

Tu vois ici les étapes possibles d'un jeu : les quatre premiers tours et le dernier tour. Le joueurs avec les X a gagné.



Nous appelons la dernière étape du jeu « état final ». Les règles du jeu déterminent exactement de quelle manière les cases peuvent être remplies de X et de O et quand le jeu se termine.

Une seule des images suivantes montre l'état final d'un jeu de morpion. Laquelle ?

- A)

X	O	X
O	X	O
O	O	X
- B)

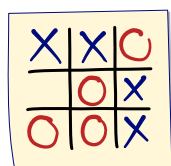
X	O	X
O	X	
O	X	X
- C)

X	X	O
	O	X
O	O	X
- D)

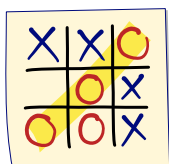
X	O	X
O	X	O
O	X	



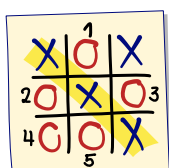
Solution



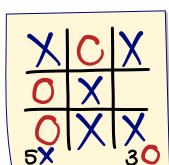
La réponse C est juste :



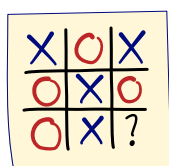
La réponse C est correcte, car un joueur a gagné (trois en diagonale) et le jeu s'est terminé.



La réponse A n'est pas correcte. Le joueur a gagné, mais le joueur a continué de remplir des cases. Comme le gagnant remplit toujours la dernière case, il ne peut jamais dessiner moins de symboles que le perdant.



La réponse B n'est pas correcte, car cinq cases ont des alors seulement trois ont des . Ce n'est pas possible, car le nombre de et de ne peuvent avoir qu'un point de différence.



Le réponse D n'est pas correcte, car elle ne montre pas un état final. Il n'y a pas de gagnant et les cases ne sont pas toutes remplies.

C'est de l'informatique !

Pour résoudre l'exercice, nous avons vérifié pour chacune des images si elle correspondait à un état final valide. On peut déduire de nouvelles règles définissant un état final à partir des règles du morpion, par exemple celles-ci :

1. La différence entre le nombre de et de doit être égale à 0, -1 ou 1.
2. Si aucun joueur n'a gagné, toutes les cases doivent être remplies.
3. Le perdant peut avoir rempli au maximum le même nombre de cases que le gagnant.
4. Il ne peut pas y avoir plus d'une seule suite de trois symboles alignés.

Ces nouvelles règles ne sont pas des règles de jeu, mais servent à vérifier si une grille remplie représente un état final. Si une image entre en conflit avec une de ces règles, elle ne peut pas représenter d'état final.

Les règles sont très importantes en informatique. Un *interprète* qui exécute un programme vérifie si le texte du programme correspond aux règles de syntaxe du langage de programmation.

En programmation, on utilise des règles appelées assertions pour pouvoir vérifier qu'un programme est correct pendant son exécution.



Mots clés et sites web

- Morpion : <https://fr.wikipedia.org/wiki/Tic-tac-toe>
- Interprète : [https://fr.wikipedia.org/wiki/Interprète_\(informatique\)](https://fr.wikipedia.org/wiki/Interprète_(informatique))
- Langage de programmation : https://fr.wikipedia.org/wiki/Langage_de_programmation

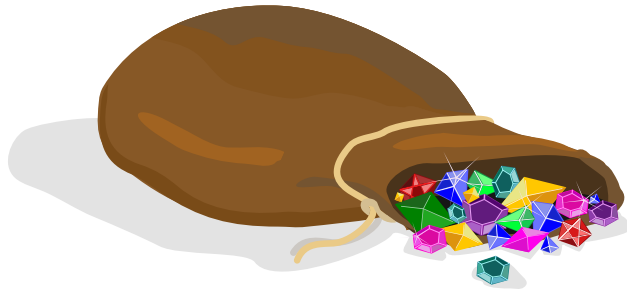




11. Pierres précieuses

Peter a plusieurs pierres précieuses. Elles ont toutes une valeur différente.

Sarah connaît les pierres précieuses de Peter, mais pas leur valeur. Elle aimerait savoir quelle pierre a le plus de valeur.



Pour cela, elle fait la chose suivante trois fois de suite :

- Elle choisit quatre pierres parmi la collection de Peter et lui demande laquelle des quatre a le plus de valeur.

Elle choisit à chaque étape quatre pierres comme elle veut, et Peter lui répond toujours honnêtement.

À la fin, Sarah sait quelle pierre précieuse a le plus de valeur.

Combien de pierres précieuses Peter peut-il avoir au maximum ?

- A) 8 pierres précieuses
- B) 10 pierres précieuses
- C) 11 pierres précieuses
- D) 12 pierres précieuses



Solution

La réponse B) : 10 pierres précieuses est correcte.

Si Peter a 10 pierres précieuses, Sarah peut choisir en tout huit pierres différentes lors de ses deux premières questions. Les deux pierres « gagnantes » de chaque question (c'est à dire le deux qui ont la plus grande valeur parmi les deux groupes de quatre) peuvent chacune être la « grande gagnante », c'est à dire la pierre ayant la plus grande valeur de toutes. Les six autres pierres comparées sont éliminées. Pour sa dernière question, Sarah compare les deux gagnantes et les deux pierres qu'elle n'a pas encore choisies jusque-là. La gagnante de la dernière question est aussi la grande gagnante.

Sarah peut donc (entre autres) procéder de cette façon pour trouver la pierre ayant la plus grande valeur parmi 10 pierres. Si Peter a 11 pierres, cela ne fonctionne pas :

Si Sarah procède comme décrit plus haut et compare huit pierres différentes lors de ses deux premières questions, il lui reste ensuite les deux gagnantes et trois autres pierres à comparer pour trouver la grande gagnante, donc trop de pierres pour sa troisième question. Si Sarah décide de comparer la gagnante de sa première question avec trois nouvelles pierres lors de sa deuxième question, cela lui apprend quelle pierre a le plus de valeur parmi les sept pierres comparées. Elle doit ensuite encore comparer cette pierre avec les quatre autres pierres qu'elle n'a pas encore choisies, ce qui fait une pierre de trop pour sa troisième question.

Si Sarah choisissait six pierres ou moins parmi 11 pour ses deux premières questions, ou si Peter avait plus de 12 pierres, elle aurait besoin d'encore plus de questions pour déterminer laquelle a le plus de valeur.

C'est de l'informatique !

Cet exercice présente un *algorithme* qui est limité par des conditions. Ici, ces conditions sont que Sarah ne peut poser que trois questions et que chaque question ne peut contenir que quatre éléments.

Malgré cette limite, l'algorithme fonctionne bien pour des collections de moins de 11 éléments, mais échoue pour de plus grandes collections.

Il existe plusieurs raisons d'imposer des contraintes à un algorithme. On pourrait par exemple imposer qu'une opération soit effectuée en un temps limite, ce qui est nécessaire pour les systèmes d'exploitation temps réel. De plus, certains procédés peuvent engendrer des coûts externes ou endommager des composants.

Ce n'est pas un problème que l'algorithme échoue à partir d'un certain seuil du moment que l'on contrôle que ce seuil ne soit jamais dépassé. La stratégie limitée de cet exercice ne doit par exemple jamais être utilisée pour des collections de plus de 10 éléments.

Mots clés et sites web

- Algorithme : <https://fr.wikipedia.org/wiki/Algorithme>



- Complexité en temps : https://fr.wikipedia.org/wiki/Complexité_en_temps

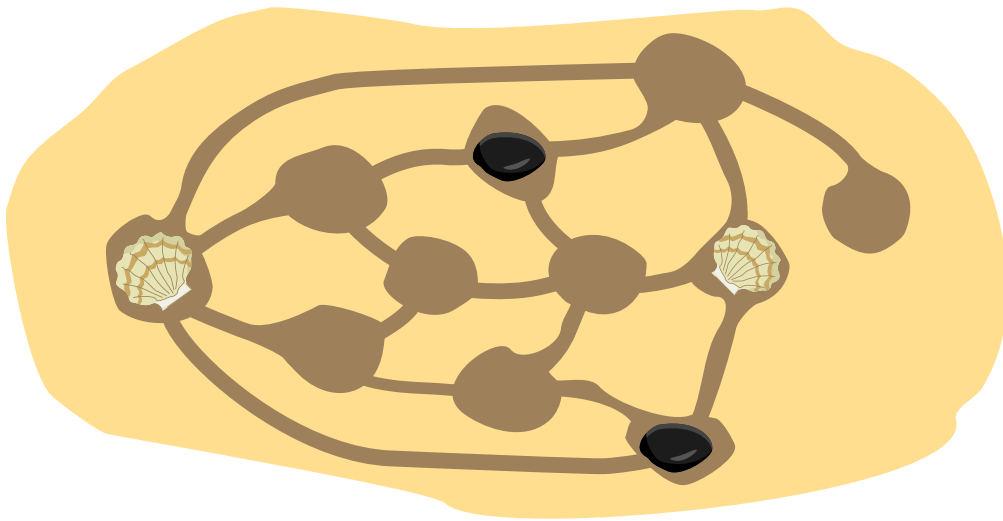




12. Galets et coquillages

Anne et Bob jouent à la plage. Ils ont creusé plusieurs trous et ont relié certains d'entre eux par des sillons tracés dans le sable. Les pièces d'Anne sont les coquillages 🐚 et les pièces de Bob les galets 🪨.

Ils placent tour à tour une de leurs pièces dans un trou inoccupé. Le premier qui met deux de ses pièces dans deux trous directement reliés a perdu. Tu vois où le jeu en est après quelques tours sur l'image ci-dessous.

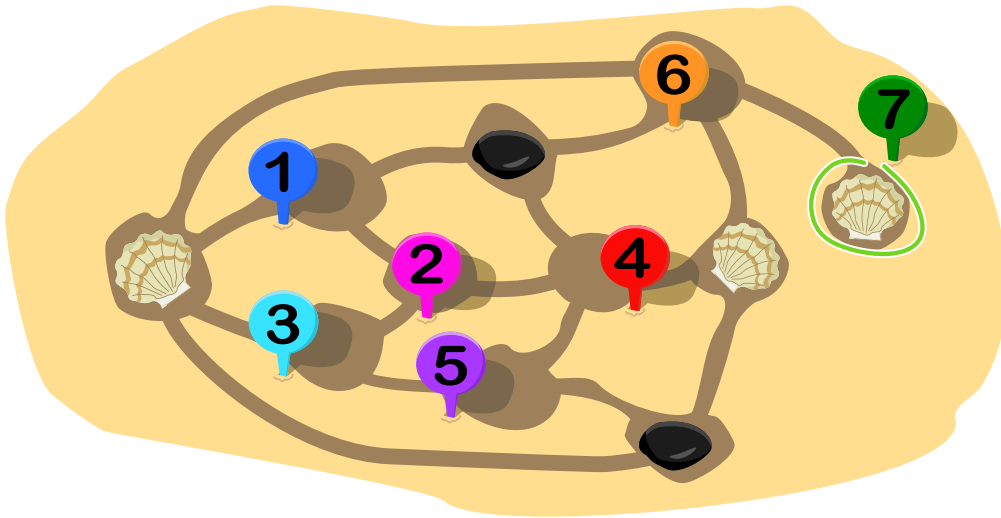


C'est le tour d'Anne. Dans lequel des trous inoccupés doit-elle mettre son prochain coquillage pour s'assurer la victoire ?

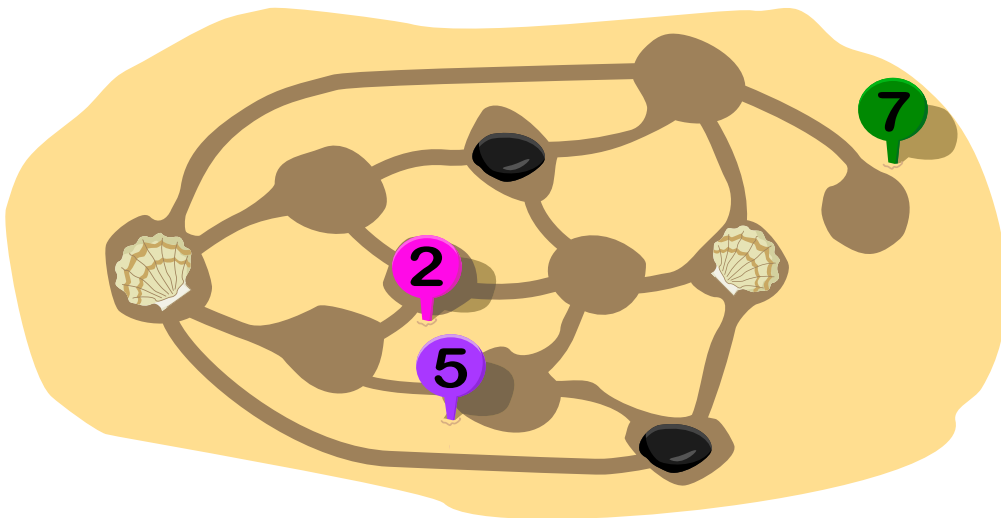


Solution

Le bonne réponse est le trou 7.



C'est le tour d'Anne. Les trous 1, 3, 4 et 6 sont hors de question pour elle ; il reste donc les trous 2, 5 et 7.



Elle remarque que les trous 1, 4 et 5 sont hors de question pour Bob. Il lui reste les trous 2, 3 et 7.

Si Anne joue le trou 7, Bob peut jouer les trous 2 ou 3 ; dans les deux cas, Anne peut encore jouer le trou 5 au prochain tour et gagner.

Si Anne jouait le trou 2, Bob pourrait jouer le trou 7 au prochain tour. Anne devrait alors jouer le trou 5, Bob le trou 3 et Bob aurait gagné.

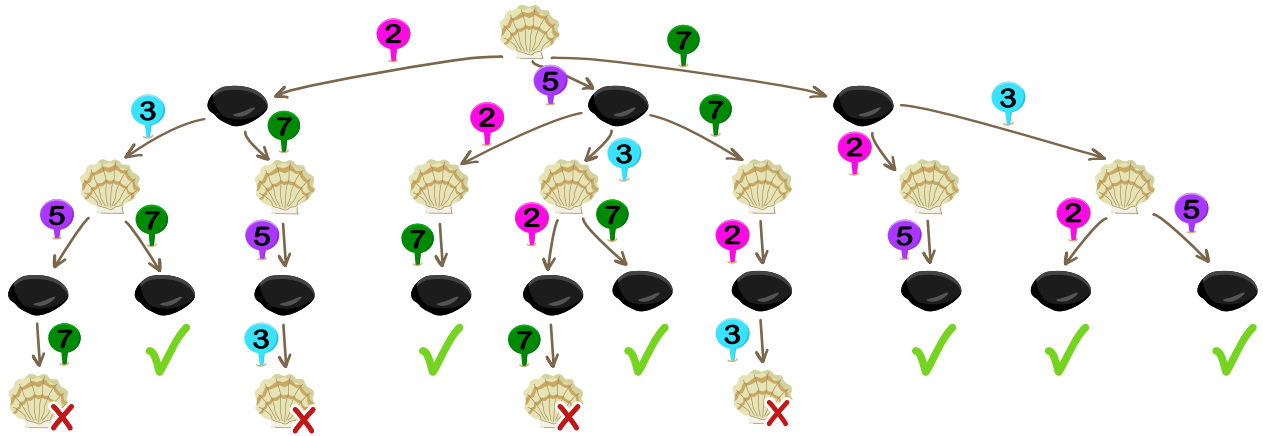
Si Anne jouait le trou 5, Bob pourrait jouer le trou 7. Anne devrait jouer le trou 2 au prochain tour, Bob le 3 et il aurait à nouveau gagné.

Bob ne pourrait par ailleurs pas gagner non plus si c'était son tour de jouer et pas celui d'Anne !



C'est de l'informatique !

Pour représenter de manière systématique les actions possibles d'Anne et de Bob dans le jeu, un arbre de jeu est utile :



On peut lire sur cet arbre de jeu quelle action assure la victoire à Anne : en partant de la branche de droite qui commence par Anne jouant le trou 7, on ne peut arriver qu'à des situations où elle gagne. En *théorie des jeux*, un domaine spécifique des mathématiques, on considère l'issue de jeux dans lesquels deux joueurs ou plus interagissent ; en informatique, on considère les algorithmes permettant d'analyser de tels arbres de jeu. Les ordinateurs avec assez de puissance de calcul peuvent déjà jouer et gagner contre des êtres humains à des jeux comme les échecs. La théorie des jeux offre également des modèles de systèmes complexes dans lesquels des «joueurs» interagissent qui sont utiles en psychologie, en économie et dans d'autres domaines. Par exemple, des modèles pour analyser le comportement d'acheteurs lors de changements de prix ou le choix de la route dans la circulation routière en sont dérivés.

Le jeu d'Anne et Bob est une version de « Col ». Col est un jeu pour deux joueurs qui a été introduit par Colin Vout et est mentionné dans le célèbre livre « On Number and Games » du mathématicien John Horton Conway.

Mots clés et sites web

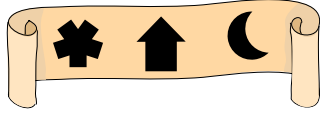
- Théorie des jeux : https://fr.wikipedia.org/wiki/Théorie_des_jeux
- John Horton Conway : https://fr.wikipedia.org/wiki/John_Horton_Conway
- On Numbers and Games : https://fr.wikipedia.org/wiki/On_Numbers_and_Games



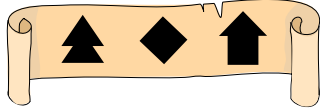


13. Coffre au trésor

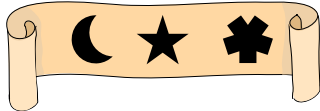
Maria trouve un coffre mystérieux. Malheureusement, le coffre est verrouillé. Pour l'ouvrir, Maria doit découvrir la « clé » : la bonne combinaison composée de trois symboles. Heuseusement, elle trouve les indications suivantes à côté du coffre :



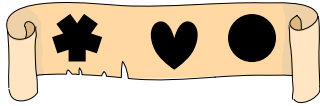
1. L'un des symboles fait partie de la clé et se trouve à la bonne position.



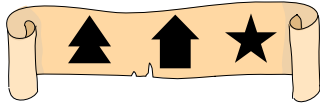
2. Aucun des symboles ne fait partie de la clé.



3. Deux des symboles font partie de la clé, mais ils se trouvent les deux à de fausses positions.



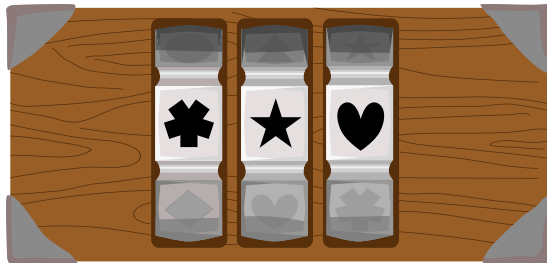
4. L'un des symboles fait partie de la clé mais se trouve à la fausse position.



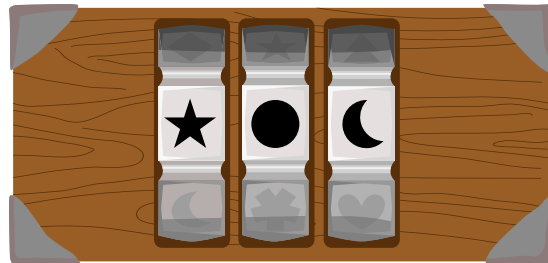
5. L'un des symboles fait partie de la clé mais se trouve à la fausse position.

L'une des combinaisons suivantes est la clé du coffre. Laquelle ?

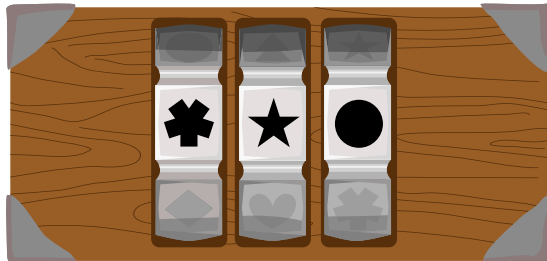
A)



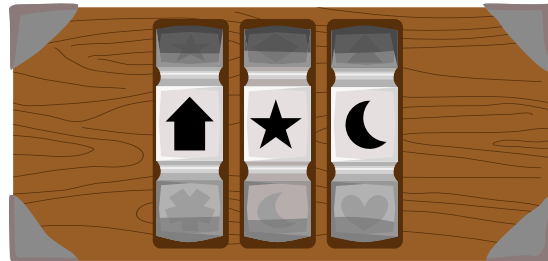
B)



C)



D)





Solution

La bonne réponse est B).

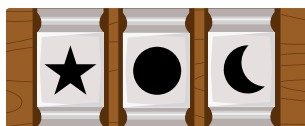
Nous commençons par déterminer quels symboles peuvent faire partie la clé. En suivant l'indication 2), nous pouvons éliminer les symboles qui ne peuvent pas faire partie de la clé : le sapin ▲, le diamant ◆ et la maison 🏠. L'indication 5) spécifie qu'un des symboles fait partie de la clé, mais à une position différente. Comme le sapin ▲ et la maison 🏠 ne peuvent pas faire partie de la clé, nous savons que l'étoile ★ doit en faire partie, mais à une autre position que dans l'indication. L'indication 3) exclut que l'étoile ne soit au milieu, on connaît donc sa position finale :



Comme il n'y a qu'une réponse qui commence par l'étoile parmi les propositions, nous avons déjà trouvé la clé. Pour nous convaincre, nous continuons à chercher les deux symboles suivants. L'indication 1) spécifie qu'un des symboles se trouve dans la clé à la même position. La maison 🏠 et la première position peuvent être exclues. Nous savons donc que la lune 🌙 se trouve à la bonne position :



L'indication 4) spécifie qu'un symbole fait partie de la clé, mais à une autre position. Nous pouvons éliminer le symbole ♣ d'après l'indication 1). De plus, seule la position du milieu est encore libre sur la clé ; le cœur ♥ ne peut donc pas faire partie de la clé. Seul le cercle peut donc occuper la position du milieu.



Il y a d'autres moyens d'arriver au bon résultat, mais ils mènent tous à la même réponse.

C'est de l'informatique !

Cet exercice peut être résolu logiquement, par exemple en procédant par exclusion. Dans notre cas, nous avons commencé avec l'indication 2) et exclu trois symboles, ce qui nous a rapidement mené à la bonne réponse. Le fait de commencer par l'indication 2) peut être vu comme une stratégie mentale, une règle ou un raccourci qui nous a aidé à prendre rapidement une décision basée sur des connaissances limitée. En informatique, de telles règles sont appelées des *heuristiques*. Elles peuvent être programmées et automatisées.



Chaque jour, nous prenons plusieurs décisions sur la base d'indications ou devons prendre en compte plusieurs conditions d'un problème pour pouvoir le résoudre. Dans cet exercice, nous avons suivi les indications et avons résolu le problème étape par étape pour pouvoir ouvrir le coffre.

Comment est-ce qu'un ordinateur résoudrait ce problème ? Il y a en tout 336 possibilités d'arranger ces huit symboles sur les trois positions. Un ordinateur les essaierait toutes. En informatique, on appelle cela une *recherche exhaustive*. La recherche exhaustive, aussi appelée *recherche par force brute* est une méthode de résolution de problèmes qui consiste à parcourir l'ensemble des solutions possible. Cette méthode peut nous sembler peu efficace parce que nous aurions besoin de beaucoup de temps pour essayer toutes les possibilités (et oublierions celles que nous avons déjà essayé), mais un ordinateur peut résoudre de tels problèmes très rapidement et efficacement. Les symboles de l'exemple pourraient aussi constituer un mot de passe. C'est pourquoi les mots de passe devraient toujours contenir autant de symboles différents que possible afin qu'une recherche exhaustive ne puisse pas le trouver en un temps réaliste.

Le fait de commencer par l'indication 2) et de minimiser le nombre de solutions possibles s'appelle en informatique *retour sur trace* (ou *backtracking* en anglais). À chaque sommet d'un arbre, les possibilités qui ne peuvent pas faire partie de la clé sont éliminées. De cette manière, le nombre de possibilités est réduit à chaque étage de l'arbre.

Mots clés et sites web

- Heuristique : [https://fr.wikipedia.org/wiki/Heuristique_\(mathématiques\)#Heuristique_au_sens_de_l'algorithmique](https://fr.wikipedia.org/wiki/Heuristique_(mathématiques)#Heuristique_au_sens_de_l'algorithmique)
- Recherche exhaustive : https://fr.wikipedia.org/wiki/Recherche_exhaustive
- Complexité (efficacité) : https://fr.wikipedia.org/wiki/Analyse_de_la_complexité_des_algorithmes
- Retour sur trace : https://fr.wikipedia.org/wiki/Retour_sur_trace
- Sommet : [https://fr.wikipedia.org/wiki/Sommet_\(théorie_des_graphes\)](https://fr.wikipedia.org/wiki/Sommet_(théorie_des_graphes))
- Arbre : [https://fr.wikipedia.org/wiki/Arbre_\(théorie_des_graphes\)](https://fr.wikipedia.org/wiki/Arbre_(théorie_des_graphes))



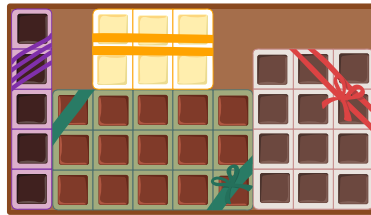


14. Empaquetage

La fabrique de chocolat « Castocolat » envoie quatre boîtes de pralinés à chacun de ses clients dans le cadre d'une campagne de publicité.

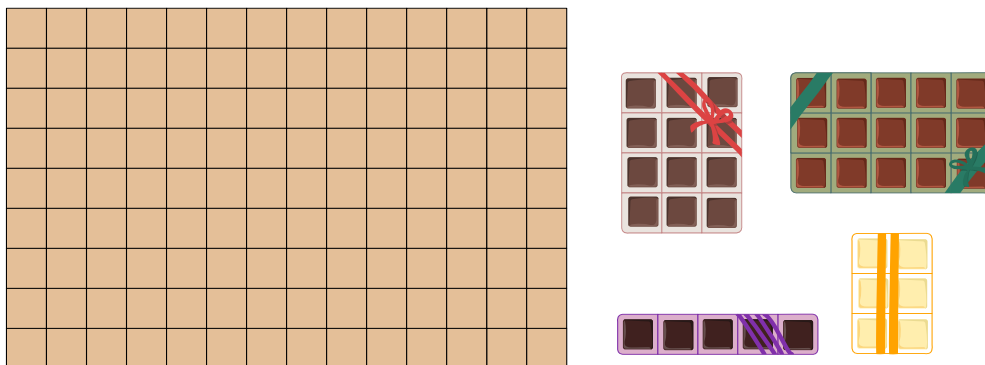
Pour économiser du matériel et des frais de port, Linus doit emballer côte à côte les quatre boîtes de pralinés de tailles différentes dans le plus petit carton possible. Les boîtes ne doivent pas être mises les unes sur les autres, car cela écraserait les pralinés.

Linus a rangé les boîtes de pralinés dans un carton de taille $5 \times 9 = 45$ pralinés comme ceci :



Lina affirme qu'elle peut prendre un plus petit carton en arrangeant les boîtes différemment.

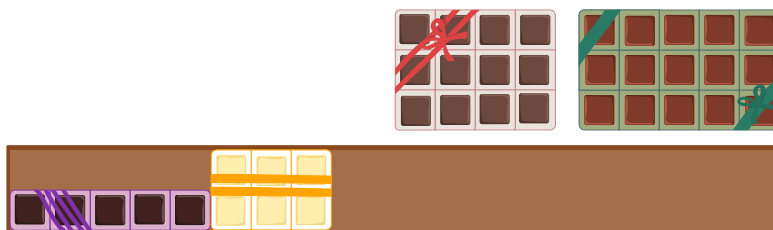
Rangé les boîtes de manière à avoir besoin du plus petit carton possible.





Solution

Linus doit mettre en tout $12 + 15 + 6 + 5 = 38$ pralinés dans un carton. Un carton dans lequel passent 38 pralinés sans espace vide doit avoir soit les dimensions 1×38 , soit 2×19 (2 et 19 sont les seuls diviseurs de 38). Les deux boîtes de pralinés de dimensions 3×4 et 3×5 n'iraient pas dans un tel carton.



Si Linus choisit un carton pour 39 pralinés (avec la place pour exactement un praliné de plus), celui-ci doit avoir la taille 1×39 ou 3×13 . Les boîtes de 3×5 , 3×4 et 3×2 iraient dans un carton de 3×13 , mais il n'y aurait plus la place pour la boîte de 1×5 .



Un carton pour 20 pralinés peut avoir les dimensions 1×40 , 2×20 , 4×10 ou 5×8 . On ne peut pas mettre toutes les boîtes dans les cartons de 1×40 et de 2×20 . Par contre, on peut mettre toutes les boîtes dans les deux autres cartons, par exemple comme cela :



Il existe encore plusieurs manières différentes de ranger les boîtes dans un carton pour 20 pralinés, mais ce n'est pas possible de les ranger dans un carton ayant moins de deux places vides.

C'est de l'informatique !

Dans cet exercice du Castor, il faut ranger des rectangles de manière à ce que le rectangle les contenant soit le plus petit possible. Ce problème est connu en informatique sous le nom de « rectangle packing » et est l'un de nombreux problèmes d'emballage. Nous pouvons trouver relativement facilement la solution *optimale* pour un petit nombre de rectangles (ici, le plus petit carton). Par contre, il est nécessaire d'automatiser le procédé pour un nombre d'objets plus grand ; nous avons donc besoin d'un algorithme qui peut être exécuté en tant que programme informatique. Malheureusement, le problème de « rectangle packing », comme beaucoup d'autres problèmes d'emballage, est *NP-complet*. Cela veut dire qu'il n'existe probablement pas d'algorithme efficace trouvant la solution optimale.



En informatique, on développe des algorithmes efficaces ne trouvant pas obligatoirement la solution optimale, mais de bonnes solutions aux problèmes NP-complets.

Les méthodes efficaces permettant d'arranger des marchandises dans des étagères, de les emballer ou de les distribuer dans des containers sont très importantes pour les entreprises de logistique, par exemple. De plus, d'autres problèmes peuvent être décrits sous la forme de problèmes d'emballage. Une tâche nécessitant M heures de travail par N travailleurs peut par exemple être représentée par un rectangle de taille $N \times M$. On peut ainsi limiter le nombre d'heures et de personnes nécessaires à un procédé en résolvant le problème de «rectangle packing» de manière optimale.

Mots clés et sites web

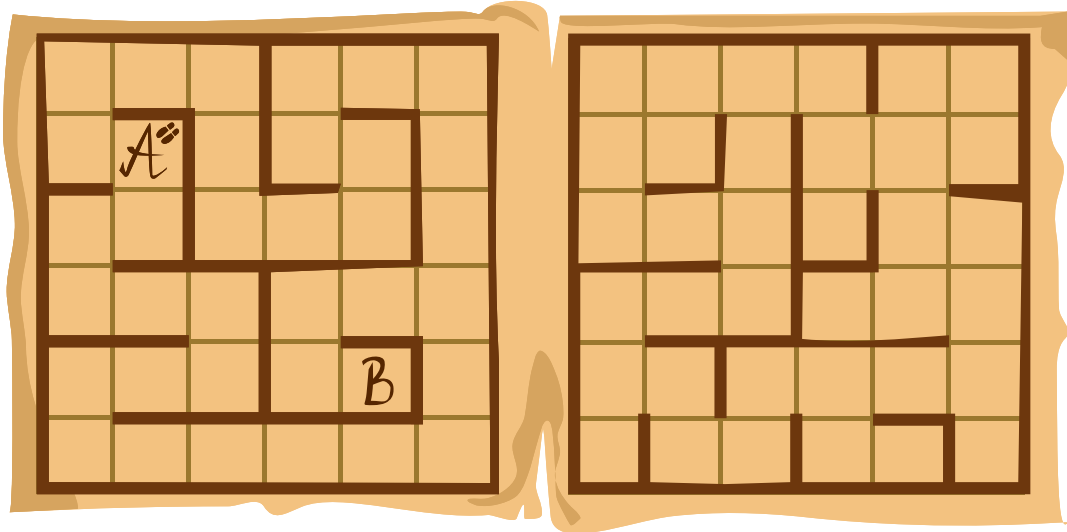
- Problème NP-complet : https://fr.wikipedia.org/wiki/Problème_NP-complet
- Optimisation : [https://fr.wikipedia.org/wiki/Optimisation_\(mathématiques\)](https://fr.wikipedia.org/wiki/Optimisation_(mathématiques))
- Complexité d'un algorithme (efficacité) :
https://fr.wikipedia.org/wiki/Analyse_de_la_complexité_des_algorithmes





15. Labyrinthe

L'école de magie a deux étages. Les étages sont exactement l'un au-dessus de l'autre. Ils sont tous les deux divisés en cases, et il y a des murs entre certaines cases :



Ron, un élève magicien, a besoin d'une seconde pour passer d'une case à l'autre sans changer d'étage. Malheureusement, Ron a oublié comment passer à travers les murs ; mais il peut passer d'une case sur un étage à la même case sur l'autre étage. Cela lui prend cinq secondes.

Ron aimerait aller de la case A à la case B le plus vite possible.

De combien de secondes au minimum Ron a-t-il besoin ?

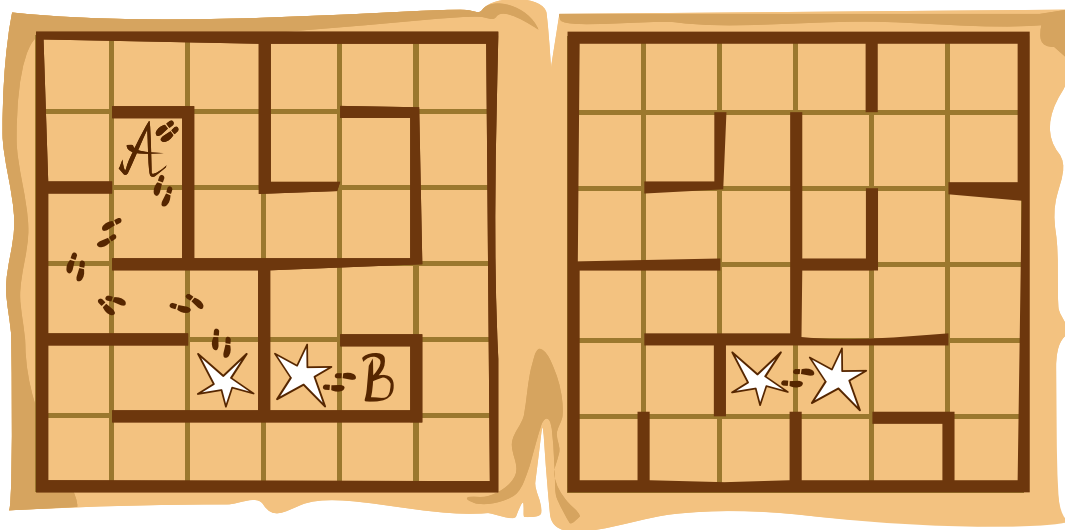
- A) 6 secondes
- B) 16 secondes
- C) 18 secondes
- D) 20 secondes



Solution

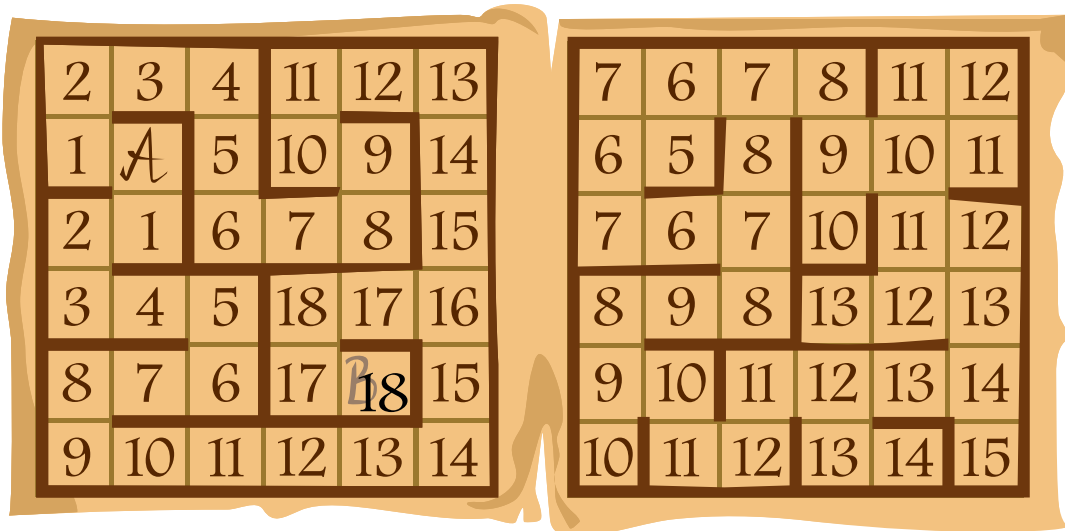
La réponse C) 18 secondes est juste.

Ron peut aller de la case A à la case B en 18 secondes en suivant le chemin suivant :



Mais est-ce le chemin le plus rapide ? Les « temps les plus courts » dont Ron a besoin pour passer de la case A à n'importe quelle autre case peuvent être calculés au fur et à mesure comme ceci : Le temps le plus court pour arriver à la case A est bien sûr 0 secondes. Ensuite, on continue étape par étape comme ceci : on choisit entre toutes les cases pour lesquelles un temps le plus court a déjà été calculé celle avec la valeur la plus basse (au départ, on choisit donc la case A). À partir de la case choisie, on considère toutes les cases possibles à l'étape suivante et recherche le chemin le plus rapide pour y arriver ; on inscrit les temps calculés dans les cases. C'est possible qu'un temps déjà inscrit plus tôt soit amélioré. La case choisie ne peut ensuite plus être prise en considération ; elle ne peut donc pas être choisie aux étapes suivantes.

Voici les temps les plus courts calculés avec cette méthode en partant de la case A :





Ron a donc en effet besoin d'au moins 18 secondes pour aller de la case A à la case B. La réponse A, 6 secondes, serait juste s'il n'y avait pas de mur entre les cases. Si Ron changeait quand même une fois d'étage, le temps serait rallongé de 10 secondes et on obtiendrait la réponse B, 16 secondes. S'il n'y avait que l'étage avec les points A et B, 20 secondes seraient nécessaires, et la réponse D serait juste.

C'est de l'informatique !

Il est souvent nécessaire de calculer les chemins les plus rapides ou les plus courts ; la planification d'itinéraire par les applications de navigation modernes en sont un exemple évident. Le problème peut être fortement simplifié si les chemins sont composés d'étapes individuelles entre des points voisins et que le « coût » de ces étapes est connu : coût en temps, argent, consommation d'énergie, quelle que soit la dimension importante dans le problème. Dans ce cas, les points, étapes et coûts des étapes peuvent être représentés dans un *graphe*. Beaucoup d'algorithmes permettant de calculer le *plus court chemin* dans un graphe de manière efficace sont connus en informatique. L'un d'entre eux, l'*algorithme de Dijkstra*, décrit par l'informaticien Edsger Dijkstra, a été utilisé dans l'explication de la réponse ci-dessus.

Les plus courts chemins jouent aussi un rôle important dans le développement de circuits pour les ordinateurs. Les composants du circuit doivent être reliés les uns aux autres au moindre coût. Les circuits modernes sont composés de plusieurs niveaux, et une connection entre deux niveaux est plus chère qu'une connection semblable sur le même niveau – comme le changement d'étage dans cet exercice du Castor est plus cher qu'une étape sur le même étage.

Mots clés et sites web

- Graphe : [https://fr.wikipedia.org/wiki/Graphe_\(mathématiques_discrètes\)](https://fr.wikipedia.org/wiki/Graphe_(mathématiques_discrètes))
- Plus court chemin : https://fr.wikipedia.org/wiki/Problème_de_plus_court_chemin
- Edsger Dijkstra : https://fr.wikipedia.org/wiki/Edsger_Dijkstra
- Algorithme de Dijkstra : https://fr.wikipedia.org/wiki/Algorithme_de_Dijkstra



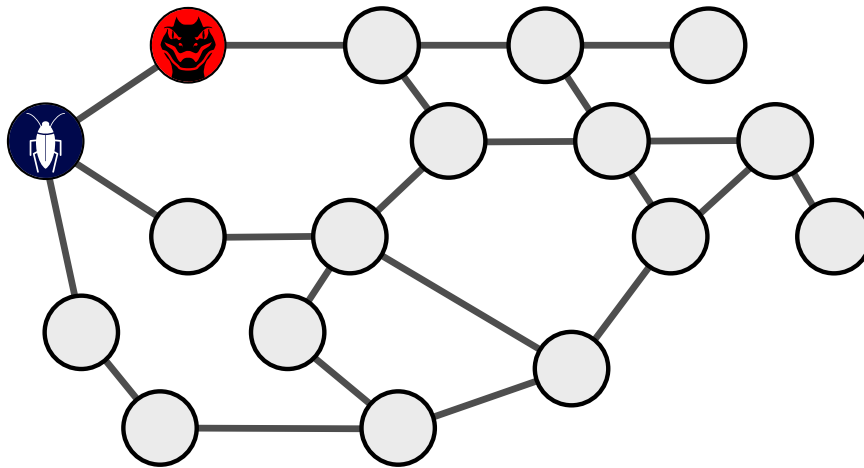


16. Virus

Deux nœuds d'un réseau informatique ont été infectés par des virus informatiques : l'un avec le virus BlueBug 🐛, l'autre avec le virus RedRaptor 🦇. Chaque matin, les deux virus se propagent : chaque virus infecte tous les nœuds qui sont directement reliés aux nœuds qu'il a déjà infectés. Lorsqu'un nœud est infecté par les deux virus, il se désactive au bout de quelques heures à cause de la surcharge 🚫. Les virus ne peuvent plus se propager depuis les nœuds désactivés les jours suivants.

Tu vois ci-dessous le réseau informatique avec les nœuds et leurs connexions directes. Les deux nœuds infectés au départ sont indiqués. Après quelques jours, tous les nœuds sont infectés ou désactivés.

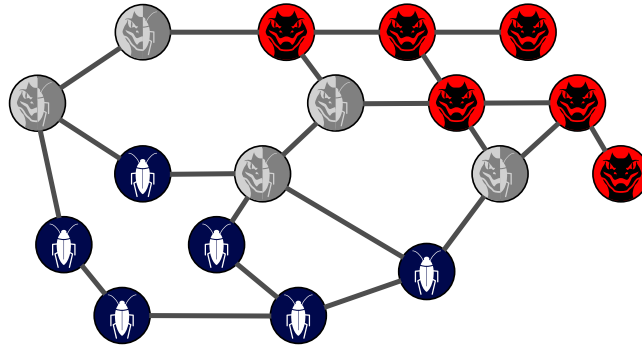
Quels nœuds sont alors désactivés, infectés par BlueBug 🐛 ou infectés par RedRaptor 🦇 ? Choisis le bon symbole pour chaque nœud.



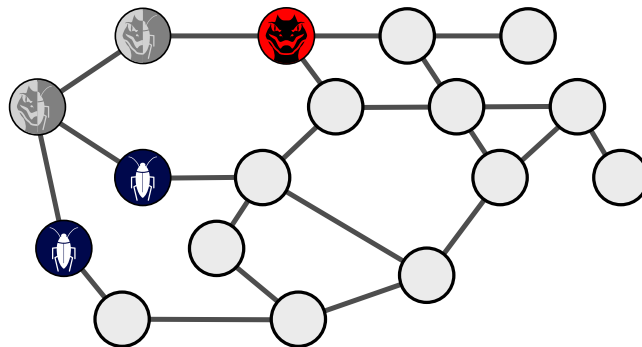


Solution

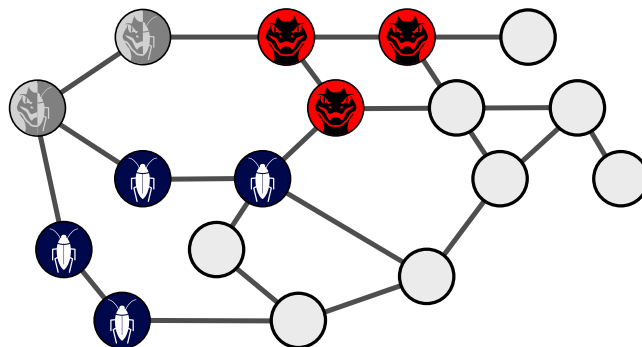
Tous les nœuds sont infectés ou désactivés après cinq jours. Voici la bonne solution :



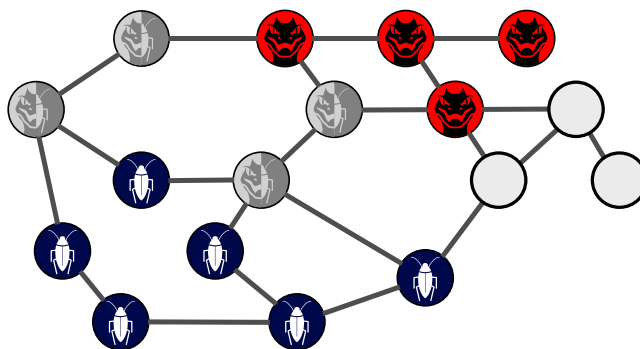
Cinq nœuds sont infectés après un jour. Les deux nœuds infectés au départ sont désactivés parce qu'ils ont été infectés par les deux virus :



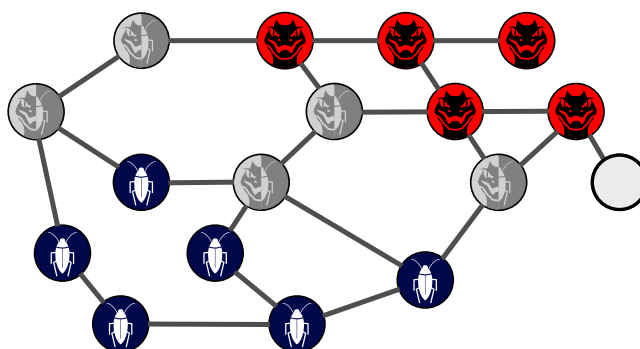
Quatre nœuds supplémentaires sont infectés au bout de deux jours :



Après trois jours, deux nœuds supplémentaires sont infectés par les deux virus et donc désactivés. Trois nœuds de plus sont infectés par BlueBug et deux par RedRaptor :



Après quatre jours, un nouveau nœud est désactivé. BlueBug ne peut plus se propager :



Le dernier nœud est infecté par RedRaptor le cinquième jour.

C'est de l'informatique !

Les virus et autres logiciels malveillants sont une grande menace pour les réseaux informatiques. Ils n'influencent pas uniquement la capacité des ordinateurs touchés, mais ont souvent une *charge utile* qui inflige des dommages supplémentaires. Par exemple, certains virus peuvent lire les données transmises, découvrir des informations sensibles comme des mots de passes et les transmettre à un destinataire. Dans certains cas, les virus peuvent chiffrer des données enregistrées sur l'ordinateur touché : si l'utilisateur veut accéder à ses données, il doit d'abord verser de l'argent sur un compte en banque. Parfois, des groupes d'ordinateurs infectés peuvent être contrôlés à distance par des criminels pour attaquer d'autres ordinateurs (*Botnet*).

Habituellement, les créateurs de virus ne cherchent pas à ce que les virus désactivent complètement un ordinateur, car cela empêche la propagation du virus. Cependant, certains virus sont créés spécialement pour le sabotage et la cyberguerre. Ceux-ci peuvent même endommager les ordinateurs infectés durablement.

L'installation des mises à jour de sécurité est importante pour la défense contre les virus. Les logiciels anti-virus peuvent améliorer la sécurité, mais sont déjà inclus dans certains systèmes d'exploitations, ce qui peut les rendre inutiles. Une sauvegarde régulière des données et une attention au comportement du système pour détecter des choses inhabituelles sont cependant indispensables.



Mots clés et sites web

- Réseau informatique : https://fr.wikipedia.org/wiki/Réseau_informatique
- Virus : https://fr.wikipedia.org/wiki/Virus_informatique
- Charge utile : https://fr.wikipedia.org/wiki/Charge_utile#Informatique
- Botnet : <https://fr.wikipedia.org/wiki/Botnet>
- Cyberguerre : <https://fr.wikipedia.org/wiki/Cyberguerre>

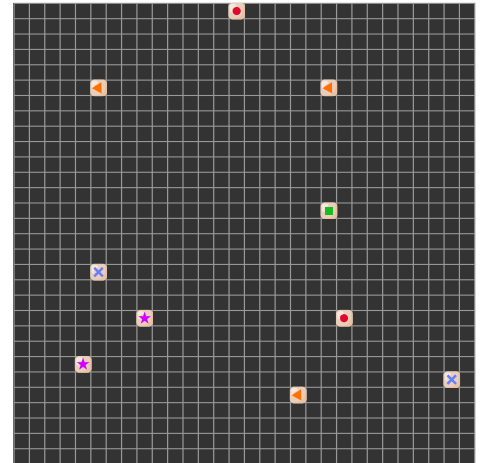


17. Carrelage

Le sol d'une pièce carrée est divisé en 30×30 cases. Sur dix des cases sont posées des puces avec les symboles colorés suivants : ●, ✕, ◀, ■ et ☆.

Un robot doit décorer le sol case par case avec ces symboles. Il utilise pour cela quatre règles différentes. Il décore une case sur laquelle il n'y a pas de puce avec...

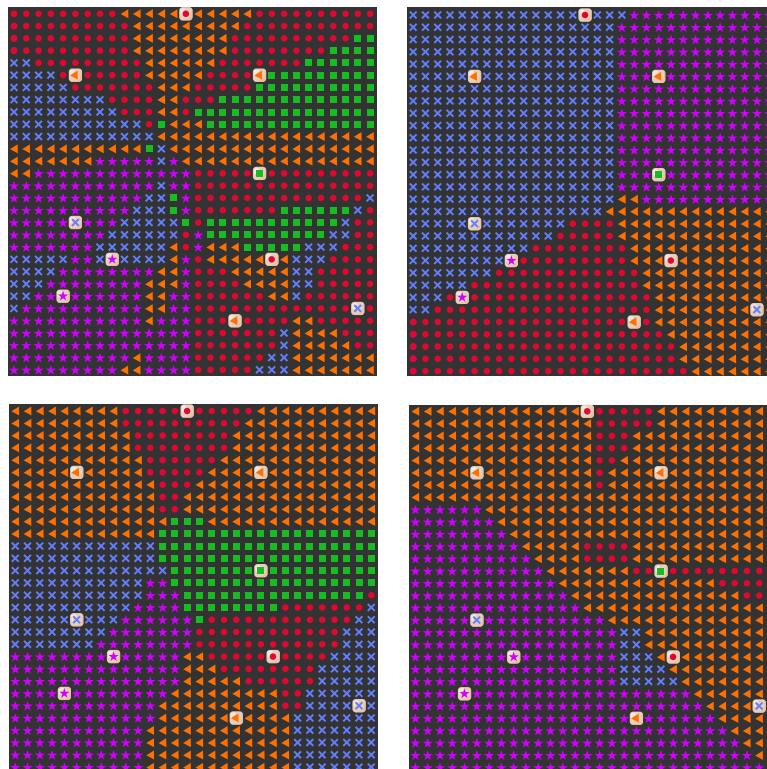
- 1 ... le symbole de la puce la plus proche de lui.
- 2 ... le symbole de la puce la plus éloignée de lui.
- 3 ... le symbole de la deuxième puce la plus proche de lui.
- 4 ... le symbole le plus fréquent parmi les six puces les plus proches de lui.



Le robot décore toutes les cases d'après la même règle. S'il y a plusieurs symboles possibles pour une case d'après la règle utilisée, le robot en choisit un au hasard.

Tu peux voir ci-dessous comment le sol est décoré avec chacune des règles.

À quelle règle correspond chaque sol ? Assigne les règles aux sols correspondants.

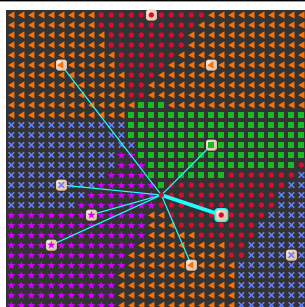




Solution

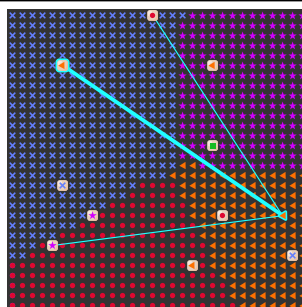
Comme toutes les cases d'un sol sont décorées en suivant la même règle, il suffit de vérifier quelle règle est utilisée pour une seule case. Nous considérons une case différente pour chaque sol :

Règle 1



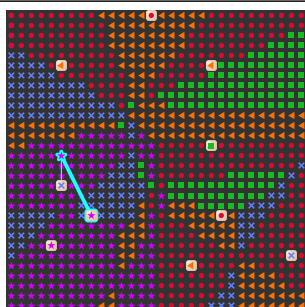
La case est décorée avec un parce la puce la plus proche est .

Règle 2



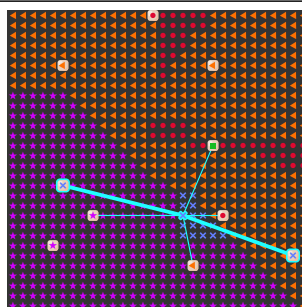
La case est décorée avec un parce que la puce la plus éloignée est .

Règle 3



La case est décorée avec une parce que la deuxième puce la plus proche est .

Règle 4



La case est décorée avec une parce que la puce est la plus fréquente parmi les six puces les plus proches.

C'est de l'informatique !

La division d'un plan et sa construction *algorithmique* jouent un rôle important dans différents domaines de l'informatique, par exemple en simulation et en graphisme.

Les *diagrammes de Voronoi*, nommés d'après le mathématicien ukrainien Georgi Feodosjewitsch Voronoi (*1868 - †1908), divisent un plan en *cellules* centrées sur un *germe*. Tous les points d'une cellule sont plus proches de leur germe que de tous les autres germes ; le résultat de la règle 1 est un diagramme de Voronoi. Ces diagrammes sont des représentations fréquentes du monde réel, par exemple des réseaux de téléphonie mobile. Ils sont aussi utilisé pour l'analyse de matchs de football ou d'autres processus socio-économiques, comme la relation entre la population et les écoles, hôpitaux ou autres fournisseurs de services à proximité.



Le météorologue Alfred H. Thiessen (*1872 - †1956) a développé en 1911 une méthode basée sur les diagrammes de Voronoi permettant de déterminer les valeurs moyennes régionales (les volumes de précipitations, par exemple) de manière plus fidèle à la réalité. Il n'a pas calculé la moyenne des mesures de différentes stations de mesure en ne prenant en compte que le nombre de stations, mais en prenant en compte la surface considérée par chaque station à l'aide d'un diagramme de Voronoi. Les mesures locales ont ainsi des poids différents dans le calcul de la moyenne pondérée.

Mots clés et sites web

- Algorithme : <https://fr.wikipedia.org/wiki/Algorithme>
- Diagramme de Voronoi : https://fr.wikipedia.org/wiki/Diagramme_de_Voronoi
- Moyenne pondérée : https://fr.wikipedia.org/wiki/Moyenne_pondérée



A. Auteur·e·s des exercices

 Esraa Almajhad	 Monika Maneva
 Liam Baumann	 Zoran Milevski
 Wilfried Baumann	 Madhavan Mukund
 Linda Björk Bergsveinsdóttir	 Ágnes Erdősne Németh
 Graeme Buckie	 Ilze Nilandere
 Marta J. Burzanska	 Veronika Ognjanovska
 Sarah Chan	 Mārtiņš Opmanis
 Kris Coolsaet	 Elsa Pellet
 Darija Dasović	 Margot Phillipps
 Christian Datzko	 Zsuzsa Pluhár
 Susanne Datzko	 Wolfgang Pohl
 Justina Dauksaite	 John-Paul Pretti
 Nora A. Escherle	 Susannah Quidilla
 Gerald Futschek	 Lorenzo Repetto
 Adam Grodeck	 Chris Roffey
 Benjamin Hirsch	 Kirsten Schlüter
 Alisher Ikramov	 Giovanni Serafini
 Thomas Ioannou	 Timur Sitdikov
 Mile Jovanov	 Bernadette Spieler
 Dong Yoon Kim	 Emil Stankov
 Hakin Kim	 Veronika Stefanovska
 Vaidotas Kinčius	 Alieke Stijf
 Lidija Kralj	 Goran Sukovic
 Regula Lacher	 Monika Tomcsányiová
 Taina Lehtimäki	 Jiří Vaníček
 Marielle Léonard	 Troy Vasiga



 Willem van der Vegt

 Michael Weigend

 Rechilda Villame

 Kyra Willekes



B. Sponsoring : Concours 2022

HASLERSTIFTUNG <http://www.haslerstiftung.ch/>



Kanton Zürich
Volkswirtschaftsdirektion
Amt für Wirtschaft und Arbeit

Standortförderung beim Amt für Wirtschaft und Arbeit Kanton Zürich



UBS

<http://www.ubs.com/>



<http://www.verkehrshaus.ch/>
Musée des transports, Lucerne



i-factory (Musée des transports, Lucerne)

senarclens
leu+partner
strategische kommunikation

<http://senarclens.com/>
Senarclens Leu & Partner

ABZ
AUSBILDUNGS- UND BERATUNGSZENTRUM
FÜR INFORMATIKUNTERRICHT

<http://www.abz.inf.ethz.ch/>
Ausbildungs- und Beratungszentrum für Informatikunterricht der ETH Zürich.

hep/ haute
école
pédagogique
vaud

<http://www.hepl.ch/>
Haute école pédagogique du canton de Vaud

Scuola universitaria professionale
della Svizzera italiana

SUPSI

<http://www.supsi.ch/home/supsi.html>
La Scuola universitaria professionale della Svizzera italiana
(SUPSI)



C. Offres supplémentaires



IT tout feu tout flamme : <https://it-feuer.ch/fr/>

En Suisse, un nombre considérable d'organisations s'engagent à promouvoir la prochaine génération d'informaticiennes et d'informaticiens. L'initiative «IT tout feu tout flamme» souhaite unir ces forces et contribuer ensemble à mieux faire connaître le sujet au public dans toute la Suisse. IT tout feu tout flamme présente une variété d'offres destinées au corps enseignant et aux élèves.



Coding club des filles :

<https://www.epfl.ch/education/education-and-science-outreach/fr/jeunepublic/coding-club/>

Programmer une application ? Inventer un jeu vidéo ? Créer une animation ? Si une de ces activités t'intéresse, cet espace est fait pour toi ! Viens échanger et partager tes idées, apprendre à coder et découvrir les métiers liés à l'informatique. Les filles de 11 à 15 ans intéressées par la programmation et l'informatique peuvent participer aux ateliers du Coding club des filles.



Roteco : <https://www.roteco.ch/fr/>

Le projet Roteco existe autour d'une communauté d'enseignantes et enseignants qui souhaitent préparer leurs élèves à évoluer dans une société numérique. Au sein de cette communauté, ils cherchent, testent, développent et partagent des activités de robotique éducative et de science informatique adaptées pour leurs classes. Ils sont informés des derniers événements ou ateliers concernant la robotique et plus largement des activités de science informatique à proximité de leur établissement.



010100110101011001001001
010000010010110101010011
010100110100100101000101
001011010101001101010011
010010010100100100100001

SS!E

www.svia-ssie-ssii.ch
schweizerischervereinfürinformatikind
erausbildung//sociétésuissepourl'infor
matique dans l'enseignement//societàsviz
zeraperl'informaticanell'insegnamento

Devenez vous aussi membre de la SSIE

<http://svia-ssie-ssii.ch/la-societe/devenir-membre/>

et soutenez le Castor Informatique par votre adhésion

Peuvent devenir membre ordinaire de la SSIE toutes les personnes qui enseignent dans une école primaire, secondaire, professionnelle, un lycée, une haute école ou donnent des cours de formation ou de formation continue.

Les écoles, les associations et autres organisations peuvent être admises en tant que membre collectif.