INFORMATIK-BIBER SCHWEIZ
CASTOR INFORMATIQUE SUISSE
CASTORO INFORMATICO SVIZZERA

Exercices et solutions 2025

Années HarmoS 7/8

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

Éditeurs :

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

SSIE
www.svia-ssie-ssii.ch
schweizerischer verein für informatik in der ausbildung // société suisse pour l'informatique dans l'enseignement // società svizzera per l'informatica nell'insegnamento

HASLERSTIFTUNG ETH zürich ABZ Ausbildungs- und Beratungszentrum für Informatikunterricht SUPSI Scuola universitaria professionale della Svizzera italiana hep/ PH LUZERN Pädagogische Hochschule

Ont collaboré au Castor Informatique 2025

Masiar Babazadeh, Jean-Philippe Pellet, Andrea Maria Schmid, Giovanni Serafini, Susanne Thut

Cheffe de projet : Nora A. Escherle

Nous adressons nos remerciements pour le travail de développement des exercices du concours à :
Patricia Heckendorn, Gymnasium Kirschgarten
Juraj Hromkovič, Regula Lacher : ETH Zürich, Ausbildungs- und Beratungszentrum für Informatikunterricht
Jens Hartmann, Stephan Koch, Dirk Schmerenbeck und Jacqueline Staub : Universität Tier, Allemagne

Le choix des exercices a été fait en collaboration avec les organisateur de Bebras en Allemagne, Autriche et Hongrie. Nous remercions en particulier :
Philip Whittington, Silvan Horvath : ETH Zürich, Ausbildungs- und Beratungszentrum für Informatikunterricht
Wolfgang Pohl, Karsten Schulz, Franziska Kaltenberger, Margaretha Schlüter, Kirsten Schlüter, Michael Weigend : Bundesweite Informatikwettbewerbe (BWINF), Allemagne
Wilfried Baumann : Österreichische Computer Gesellschaft, Autriche
Gerald Futschek, Lukas Lehner : Technische Universität Wien, Autriche
Zsuzsa Pluhár, Bence Gaal : ELTE Informatikai Kar, Hongrie

La version en ligne du concours a été réalisée sur l'infrastructure cuttle.org. Nous remercions pour la bonne collaboration :
Eljakim Schrijvers, Justina Oostendorp, Alieke Stijf, Kyra Willekes : cuttle.org, Pays-Bas
Andrew Csizmadia : Raspberry Pi Foundation, Royaume-Uni

Les tâches de programmation ont été créées et développées spécialement pour la plate-forme en ligne. Nous remercions chaleureusement pour leur initiative :
Jacqueline Staub : Universität Tier, Allemagne
Dirk Schmerenbeck : Universität Trier, Allemagne
Dave Oostendorp : cuttle.org, Pays-Bas

Pour le support pendant les semaines du concours, nous remercions en plus :
Eveline Moor : Société suisse pour l'informatique dans l'enseignement

Nous remercions les personnes suivantes pour l'organisation et la réalisation de la finale suisse :
Dennis Komm, Hans-Joachim Böckenhauer, Angélica Herrera Loyo, Andre Macejko, Moritz Stocker, Philip Whittington, Silvan Horvath : ETH Zürich, Ausbildungs- und Beratungszentrum für Informatikunterricht
Pour la correction des épreuves :
Clemens Bachmann, Morel Blaise, Tobias Boschung, Davud Evren, Jay Forrer, Sven Grübel, Urs Hauser, Fabian Heller, Jolanda Hofer, Alessandra Iacopino, Saskia Koller, Richard Královič, Jan Mantsch, Adeline Pittet, Alexander Skodinis, Emanuel Skodinis, Jasmin Sudar, Valerie Verdan, Chris

Wernke
Pour la traduction française des épreuves :
Jean-Philippe Pellet : Haute école pédagogique du canton de Vaud

Christoph Frei : Chragokyberneticks (Logo Castor Informatique Suisse)
Andrea Leu, Sarah Beyeler, Maggie Winter : Senarclens Leu + Partner AG

Des remerciements particuliers sont dûs pour leur grand soutien à Juraj Hromkovič, Dennis Komm, Gabriel Parriaux et la Fondation Hasler. Sans eux, ce concours n'existerait pas.

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 2025 a été réalisé par la Société suisse pour l'informatique dans l'enseignement (SSIE) et généreusement soutenu par la Fondation Hasler. Parmi les autres partenaires et sponsors qui ont soutenu financièrement le concours, citons Abraxas Informatik AG, l'Office de l'école obligatoire et du conseil (OECO) du canton de Berne, l'Office de l'économie (AWI) du canton de Zurich, CYON AG et UBS.

Cette brochure a été produite le 10 décembre 2025 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 1er décembre 2025.

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.bebras.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 d'imagination. Les exercices sont expressément conçus pour développer un intérêt durable pour l'informatique, au-delà de la durée du concours.

Le concours Castor Informatique 2025 a été fait pour cinq tranches d'âge, basées sur ces années scolaires :

- Années HarmoS 5 et 6
- 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 fausse 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 fausse	−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 les années HarmoS 5 et 6, et 36 pour les années HarmoS 7 et 8).

Le nombre de points maximal était ainsi de 180 (ou 108 pour les années HarmoS 5 et 6, 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 :

Société suisse pour l'informatique dans l'enseignement
SVIA-SSIE-SSII
Castor Informatique
Jean-Philippe Pellet

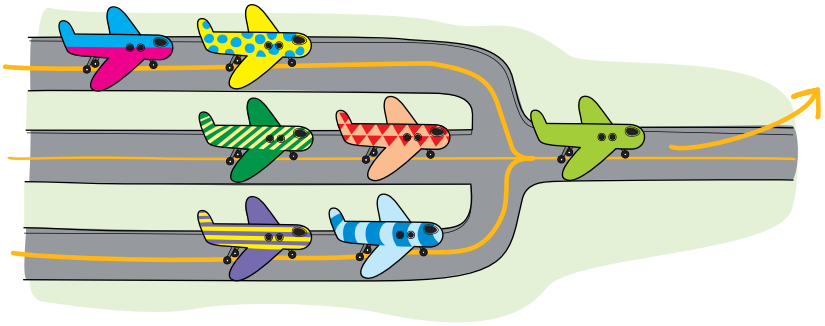
`https://www.castor-informatique.ch/kontaktieren/`
`https://www.castor-informatique.ch/`

Table des matières

1. Avions

Sept avions veulent décoller ce matin. Tous décollent de la même piste de décollage à droite.

Les avions avancent sur les lignes et ne peuvent pas se dépasser les uns les autres.



L'horaire montre dans quel ordre les sept avions décollent. Certains avions manquent encore.

Complète l'horaire avec les avions manquants.

Heure	
10:45	
10:52	
10:55	
10:59	
11:00	
11:10	
11:11	

Solution

Voici la bonne réponse :

Heure	
10:45	
10:52	
10:55	
10:59	
11:00	
11:10	
11:11	

1. En premier (à 10h45), l'avion  décolle ; il est déjà sur la piste de décollage.
2. En deuxième (à 10h52), l'avion  décolle.
3. En troisième (à 10h55), l'avion  décolle : il est devant l'avion  . Pour que celui-ci puisse partir en quatrième comme indiqué dans l'horaire, l'avion  doit avoir décollé avant.
4. En quatrième (à 10h59), l'avion  décolle.
5. En cinquième (à 11h00), l'avion  décolle : il est devant l'avion  . Pour que celui-ci puisse partir en sixième comme indiqué dans l'horaire, l'avion  doit avoir décollé avant.
6. En sixième (à 11h10), l'avion  décolle.
7. En septième et dernier (à 11h11), l'avion  décolle. Tous les autres avions ont déjà décollé.

C'est de l'informatique !

L'ordre de décollage des avions dépend aussi de l'ordre dans lequel ils attendent. Certains avions ne peuvent décoller qu'après le décollage des avions qui attendent devant eux. Dans un vrai aéroport, beaucoup d'autres conditions doivent être prises en considération lors de la planification de décollages, par exemple les retards, les problèmes techniques ou les changements météorologiques. Le plus souvent, des logiciels informatiques sont utilisés pour la planification : ils peuvent s'adapter rapidement aux changements tout en créant de bons horaires.

En informatique, l'élaboration d'horaire, comme l'horaire de décollage de cet exercice du castor, est un problème d'*ordonnancement*. L'ordonnancement peut être un problème compliqué, par exemple

lorsqu'une seule ressource (comme une piste de décollage) est à disposition, lorsque certains événements dépendent les uns des autres ou lorsqu'un certain ordre doit être respecté pour éviter des problèmes.

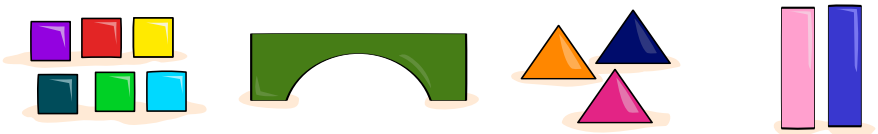
L'ordonnancement est un problème particulièrement difficile en informatique. Le défi est de calculer un plan optimal pour chaque situation possible. Lorsque de nombreux paramètres et contraintes doivent être pris en compte, même un ordinateur rapide peut avoir besoin de beaucoup de temps pour résoudre le problème. Ce genre de problème est appelé *NP-difficile*. Cela signifie qu'il est facile de déterminer si un horaire donné est correct, mais très difficile de trouver le meilleur des horaires. C'est comme pour un casse-tête compliqué : c'est facile de vérifier la solution, mais ça peut être dur de la trouver.

Mots clés et sites web

- Ordonnancement : `https://fr.wikipedia.org/wiki/Théorie_de_l'ordonnancement`
- NP-difficile : `https://fr.wikipedia.org/wiki/NP-difficile`

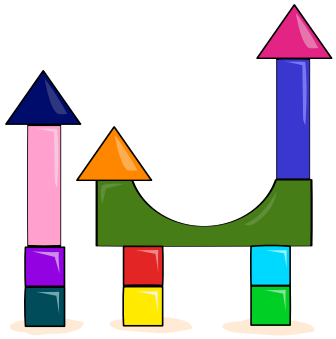
2. Jeu de construction

Ali a les plots suivants :



6 cubes | 1 pont | 3 pyramides | 2 parallélépipèdes

Zaha, la sœur d'Ali, lui dit ce qu'il doit faire avec les plots en lui donnant des instructions pas à pas. Ali suit chaque instruction tout de suite. Voilà sa construction à la fin :



Dans quel ordre Zaha a-t-elle donné les instructions ?

Die einzelnen Anweisungen müssen in beliebiger Reihenfolge angeordnet werden können. Es gibt diese fünf Anweisungen :

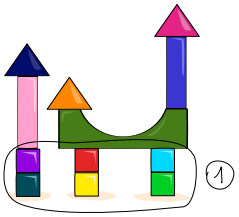
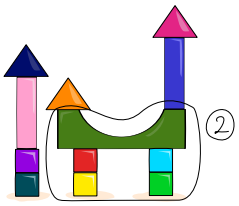
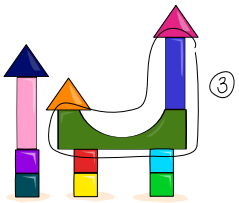
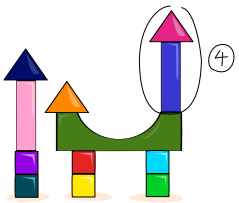
- Pose les deux parallélépipèdes sur ta construction.
- Aligne les tours de cubes.
- Pose les pyramides sur ta construction.
- Construis 3 tours de 2 cubes chacune.
- Pose le pont sur ta construction.

Solution

Zaha a donné les instructions dans cet ordre-là :

1. Construis trois tours de deux cubes chacune.
2. Aligne les tours de cubes.
3. Pose le pont sur ta construction.
4. Pose les deux parallélépipèdes sur ta construction.
5. Pose les pyramides sur ta construction.

Expliquons pourquoi cet ordre-là crée la construction d'Ali.

Image	Étape
	Comme les tours de cubes sont les seules parties de la construction à toucher par terre, elles doivent être construites en premier (instruction 1), puis être alignées (instruction 2).
	Ensuite, le pont doit être posé sur la construction (instruction 3). Il doit être sur les tours de cubes, mais sous les parallélépipèdes et les pyramides.
	Ensuite, il faut poser les parallélépipèdes (instruction 4), car ils sont directement sur le pont, mais sous les pyramides.
	Finalement, les pyramides doivent être posées (instruction 5). Elles sont sur les parallélépipèdes et aucun autre plot n'est posé en dessus.

Comme la plupart des instructions disent que des plots doivent être posés sur la construction déjà réalisée, l'ordre des instructions dépend directement de quels plots sont posés sur et sous quels autres plots.

Ce n'est pas possible de faire cette construction en suivant les instructions dans un autre ordre.

C'est de l'informatique !

Si des plots sont posés par terre les uns à côté des autres, c'est impossible de dire après coup dans quel ordre ils ont été posés. Les actions effectuées, c'est-à-dire le fait de poser les plots, sont indépendantes les unes des autres. Par contre, si on empile des plots les uns sur les autres, on ne peut le faire que l'un après l'autre en commençant par le bas. Il est nécessaire de poser le plot le plus bas pour y mettre le plot suivant.

Les programmes informatiques sont faits d'une suite d'instructions comme celle de Zaha dans cet exercice. Ces instructions peuvent être simples mais aussi très compliquées. Dans tous les cas, c'est important que les informaticiennes et informaticiens réfléchissent bien à l'effet de chaque instruction en programmant – et comment une instruction dépend des effets d'autres instructions. En prévoyant cela avant de programmer, ce n'est pas difficile de mettre les instructions dans le bon ordre – comme tu l'as fait dans cet exercice du Castor.

Mots clés et sites web

- *Instruction* : `https://fr.wikipedia.org/wiki/Instruction_informatique`

3. Retour à la maison

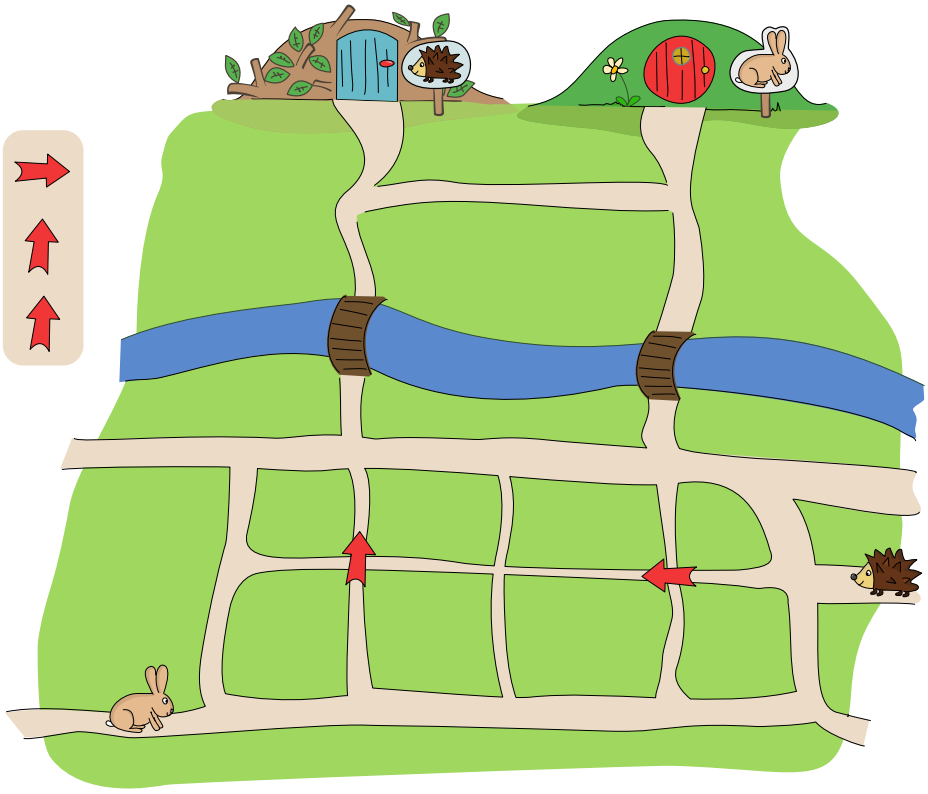
Un lapin  et un hérisson  veulent rentrer chez eux.

Chacun a sa propre maison :  et  .

Le lapin et le hérisson marchent tout droit sur les chemins. Ils ne tournent que quand ils arrivent à un croisement avec une flèche : ils suivent alors la direction de la flèche.

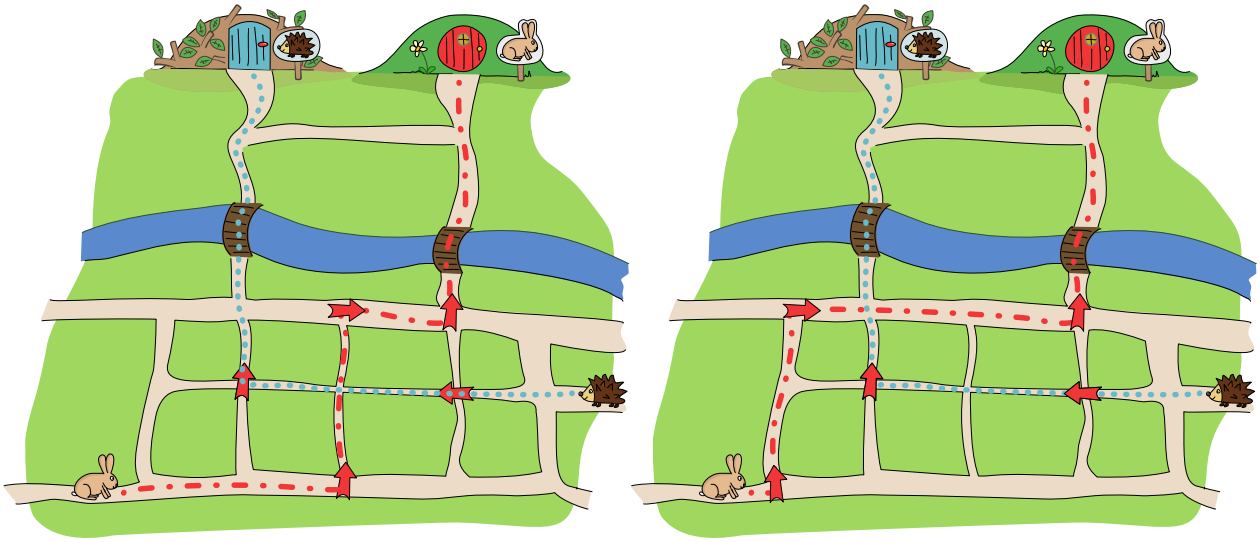
Il y a déjà des flèches à quelques croisements. Ces flèches mènent le hérisson chez lui, mais pas le lapin. Heureusement, il reste encore 3 flèches à placer.

Pose les 3 flèches restantes sur des croisements de manière à ce que le lapin ***et*** *le hérisson puissent rentrer chez eux.*



Solution

Il y a deux bonnes réponses :



Pour rentrer chez lui, le lapin doit tourner vers le « haut » (en direction des maisons). Il peut le faire au premier ou au troisième croisement par lequel il passe.

Ce qui est important, c'est que le lapin et le hérisson ne passent jamais par les mêmes flèches, parce qu'ils suivraient alors le même chemin et arriveraient à la même maison. C'est ce qui arriverait si le lapin tournait vers le haut au deuxième ou quatrième croisement par lequel il passe : il passerait par les flèches menant le hérisson chez lui. Le lapin ne peut pas non plus tourner vers le haut au cinquième croisement, car il devrait ensuite tourner à gauche pour arriver chez lui, et il n'y a pas de flèche vers la gauche.

Le lapin doit donc tourner vers le haut soit au premier, soit au deuxième croisement. Les deux autres flèches doivent être posées comme dessiné ci-dessus. Ce sont les seuls chemins menant le lapin chez lui sans déranger le chemin du hérisson.

C'est de l'informatique !

En rentrant chez eux, le lapin et le hérisson suivent la règle suivante : « S'il y a une flèche au croisement par lequel tu passes, suis la direction de la flèche ».

En informatique, de telles règles sont des *algorithmes*. L'algorithme ne change pas : il dit si et comment la direction doit changer à un croisement. Le changement de direction est déterminé par des flèches. Ces flèches sont des données qui sont traitées par l'algorithme ; on parle aussi d'*entrées* de l'algorithme. Dans l'algorithme de cet exercice du castor, l'entrée, donc la position et direction des flèches, détermine si la *sortie* de l'algorithme est juste, donc si les deux animaux rentrent chacun à sa maison.

Il pourrait y avoir une règle similaire pour le lavage d'un habit : « Choisis la température indiquée sur l'étiquette ». C'est seulement lorsque l'entrée, c'est-à-dire la température sur l'étiquette, est la bonne que la sortie est juste, c'est-à-dire que l'habit ressort propre et de la même taille qu'avant le lavage.

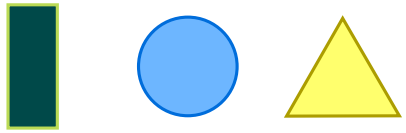
Ces exemples montrent qu'il est essentiel de faire attention aux entrées d'un algorithme, car la qualité des sorties dépend directement de la qualité des entrées. C'est par exemple le cas pour plusieurs systèmes d'IA. Les systèmes d'IA utilisent des modèles de la réalité créés à partir de données sur la réalité. Si ces données ne sont pas bien choisies, le modèle ne fonctionnera pas bien. Par exemple, un modèle de maladie ne fonctionnera pas bien pour les femmes si les données d'entrée ne concernent que les hommes.

Mots clés et sites web

- Recherche de chemin : `https://fr.wikipedia.org/wiki/Recherche_de_chemin`
- Algorithme : `https://fr.wikipedia.org/wiki/Algorithme`

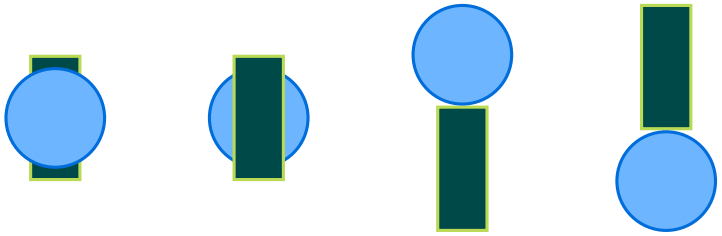
4. Hivobu

En Hivobu, la langue d'un pays lointain, ces trois formes sont prononcées :



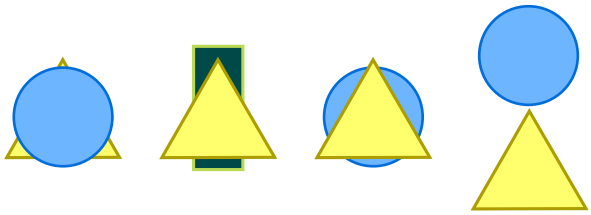
RAH OH TEH

En Hivobu, lorsque l'on met deux formes l'une derrière l'autre ou l'une en dessous de l'autre, cela se dit comme ça :



OH RAH CO RAH OH CO OH RAH DU RAH OH DU

Comment écrit-on TEH OH CO en Hivobu ?



A) B) C) D)

Solution

La bonne réponse est C.

Nous savons comment prononcer les différentes formes :

- Le rectangle  se dit RAH ;
- Le cercle  se dit OH ; et
- Le triangle  se dit TEH.

Nous savons aussi comment dire quand deux formes sont l'une derrière l'autre ou l'une au-dessous de l'autre : on commence par le nom de la première forme, puis de la deuxième forme, et ensuite on ajoute

- CO si la deuxième forme est derrière la première,
- DU si la deuxième forme est sous la première.

Donc si l'on a écrit, comme dans la réponse C, un triangle (TEH) et un cercle (OH) qui se trouve derrière (CO) le triangle, cela se dit TEH OH CO. Mais comment se disent les autres réponses en Hivobu ?

- La réponse A se dit OH TEH CO : un cercle (OH) et un triangle (TEH) derrière (CO).
- La réponse B se dit TEH RAH CO : un triangle (TEH) et un rectangle (RAH) derrière (CO).
- La réponse D se dit OH TEH DU : un cercle (OH) et un triangle (TEH) en dessous (DU).

C'est de l'informatique !

Est-ce que les ordinateurs sont intelligents ? Ils peuvent faire des calculs compliqués très rapidement, trouver des informations sur internet et bien plus. Mais pour qu'ils puissent faire ça, nous devons leur dire exactement ce qu'ils doivent faire. Même les logiciels d'IA avec lesquels on peut discuter ne fonctionnent que si on leur a dit exactement comment faire.

Les ordinateurs ne comprennent que des instructions précises qui doivent être formulées dans des langages qui ont des structures claires. Ces langages sont appelés *langages formels*, et les ordinateurs comprennent le mieux les langages qui ont une structure simple.

C'est le cas du langage de cet exercice du castor : ce que nous avons appris du langage Hivobu concernant les formes a une structure très simple. On commence par nommer les deux formes, puis on ajoute un mot qui décrit leur position. Si l'on ne respecte pas cette *syntaxe*, par exemple en mettant le dernier mot au milieu, personne ne sait ce que l'on veut dire en Hivobu. Cette forme de syntaxe est

appelée *notation post-fixée* en informatique. Un ordinateur s'attendant à une instruction en notation postfixée ne comprendrait pas non plus une instruction dans le mauvais sens.

Mots clés et sites web

- Notation Post-fixée : `https://fr.wikipedia.org/wiki/Notation_polonaise_inverse`
- Langage formel : `https://fr.wikipedia.org/wiki/Langage_formel`

5. Bois pour le barrage

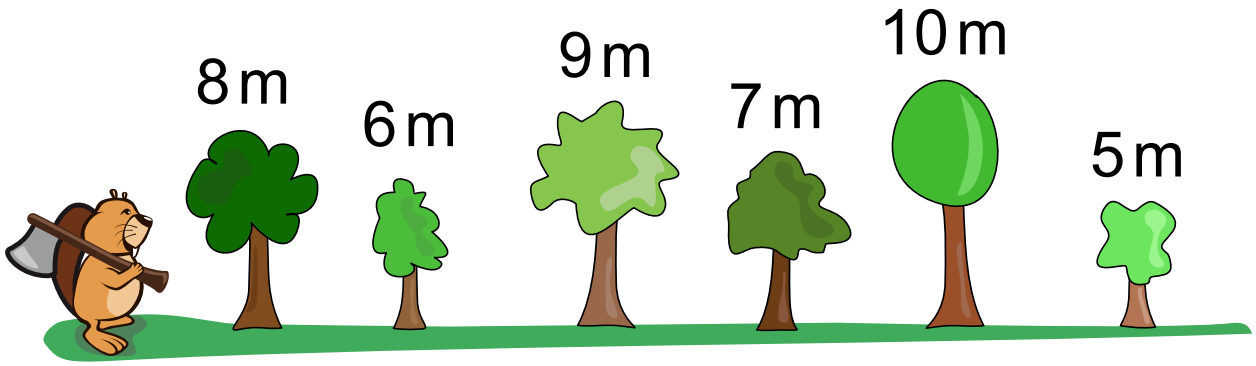
Les castors doivent couper plusieurs arbres pour la construction de leur prochain barrage.

Les castors ont 6 arbres à choix. Ils savent combien de mètres de bois donne chacun des arbres. Ils veulent avoir le plus de mètres de bois possible. Ils peuvent choisir le premier arbre librement. Pour choisir l'arbre suivant, ils doivent suivre ces deux règles :

- Règle 1 : l'arbre suivant doit être à la droite du dernier arbre abattu.
- Règle 2 : l'arbre suivant doit être plus petit, donc avoir moins de mètres de bois, que le dernier arbre abattu.

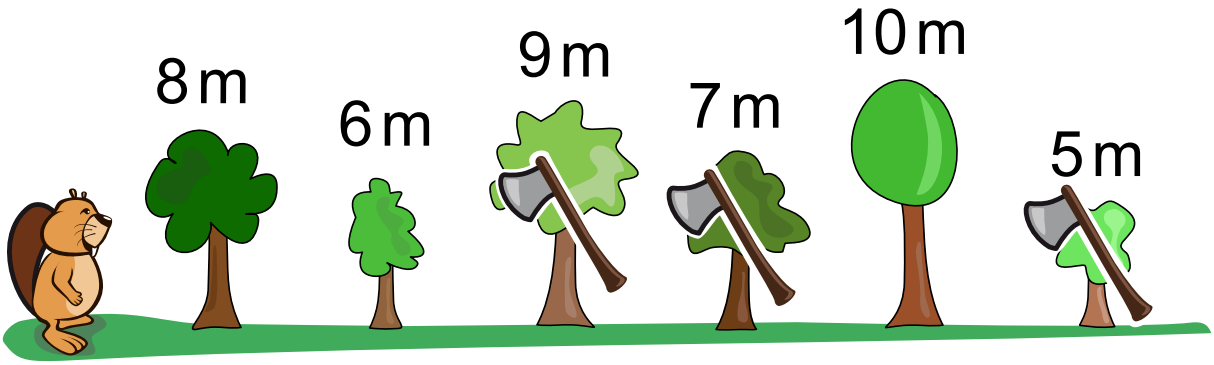
Par exemple, s'ils abattent l'arbre de 6 m, ils ne peuvent ensuite plus qu'abattre l'arbre de 5 m. Ils ont donc 11 mètres de bois à la fin.

Quels arbres les castors doivent-ils abattre pour avoir le plus de mètres de bois possible en suivant leurs règles ?



Solution

Voici la bonne réponse :



Le but de l'exercice est de trouver une suite partielle d'arbres avec un nombre de mètres de bois décroissant de gauche à droite parmi la suite de 6 arbres, et cela de manière à ce que la somme des mètres de bois soit maximale. Comme nous cherchons la somme maximale, nous ne devons considérer que les suites partielles ne pouvant plus être augmentées en respectant les règles. Nous appelons de telles suites *complètes*.

Si nous commençons avec l'arbre de 8 m (abrégé par 8), nous pouvons faire deux suites complètes : 8, 6, 5 et 8, 7, 5. Les suites 8, 7 (5 peut être ajouté) et 8, 5 (7 ou 6 peut être ajouté) ne sont pas complètes.

Le tableau suivant montre toutes les suites complètes et indique, pour chaque suite, le nombre de mètres de bois total correspondant.

Suite complète	**Mètres de bois**
8, 6, 5	19
8, 7, 5	20
6, 5	11
9, 7, 5	21
10, 5	15

Pour avoir le plus de bois possible, c'est-à-dire 21 mètres, en respectant les règles d'abattage, les castors doivent donc abattre les arbres 9, 7 et 5.

C'est de l'informatique !

Les castors cherchent à trouver un ensemble d'arbres respectant leurs règles de manière à avoir le plus grand nombre de mètres de bois possible. Dans cet exercice du castor, 21 mètres est l'*optimum*, c'est-à-dire la meilleure solution possible. Tu as donc résolu un *problème d'optimisation*.

De manière générale, un problème d'optimisation consiste à trouver la meilleure valeur possible. Cela peut être la plus grande valeur, comme dans cet exercice, mais aussi la plus petite valeur, par exemple si tu cherches le chemin le plus court pour aller à l'école. C'est dans la nature humaine de

rechercher la méthode la plus rapide, courte, peu chère ou rigolote : c'est de l'optimisation. C'est aussi de l'optimisation lorsqu'une entreprise cherche à payer le moins de salaires possible ou à gagner le plus d'argent possible en louant des appartements.

Beaucoup de problèmes d'optimisation sont tellement complexes qu'ils doivent être résolus à l'aide d'outils informatiques. Lorsqu'une entreprise de livraison veut planifier ses itinéraires pour consommer le moins d'énergie possible ou qu'un fournisseur d'énergie veut faire le plus de profit possible avec ses centrales électriques, il y a tellement de données et de conditions qui doivent être respectées tout en gardant une gestion flexible que les résultats obtenus par ordinateur sont souvent nettement meilleurs que ceux obtenus sans. Heureusement, il existe beaucoup de méthodes informatiques permettant de résoudre beaucoup de sortes de problèmes d'optimisation.

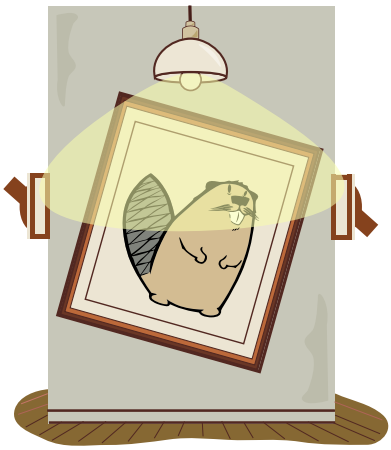
Pour les personnes intéressées : Le problème de cet exercice est appelé « Maximum Sum Decreasing Subsequence ». Regarde `ce site`. Tu peux y voir comment, en partant d'approches simples mais mauvaises, on peut améliorer pas à pas ces approches pour résoudre des problèmes d'optimisation et les implémenter dans des programmes informatiques.

Mots clés et sites web

- Optimisation : `https://fr.wikipedia.org/wiki/Optimisation_(mathématiques)`
- Recherche exhaustive : `https://fr.wikipedia.org/wiki/Recherche_exhaustive`

6. Étrange lampe

Victoria Volt a construit une lampe étrange. Cette lampe a deux interrupteurs : un à gauche et un à droite. Chacun de ces interrupteurs peut être soit **allumé** (,), soit **éteint** (,)



L'étrange lampe a encore un troisième interrupteur secret : une image ! Le fonctionnement des deux premiers interrupteurs change suivant l'inclinaison de l'image (, , ou).

Le tableau ci-dessous montre comment chaque interrupteur **allume** ou **éteint** la lumière dans chaque cas :

Image	Interrupteur	Lumière
	Un seul est **allumé** : ou	**allumée**
	sinon	**éteinte**
	Les deux sont **éteints** :	**éteinte**
	sinon	**allumée**
	Les deux sont **allumés** :	**allumée**
	sinon	**éteinte**

L'interrupteur de gauche est éteint et celui de droite est allumé . Comment l'image doit-elle être inclinée pour que la lumière soit éteinte ?

Solution

La réponse est : la lumière est éteinte lorsque l'image est inclinée vers la gauche  .

Regardons les trois inclinaisons possibles de l'image plus en détail :

Comme seul un interrupteur est **allumé**, la lumière est **allumée**. Cette réponse est fausse.

Comme les interrupteurs ne sont *pas* les *deux* **éteints**, la lumière est **allumée**. Cette réponse est également fausse.

Comme les interrupteurs ne sont *pas* les *deux* **allumés**, la lumière est **éteinte**. C'est la bonne réponse.

C'est de l'informatique !

Les interrupteurs de Victoria peuvent prendre chacun deux valeurs : **allumé** ou **éteint**. La lumière peut aussi prendre ces deux valeurs. Pour chaque position de l'image, on peut calculer la valeur de la lumière à partir des valeurs des interrupteurs comme décrit dans le tableau.

La plus petite unité de stockage des ordinateurs, le *bit*, peut lui aussi prendre deux valeurs. En informatique, ces valeurs sont le plus souvent appelées **1** et **0**, mais aussi **vrai** et **faux** ou **oui** et **non**. Comme l'on peut combiner des nombres à l'aide d'*opérateurs* ($+$, $-$, $\times$, :), on peut combiner des bits à l'aide d'*opérateurs logiques*. Ces opérateurs sont installés dans les ordinateurs sous la forme de *portes logiques*. Ces portes peuvent combiner et calculer avec des bits de toutes les manières imaginables : c'est grâce à cela que les ordinateurs peuvent traiter et modifier des données.

Dans cet exercice du castor, l'image détermine quelle opération logique est effectuée par les interrupteurs. Si l'image est droite, les interrupteurs fonctionnent comme une porte *OU exclusif* (XOR) : la lumière est allumée si **soit** l'interrupteur de gauche, **soit** celui de droite est allumé. Si l'image est inclinée à droite, ils fonctionnent comme une porte *OU* (OR) : si l'interrupteur de droite **ou** l'interrupteur de gauche est allumé, la lumière est allumée. Si l'image est inclinée à gauche, les interrupteurs fonctionnent comme une porte *ET* (AND) : la lumière est allumée si l'interrupteur de gauche **et** l'interrupteur de droite sont allumés.

Mots clés et sites web

- Logique : `https://fr.wikipedia.org/wiki/Logique`
- Porte logique : `https://fr.wikipedia.org/wiki/Porte_logique`

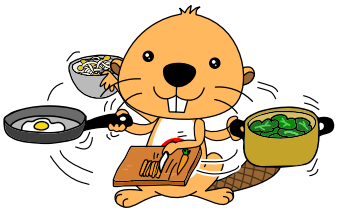
- Algèbre de Boole: `https://fr.wikipedia.org/wiki/Algèbre_de_Boole_(logique)`

7. Bibimbap

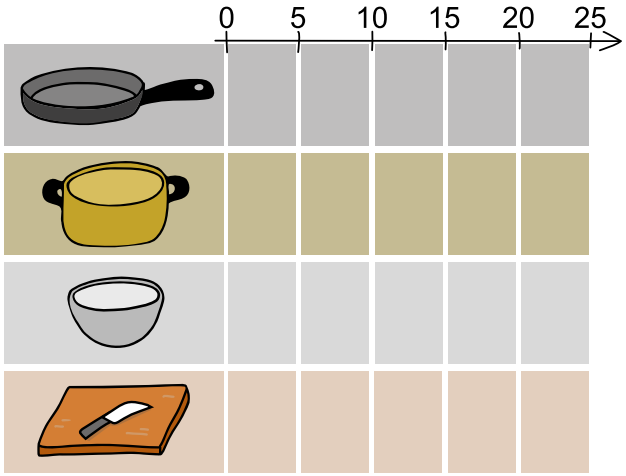
Un cuisinier aimerait préparer le plat traditionnel coréen bibimbap (비빔밥). Pour cela, il utilise (entre autres) quatre ustensiles : une casserole, une poêle, une planche et un saladier. Il prépare ensuite les quatre ingrédients comme cela :

- Épinards : d'abord cuire (pendant 10 min), ensuite couper (5 min).
- Pousses : d'abord laver (5 min), ensuite cuire (10 min).
- Carottes : d'abord couper (5 min), ensuite faire revenir (10 min).
- Œuf : faire cuire (5 min).

Le cuisinier peut utiliser plusieurs ustensiles en même temps, mais il ne peut utiliser chaque ustensile que pour un ingrédient à la fois. Par exemple, le cuisinier peut cuire des épinards dans la casserole en même temps qu'il fait cuire un œuf dans la poêle, mais il ne peut pas faire cuire un œuf et faire revenir des carottes dans la poêle en même temps.

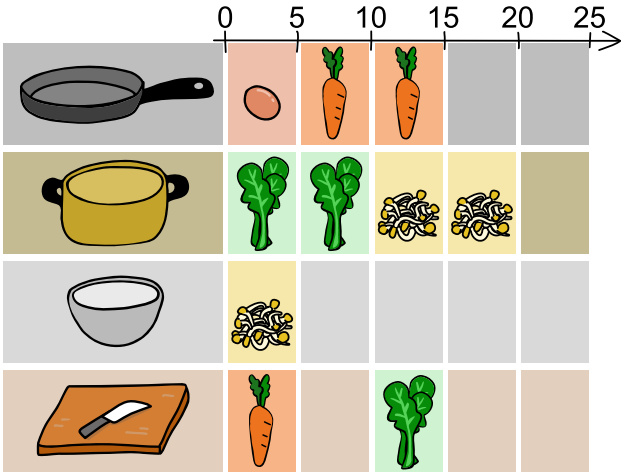


Établis une planification qui permette au cuisinier de préparer les ingrédients pour le bibimbap le plus rapidement possible.



Solution

Voici la bonne réponse :



Le cuisinier peut utiliser plusieurs ustensiles en même temps, mais chaque ustensile ne peut être utilisé que pour un ingrédient à la fois. Il faut donc réfléchir, pour chaque ustensile, à quel moment quel ingrédient peut y être préparé pour que la préparation complète dure le moins longtemps possible.

La préparation va durer au moins 20 minutes, car les épinards et les pousses doivent les deux cuire pendant 10 minutes dans la casserole. Si les épinards sont cuits en premier, les pousses peuvent être lavées pendant ce temps, et les épinards peuvent ensuite être coupés pendant que les pousses cuisent. L'œuf peut revenir dans la poêle pendant que le cuisinier coupe les carottes, et faire revenir celles-ci quand l'œuf est prêt – comme sur l'image ci-dessus.

La planification ci-dessus ne montre pas seulement qu'il faut au moins 20 minutes pour préparer les ingrédients, mais aussi qu'ils peuvent être préparés en 20 minutes maximum. Toutes les planifications qui durent 20 minutes tout en respectant les durées et l'ordre de préparation sont de bonnes réponses.

C'est de l'informatique !

Dans cet exercice du castor, il s'agit de planifier le déroulement de plusieurs activités. En informatique, ce type de tâches relève de l'*ordonnancement*. Comme pour beaucoup de problèmes d'ordonnancement, les ressources à disposition sont limitées : il n'y a qu'une poêle, qu'une casserole, qu'une planche et qu'un saladier. De plus, certaines activités peuvent avoir lieu au même moment et d'autres doivent se faire l'une après l'autre – parce qu'elles concernent le même ingrédient ou le même ustensile.

L'ordonnancement est un problème plus vieux que les premiers ordinateurs. Les techniques utilisées aujourd'hui par les outils de gestion de projet datent de l'époque des premiers ordinateurs, autour des années 1950. Une de ces méthodes, et celle que nous avons utilisée pour résoudre cet exercice, est celle du *chemin critique* : il s'agit de déterminer la plus longue suite d'activités dépendantes les unes des autres, et d'éviter les pauses dans sa réalisation.

Après la publication de sa méthode du chemin critique, John Fondahl, professeur à l'université de Stanford, n'était pas satisfait des implémentations informatiques de la méthode. Il a donc présenté ses propres approches pour appliquer le chemin critique sans ordinateur. C'est intéressant de noter que ce sont exactement ces approches qui sont utilisées encore aujourd'hui par la plupart des logiciels de gestion de projet. Les algorithmes existent depuis plus de mille ans, et il vaut toujours la peine de réfléchir à un algorithme sans tout de suite considérer comment il pourra être utilisé en informatique. Au final, un meilleur algorithme est toujours mieux adapté à l'implémentation par ordinateur.

Mots clés et sites web

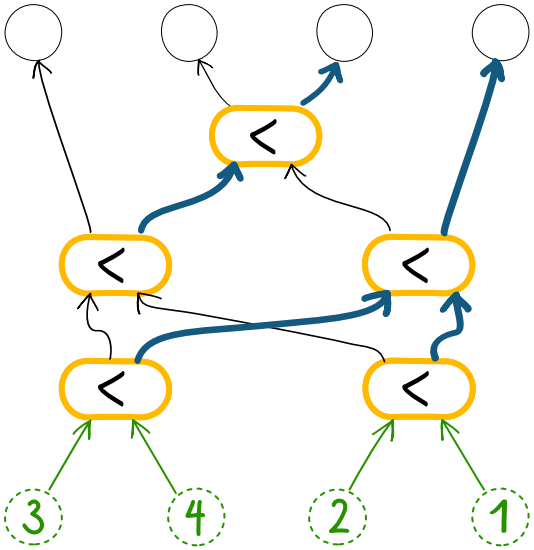
- Scheduling: `https://fr.wikipedia.org/wiki/Ordonnancement_(informatique)`
- Chemin critique: `https://fr.wikipedia.org/wiki/Chemin_critique`
- Stanford Technical Report John Fondahl (siehe Vorwort):
  `https://catalog.hathitrust.org/Record/005766951`
- Algorithme: `https://fr.wikipedia.org/wiki/Algorithme`

8. Mystérieuse machine

Les castors ont une «machine à nombres».

Quatre nombres sont entrés dans les champs d'entrée, par exemple 3, 4, 2 et 1.

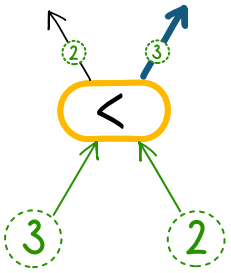
Les nombres traversent la machine de bas en haut en passant le long des flèches et des interrupteurs jusqu'à arriver aux champs de sortie.



Chacun des cinq interrupteurs compare les deux nombres entrants et dirige…

- … le plus petit nombre vers la gauche et
- … le plus grand nombre vers la droite.

Voici un exemple :



Quelle tâche cette machine accomplit-elle ?

A) Elle trie les nombres par ordre décroissant ; le résultat de l'exemple est 4, 3, 2, 1.

B) Elle trie les nombres par ordre croissant ; le résultat de l'exemple est 1, 2, 3, 4.

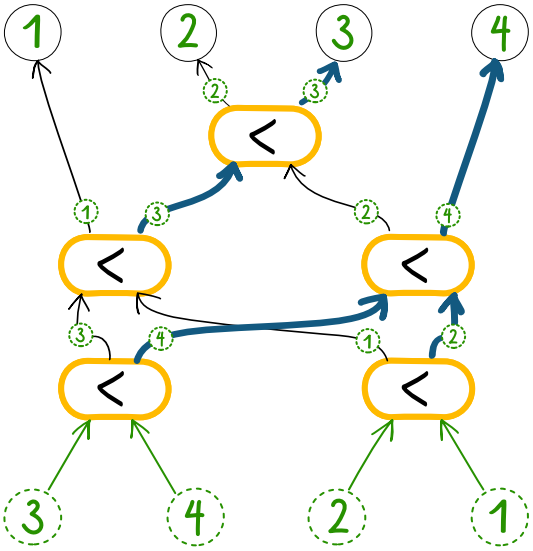
C) Elle donne les nombres dans leur ordre d'entrée ; le résultat de l'exemple est 3, 4, 2, 1.

D) Elle donne les nombres dans l'ordre inverse de leur ordre d'entrée ; le résultat de l'exemple est 1, 2, 4, 3.

Solution

La bonne réponse est B : la machine trie les nombres par ordre croissant ; le résultat de l'exemple est 1, 2, 3, 4.

En simulant le fonctionnement de la machine, on peut trouver la bonne réponse et exclure toutes les autres possibilités.



Lors de la première étape, la machine fait deux comparaisons d'une paire nombres nombres chacune et le redirige d'après leur taille. Lors de la deuxième étape, elle compare deux nombres à gauche et deux nombres à droite et détermine ainsi le plus petit et le plus grand nombre des quatre. Finalement, une dernière comparaison des deux nombres du milieu permet de compléter le tri par ordre croissant.

C'est de l'informatique !

En informatique, la machine à nombres de cet exercice du castor est connue sous le nom de *réseau de tri*. Un réseau de tri est composé d'une série de composants identiques, les *comparateurs*. Chaque comparateur reçoit deux valeurs numériques sur deux *fils* et les compare. Il les dirige ensuite sur deux fils sortants : la valeur la plus petite d'un côté et la plus grande de l'autre. Un réseau de tri peut être représenté verticalement, avec les petites valeurs à gauche et les grandes à droite, ou horizontalement, avec les petites valeurs en haut et les grandes en bas.

N'importe quelle suite de nombres peut être triée en combinant assez de comparateurs. La machine de cet exercice nous montre que quatre nombres peuvent être triés avec cinq comparateurs. Il faut neuf comparateurs pour trier cinq nombres et douze comparateurs pour en trier six.

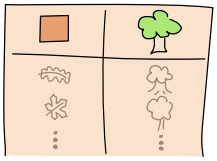
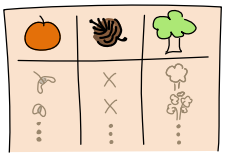
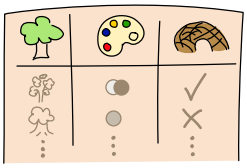
Les réseaux de tri sont pratiques, car les comparateurs sont de simples pièces électroniques peu chères et permettent de fabriquer des réseaux de tri facilement. En plus, plusieurs comparateurs peuvent travailler en même temps, ce qui accélère le tri : de manière générale, le nombre d'étapes non parallèles est plus petit que le nombre de nombres à trier. Un désavantage des réseaux de tri est qu'ils sont conçus pour trier des suites de nombres de longueur fixe.

Mots clés et sites web

- Réseau de tri : `https://fr.wikipedia.org/wiki/Réseau_de_tri`
- Parallélisme : `https://fr.wikipedia.org/wiki/Parallélisme_(informatique)`

9. De la feuille à la hutte

Étienne et ses amis aiment se promener. Pendant leurs promenades, ils rassemblent des informations sur les arbres qu'ils rencontrent et les notent dans de longs tableaux.

Tableau	Description
	**Serge** rassemble des informations sur la forme des feuilles et l'espèce d'arbre qui leur correspond.
	**Quirina** rassemble des informations sur les fruits des arbres, s'ils viennent de conifères et l'espèce d'arbre sur laquelle ils poussent.
	**Ladina** rassemble des informations sur l'espèce des arbres, la couleur de leur bois et s'ils peuvent être utilisés pour construire des huttes de castor.

Étienne a trouvé une feuille dans la forêt et en connaît la forme. Il aimerait savoir si l'espèce d'arbre correspondante peut être utilisée pour construire des huttes de castor.

À qui et dans quel ordre Étienne doit-il poser des questions pour le savoir ?

A) Seulement à Ladina.

B) D'abord à Serge, puis à Quirina.

C) D'abord à Serge, puis à Ladina.

D) D'abord à Quirina, puis à Serge, puis à Ladina.

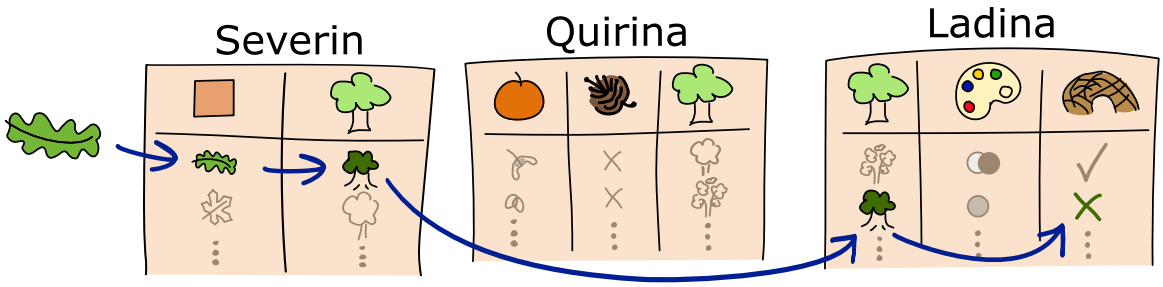
Solution

La bonne réponse est C : d'abord à Serge, puis à Ladina.

Il n'y a que le tableau de Ladina qui dit si une sorte de bois peut être utilisée pour construire des huttes de castor. Mais si Étienne n'a d'informations que sur la feuille, il ne peut pas choisir une ligne dans le tableau de Ladina. Pour cela, il a besoin de connaître l'espèce de l'arbre ou la couleur de son bois. La réponse A est donc fausse : ça ne suffit pas de demander seulement à Ladina.

Le tableau de Quirina ne contient pas d'informations sur les feuilles ni sur l'utilisation du bois pour des huttes. Son tableau n'est pas utile à Étienne, les réponses B et D sont donc fausses.

Par contre, le tableau de Serge contient des informations sur les feuilles. Comme Étienne connaît la forme de la feuille, il peut commencer par trouver la ligne correspondante dans le tableau de Serge et apprendre de quelle espèce d'arbre vient la feuille. Il peut ensuite utiliser cette information pour trouver la bonne ligne dans le tableau de Ladina et déterminer si le bois peut être utilisé pour construire des huttes de castor. Par exemple, si Étienne détermine qu'il s'agit d'une feuille de chêne grâce à la forme de la feuille et au tableau de Serge, il peut regarder dans le tableau de Ladina si le chêne peut être utilisé pour construire des huttes.

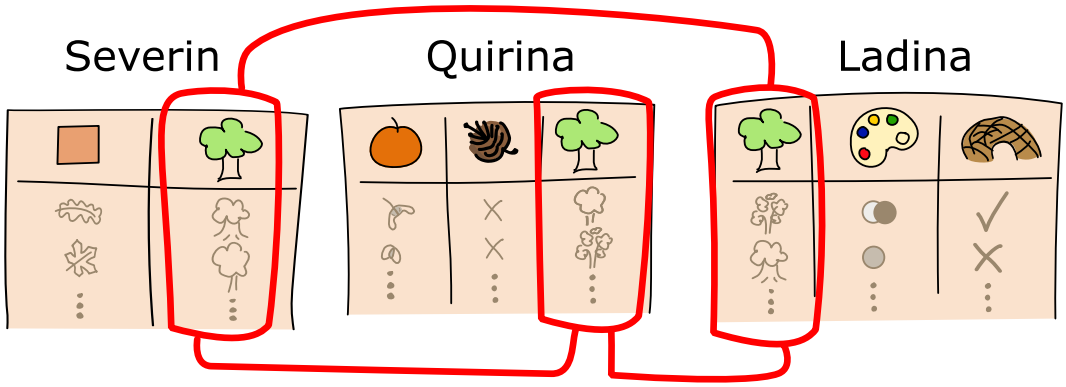


C'est de l'informatique !

Cet exercice du castor illustre les concepts fondamentaux des *bases de données relationnelles*. Elles sont très souvent utilisées en informatique pour la gestion de grandes (et moins grandes) quantités de données. Les bases de données relationnelles sont composées de tableaux de données – comme les tableaux des amis d'Étienne – qui sont aussi appelés *relations* ou *tables*. Chaque colonne d'un tableau est un *attribut* et chaque ligne un *enregistrement* ou *nuplet*. Différents tableaux sont reliés par un attribut commun – ici, l'espèce d'arbre. On appelle cet attribut qui sert de référence entre les tableaux *clé étrangère*.

La question d'Étienne (si le bois de l'arbre correspondant à une certaine feuille peut être utilisé pour construire des huttes) est appelée une *requête* dans le cadre de bases de données informatiques. Cette requête nécessite de combiner plusieurs tableaux pour obtenir l'information désirée. Cette opération reliant les tableaux combine les lignes de plusieurs tableaux à l'aide d'une clé. Ainsi, les données enregistrées dans plusieurs tableaux (ici, l'espèce d'arbre, la forme des feuilles et l'utilisation

du bois) peuvent être regroupées dans un seul tableau pour répondre à une requête. Dans notre cas, l'espèce de l'arbre peut être utilisée pour relier les tableaux des amis :



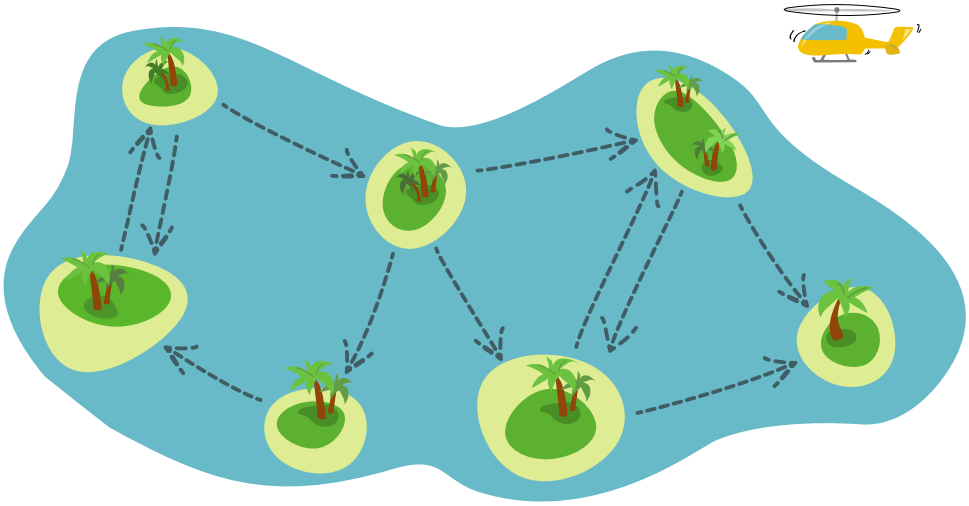
Les bases de données relationnelles sont si importantes qu'il existe un langage spécial pour les requêtes et autres opérations sur les bases de données, le *SQL (Structured Query Language)*.

Mots clés et sites web

- Base de données relationnelle : `https://fr.wikipedia.org/wiki/Base_de_données_relationnelle`
- Clé : `https://fr.wikipedia.org/wiki/Clé_(structure_de_données)`
- SQL : `https://fr.wikipedia.org/wiki/SQL`
- Attribut : `https://fr.wikipedia.org/wiki/Base_de_données#Attribut`
- Uplet : `https://fr.wikipedia.org/wiki/Uplet`

10. Archipel des castors

L'archipel des castors compte 7 îles. On peut se déplacer entre les îles en bateau, mais seulement dans le sens des flèches.



Une équipe de chercheurs aimerait étudier la vie animale sur chacune des 7 îles. Chaque expédition de l'équipe se passe de la façon suivante :

1. L'équipe se rend en hélicoptère sur n'importe quelle île ;
2. Elle prend ensuite le bateau pour se rendre sur d'autres îles,
3. Elle retourne sur l'île où est resté l'hélicoptère pour rentrer.

L'équipe constate qu'une seule expédition ne suffit pas à visiter toutes les îles.

Quel est le nombre minimum d'expéditions que l'équipe doit faire pour visiter toutes les îles ?

A) 2 expéditions
B) 3 expéditions
C) 4 expéditions
D) 5 expéditions
E) 6 expéditions
F) 7 expéditions

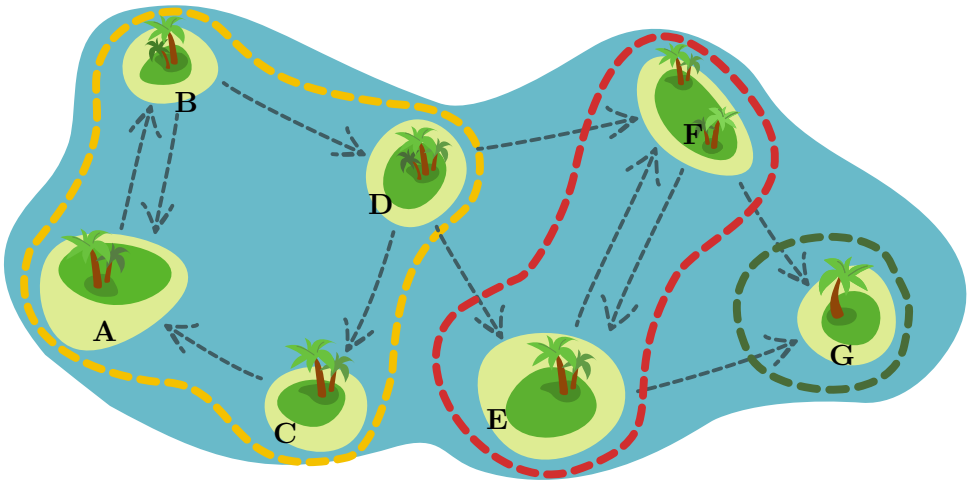
Solution

La bonne réponse est 3.

Pour faire un minimum d'expéditions, l'équipe doit visiter le plus d'îles possible à chaque expédition. L'équipe doit aussi pouvoir revenir à l'île de départ pour retrouver l'hélicoptère. Lors de chaque expédition, l'équipe ne peut donc visiter que des îles à partir desquelles elles peuvent revenir à l'île de départ.

Nous répartissons les îles en plusieurs groupes. Chaque groupe compte autant d'îles que possible, mais seulement des îles à partir desquelles toutes les autres îles du groupe peuvent être atteintes en bateau, directement ou indirectement. L'équipe peut alors choisir n'importe quelle île de départ dans le groupe pour une expédition et visiter toutes les autres îles du groupe. Ils ne peuvent cependant pas en visiter plus, car si l'équipe quitte le groupe d'îles, elle ne peut plus y revenir et donc ne peut pas rejoindre l'hélicoptère. L'équipe doit donc faire autant d'expéditions qu'il y a de groupes d'îles.

On peut déterminer un groupe comme cela : on choisit n'importe quelle île n'appartenant à aucun groupe comme première île d'un nouveau groupe. On assigne ensuite à ce groupe toutes les autres îles que l'on peut atteindre depuis la première île, et desquelles on peut aussi atteindre la première île. L'image montre que les sept îles forment trois groupes : {A,B,C,D}, {E,F} et {G}. L'équipe de recherche doit donc faire trois expéditions pour visiter toutes les îles.



Mais pourquoi une seule expédition ne suffit-elle pas ? On peut atteindre toutes les autres îles depuis certaines îles, comme l'île C par exemple ! Mais en faisant cela, on quitterait le groupe de la première île et ne pourrait pas y revenir, et donc pas retrouver l'hélicoptère.

C'est de l'informatique !

Les îles sont partiellement reliées les unes aux autres par des bateaux. On peut représenter les îles et les lignes de bateau par un *graphe*. Un graphe est une structure qui modélise les relations entre des objets – comme les lignes de bateau entre les îles dans cet exercice du castor. Les îles y sont les *nœuds* du graphe et les lignes de bateau les *arêtes orientées*.

Le concept de groupes utilisé ici peut aussi être appliqué aux graphes : un groupe est un ensemble de nœuds dans lequel chaque paire de nœuds est reliée par des arêtes, directement ou indirectement. De tels groupes sont appelés *composantes fortement connexes* (CFC) dans les graphes. Les graphes et les composantes fortement connexes jouent un rôle important dans beaucoup d'applications informatiques. Voici quelques exemples :

- Sur internet, les sites contenant tous des liens les uns vers les autres sont des CFC.
- Sur les réseaux sociaux, les CFC sont des « bulles » d'utilisateurs se suivant mutuellement.
- Dans les réseaux de transports publics, les CFC forment des zones dans lesquelles tous les arrêts sont reliés.

Il existe des algorithmes informatiques pour déterminer efficacement les CFC d'un graphe. Le plus connu est l'*algorithme de Tarjan*. Robert Tarjan est un informaticien américain qui a contribué au développement de nombreux algorithmes, plusieurs d'entre eux portant son nom. Il a reçu le plus prestigieux prix en informatique, le Prix Turing, à l'âge de 38 ans.

Mots clés et sites web

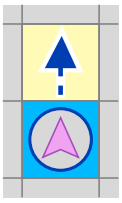
- Graphe orienté : `https://fr.wikipedia.org/wiki/Graphe_orienté`
- Composante fortement connexe : `https://fr.wikipedia.org/wiki/Composante_fortement_connexe`
- Algorithme de Tarjan : `https://fr.wikipedia.org/wiki/Algorithme_de_Tarjan`

11. Lefty II

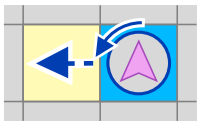
Lefty est un robot qui se déplace sur une grille à cases carrées. Il peut y avoir des murs rouges entre les cases. Lefty doit atteindre le but vert.

Lefty peut se déplacer d'exactement deux manières. Il peut :

avancer d'une case

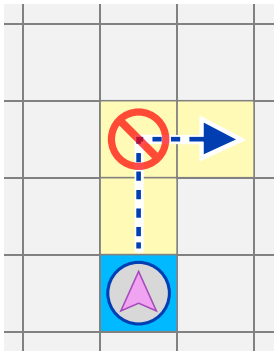


se tourner vers la gauche
puis avancer d'une case

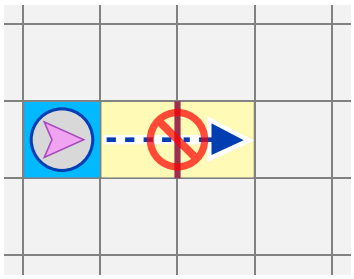


Lefty ne peut pas tout faire. Par exemple, il ne peut…

… **pas** simplement tourner à droite

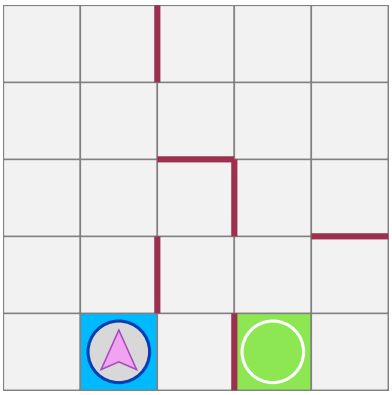


… **pas** traverser des murs



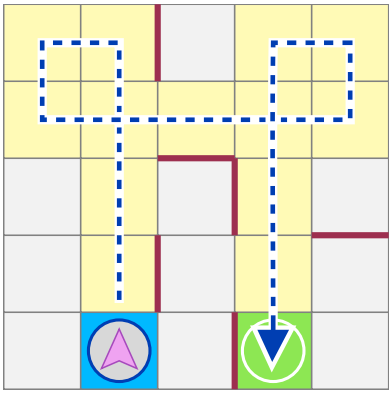
Par quelles cases Lefty ***doit****-il passer pour atteindre le but ?*

Choisis ***le moins de cases possible****.*



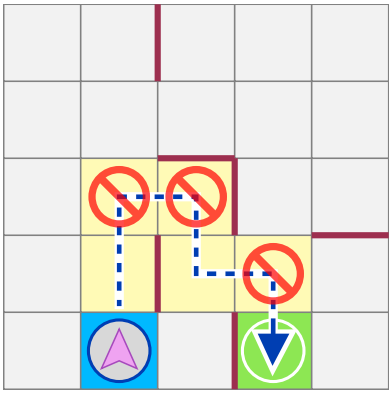
Solution

Voici la bonne réponse :



Lefty atteint le but lorsqu'il passe par ces cases et il ne se déplace que des deux manières dont il est capable.

Il n'y a pas d'autre chemin passant par le même nombre ou moins de cases permettant à Lefty d'atteindre le but. Il ne peut pas prendre le chemin direct sans devoir tourner à droite.

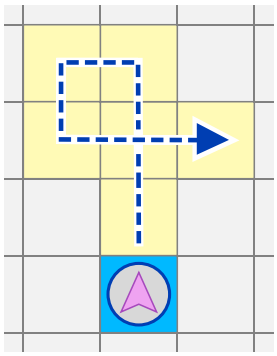


C'est de l'informatique !

Les fonctions de Lefty sont très limitées. Si seulement il pouvait se déplacer d'autres manières ! S'il pouvait tourner à droite ou même grimper par-dessus les murs, sa vie sur la grille serait beaucoup plus simple. Lefty serait beaucoup plus sûr de lui s'il avait un *jeu d'instructions* plus complexe. Les choses dont un robot est capable sont déterminées par des instructions dans le programme qui le dirige.

Mais est-ce vraiment nécessaire ? Lefty peut par exemple tourner à droite en tournant à gauche trois fois de suite. Il faudrait juste éliminer la règle obligeant Lefty à avancer après chaque rotation vers la gauche. Il pourrait alors se tourner et avancer dans toutes les directions. Et au lieu de grimper par-dessus les murs, il peut les contourner. Cela signifie qu'un jeu d'instructions réduit peut tout à faire être suffisant au fonctionnement d'un robot. Pour implémenter des comportements plus complexes et/ou plus rares, on peut développer des *sous-programmes* qui combinent des instructions

simples en une nouvelle instruction. Par exemple, un sous-programme pourrait être utilisé pour décrire comment Lefty peut tourner à droite (ce qui est utilisé deux fois dans la solution ci-dessus) :



En informatique,les deux approches suivantes sont les plus utilisées pour élaborer les jeux d'instructions d'un *processeur* : *CISC* (« Complex Instruction Set Computer », processeur à jeu d'instructions étendu) et *RISC* (« Reduced Instruction Set Computer », processeur à jeu d'instructions réduit) – comme celui de Lefty dans cet exercice du castor. Un CISC a en général beaucoup d'instructions différentes qui peuvent être très puissantes (comme grimper par-dessus un mur), mais qui sont rarement utilisées. Un RISC a peu d'instructions plus simples qui sont souvent utilisées.

Ces deux types d'*architecture* ont chacun leurs avantages et inconvénients. Les processeurs des grandes marques sont soit des processeurs CISC, soit des processeurs RISC, bien que les processeurs RISC soient plus souvent utilisés récemment.

Mots clés et sites web

- Processeur : `https://fr.wikipedia.org/wiki/Processeur`
- jeu d'instructions : `https://fr.wikipedia.org/wiki/Jeu_d'instructions`
- CISC : `https://fr.wikipedia.org/wiki/Microprocesseur_à_jeu_d'instructions_étendu`
- RISC : `https://fr.wikipedia.org/wiki/Processeur_à_jeu_d'instructions_réduit`

12. Labyrinthe

Dans l'un des jeux vidéos de Momo, un robot peut se déplacer sur des cases . Lorsqu'il se trouve devant un obstacle ou un mur, il ne peut pas aller plus loin et doit changer de direction.

Les obstacles et le but peuvent changer de position à chaque niveau du jeu. Un niveau est réussi lorsque le robot atteint le but.

Momo peut diriger le robot en utilisant des programmes. Il peut utiliser les 4 commandes suivantes pour écrire ses programmes :

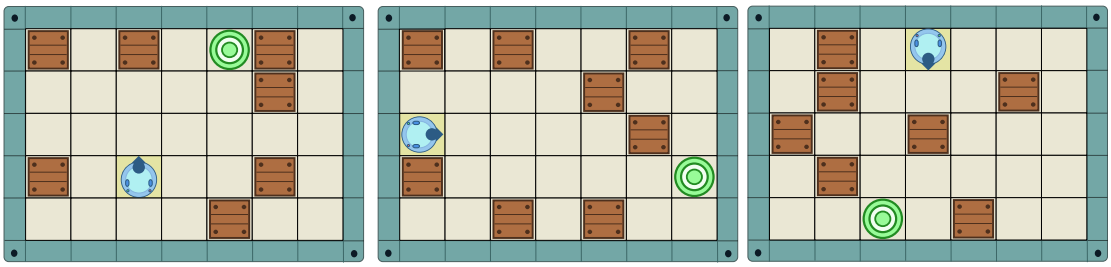
Avance d'une case.

Avance jusqu'à ce que tu ne puisses plus aller plus loin.

Tourne de 90 degrés dans le sens inverse des aiguilles d'une montre.

Tourne de 90 degrés dans le sens des aiguilles d'une montre.

Momo connaît déjà les 3 prochains niveaux. Il aimerait écrire un programme avec le moins de commandes possible lui permettant de réussir les 3 niveaux.



Écris un programme avec lequel le robot de Momo réussit les 3 niveaux.

Avance d'une case.

Avance jusqu'à ce que tu ne puisses plus aller plus loin.

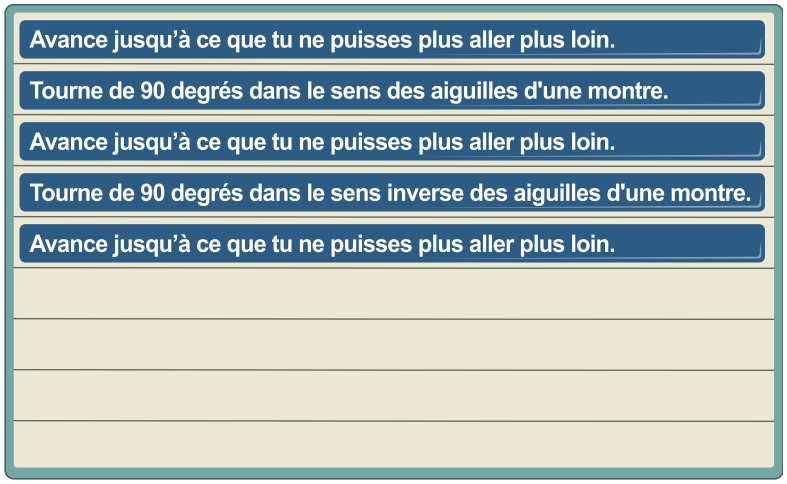
Tourne de 90 degrés dans le sens inverse des aiguilles d'une montre.

Tourne de 90 degrés dans le sens des aiguilles d'une montre.

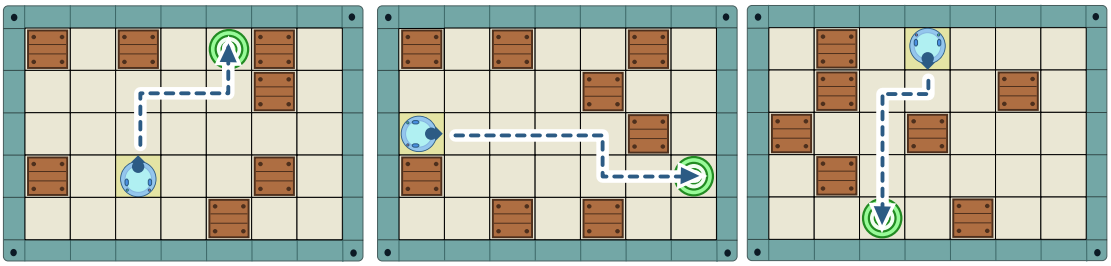
Solution

Voici la bonne réponse :

Plusieurs programmes permettent de réussir chaque niveau, mais un seul permet de les réussir les trois :



Les flèches montrent quels chemins le programme fait prendre au robot dans chaque niveau :



Dans le niveau 1 (sur l'image de gauche), le but se trouve à droite et au-dessus du robot. La commande « Avance jusqu'à ce que tu ne puisses plus aller plus loin » doit donc être utilisée au moins deux fois : une fois pour arriver à droite, et une fois pour arriver en haut. Le robot regarde vers le haut au début et à la fin ; il doit donc d'abord se tourner dans le sens des aiguilles d'une montre pour tourner à droite, puis dans le sens inverse pour se retourner vers le haut. Il faut donc au moins 4 commandes pour réussir le niveau 1.

Dans le niveau 2 (sur l'image du centre), le robot doit contourner un obstacle au milieu sur son chemin vers la droite et a besoin de la commande « Avance jusqu'à ce que tu ne puisses plus aller plus loin » une fois de plus. Il faut donc un programme avec au moins cinq commandes pour réussir le niveau 2.

Ce programme à cinq commandes réussit aussi les niveaux 1 et 3 (sur l'image de droite). Il n'existe pas de programme plus court permettant de réussir les trois niveaux.

C'est de l'informatique !

Les 4 commandes pour le robot de cet exercice du castor forment un langage de programmation simple. Le robot peut être dirigé par une suite de ces commandes. En informatique, on appelle un telle suite une *séquence de commandes*. La séquence est la structure de contrôle fondamentale la plus simple de la *programmation structurée*.

La commande « Avance d'une case » n'est pas utile pour résoudre cet exercice. À la place, le programme utilise la commande « Avance jusqu'à ce que tu ne puisses plus aller plus loin ». Cette commande permet de répéter le pas en avant jusqu'à ce que le robot atteigne un obstacle ou le bord du plan de jeu. La répétition, ou *boucle*, est une autre structure de contrôle fondamentale en programmation structurée. Il peut y avoir un nombre fixe de répétitions (« Avance de cinq cases »), mais les boucles les plus utilisées sont celles avec condition d'arrêt, comme « Avance jusqu'à ce tu ne puisses plus aller plus loin ». Comme cette commande permet au robot de parcourir des chemins de longueurs différentes, il peut réussir les trois niveaux.

Tu peux certainement imaginer des niveaux que le robot ne peut pas réussir à l'aide de ce programme, par exemple s'il doit changer de direction plusieurs fois pour atteindre son but. Les informaticiennes et informaticiens essaient de créer des *algorithmes* qui ne soient pas spécifiques à certains cas, mais fonctionnent systématiquement pour tous les cas imaginables. Pour cela, ils regardent aussi si des algorithmes existent déjà pour résoudre des problèmes similaires. Les niveaux du jeu de Momo ressemblent à des labyrinthes, et il existe des algorithmes généraux pour résoudre les labyrinthes. Ils fonctionnent pour tous les labyrinthes, même si leurs solutions sont parfois des chemins plus compliqués que nécessaire.

Mots clés et sites web

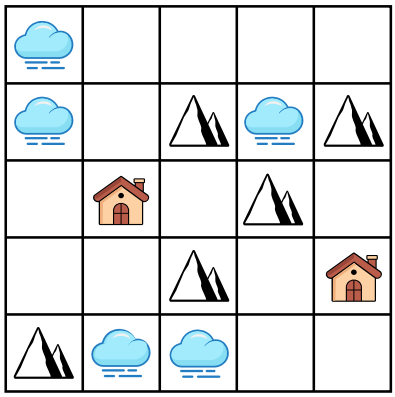
- Structure de contrôle : `https://fr.wikipedia.org/wiki/Structure_de_contrôle`
- Programmation structurée : `https://fr.wikipedia.org/wiki/Programmation_structurée`
- Résolution de labyrinthe : `https://fr.wikipedia.org/wiki/Résolution_de_labyrinthe`

13. Jour de brouillard

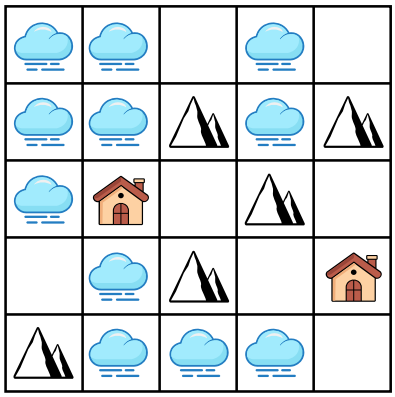
Aujourd'hui, il y a du brouillard au pays montagneux. Le brouillard s'étend d'heure en heure.

À l'aube, le brouillard ne couvre que quelques régions. Durant chaque heure qui passe, le brouillard s'étend d'une région couverte à toutes ses régions voisines : à gauche, à droite, en haut et en bas. Le brouillard couvre aussi des maisons. Seules les montagnes ne peuvent pas être couvertes par le brouillard.

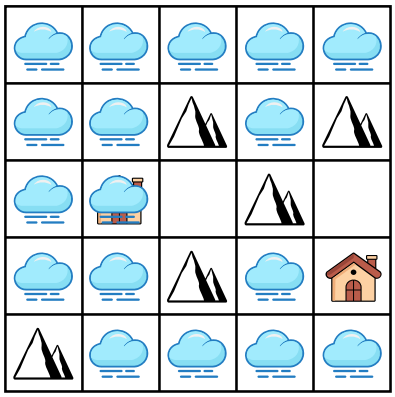
Voici un exemple :



À l'aube

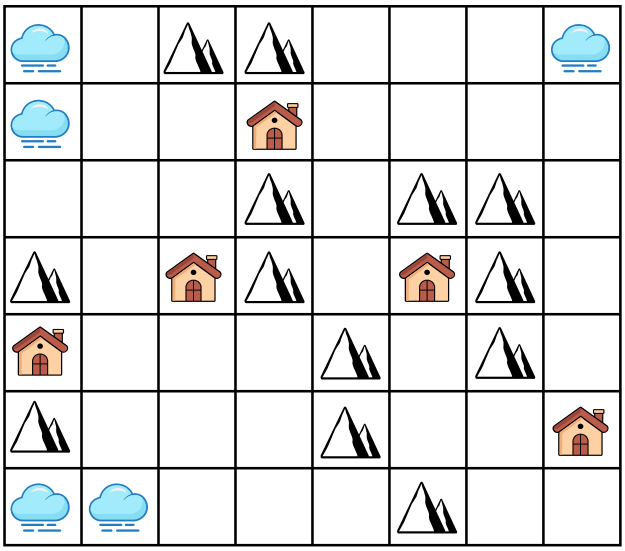


1 heure plus tard



2 heures plus tard

*Quelle maison est la **dernière** à être couverte par le brouillard ?*

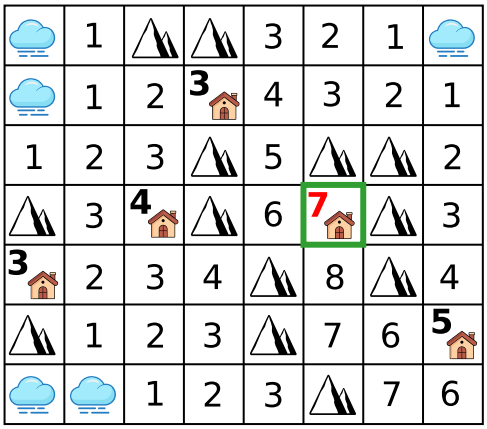


Solution

La bonne réponse est la maison dans la quatrième ligne et la sixième colonne.

Le brouillard s'étend sur le pays comme de l'eau qui coule : d'abord sur les régions voisines des régions couvertes, puis sur les régions voisines des voisines, et ainsi de suite. Il suffit donc d'observer le pays jusqu'à ce que toutes les maisons sauf une soient couvertes pour avoir la bonne réponse.

La dernière maison a être couverte est encadrée en vert sur l'image ci-dessous. L'image montre également combien d'heures il a fallu au brouillard pour couvrir chaque région. On voit que la maison encadrée a le plus grand nombre de toutes les maisons, et qu'elle est la seule maison avec ce nombre. Il n'y a donc qu'une bonne réponse possible.



On peut aussi aborder le problème différemment : la dernière maison à être couverte est celle qui se trouvait le plus loin du brouillard au départ. On peut mesurer cette distance pour chacune des maisons pour trouver la bonne réponse. La distance d'une maison au brouillard est mesurée le long de chemin du brouillard en comptant les régions voisines non diagonales et en contournant les montagnes. Cette méthode prend plus de temps que la précédente, car il faut faire n x m calculs pour n maisons et m régions couvertes initialement.

C'est intéressant de noter qu'un regard humain peut facilement se tromper : au premier coup d'œil, c'est la maison en bas à droite qui semble être la plus éloignée de toutes les régions couvertes. Cependant, comme aucune montagne ne se trouve entre elle et le brouillard, est est plus vite couverte que la maison de la solution.

C'est de l'informatique !

Dans cet exercice du castor, le brouillard couvre les régions du pays qui peuvent être atteintes par les brouillards depuis au moins une des régions initialement couvertes les unes après les autres. Un ensemble de régions pouvant être atteintes depuis une seule région couverte peut être qualifié de *connexe*. Dans le pays de cet exercice, toutes les régions forment un seul ensemble connexe. L'ajout d'une seule montagne dans la deuxième ligne, cinquième colonne diviserait le pays en deux ensembles connexes.

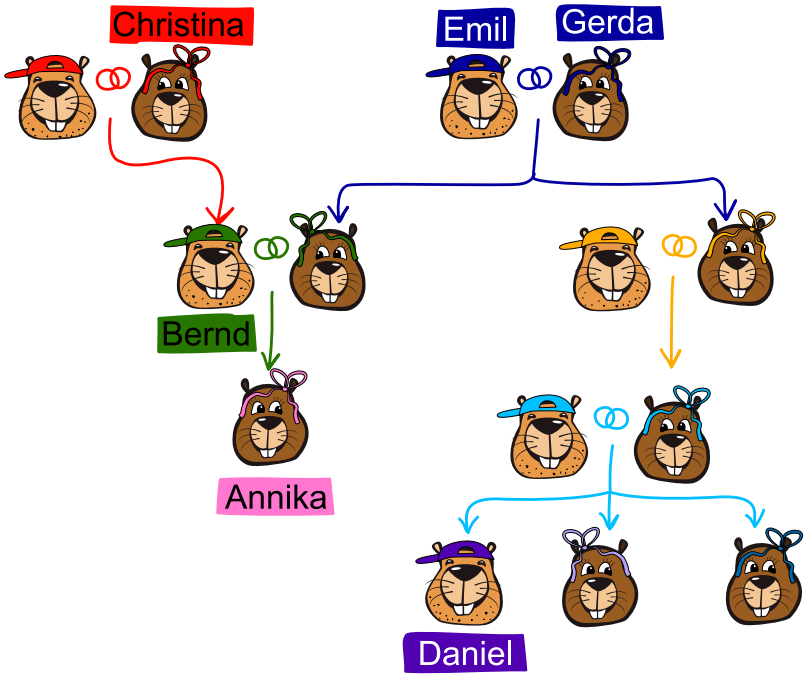
Les ensembles connexes sont intéressants pour différents domaines de l'informatique. Une zone de couleur unie sur une image (numérique) est un ensemble de pixels connexe de la même couleur ; on peut le déterminer à l'aide d'un *algorithme de remplissage par diffusion* qui fonctionne de manière similaire au brouillard de cet exercice du castor. Un groupe de personnes parmi lesquelles chaque personne est amie avec au moins une autre personne du groupe est aussi un ensemble connexe si l'on considère l'amitié comme une relation de voisinage. De la même manière, les « bulles » des réseaux sociaux peuvent être considérées comme des ensembles connexes. Ici aussi, il existe des algorithmes informatiques permettant de trouver ces ensembles connexes, comme le *parcours en profondeur* ou le *parcours en largeur*.

Mots clés et sites web

- Parcours en profondeur :
  `https://fr.wikipedia.org/wiki/Algorithme_de_parcours_en_profondeur`
- Graphe connexe : `https://fr.wikipedia.org/wiki/Graphe_connexe`
- Algorithme de remplissage par diffusion :
  `https://fr.wikipedia.org/wiki/Algorithme_de_remplissage_par_diffusion`

14. Arbre généalogique

Les castors Annika et Daniel veulent connaître leur lien de parenté. Annika a un arbre généalogique de leur famille commune. Les castors mâles y portent un chapeau et les castors femelles un ruban.



Annika utilise une abréviation :

- `père(X)` signifie « père du castor X »
- `mère(X)` signifie « mère du castor X »

Bernd est le père d'Annika, et Christina est la mère de Bernd. Annika décrit cela avec des équations :

- `père(Annika) = Bernd`
- `mère(Bernd) = Christina`

Annika peut aussi décrire son lien de parenté à Christina à l'aide d'une seule équation :

- `mère(père(Annika)) = Christina`, ce qui signifie « Christina est la mère du père d'Annika ».

Elle aimerait maintenant une équation qui décrive son lien de parenté à Daniel.

Complète l'équation suivante pour qu'elle décrive le lien de parenté entre Daniel et Annika.

père mère

père (mère (Annika)) = (((Daniel)))

Solution

Voici la bonne réponse :

père (mère (Annika)) = père (mère (mère (Daniel)))

Nous commençons par le côté droit de l'équation et voyons qu'Emil est le père de la mère d'Annika, donc `père(mère(Annika)) = Emil`. Pour remplir les trous de l'autre côté de l'équation, nous devons déterminer le lien de parenté entre Emil et Daniel. Pour cela, nous observons l'arbre généalogique et allons de Daniel à Emil pas à pas :

1. La mère de Daniel, `mère(Daniel)`, est apparentée à Emil ; pas le père de Daniel.
2. La mère de la mère de Daniel, donc la grand-mère de Daniel, `mère(mère(Daniel))`, est apparentée à Emil, car…
3. … Emil est le père de cette grand-mère : `Emil = père(mère(mère(Daniel)))`.

C'est de l'informatique !

Annika utilise ses abréviations `père(X)` et `mère(X)` pour les liens de parent à enfant dans son arbre généalogique, comme des *fonctions* mathématiques qui prennent comme *argument* (ici, une personne dans l'arbre généalogique) une *valeur* (une autre personne). Les fonctions sont aussi utilisées en informatique ; ce sont des modules du code des programmes avec lesquels on peut directement implémenter des fonctions mathématiques. Une telle fonction est appelée avec un ou plusieurs arguments (aussi appelés paramètres) et retourne une valeur comme solution après exécution.

Cet exercice du castor montre comment des appels de fonction peuvent être combinés pour effectuer des calculs complexes. Dans une telle *composition de fonction*, les résultats de l'appel de la fonction intérieure sont utilisés comme arguments pour l'appel de la fonction extérieure. L'appel d'une fonction composée est effectué de l'intérieur vers l'extérieur : par exemple, pour effectuer `père(mère(mère(Daniel)))`, on commence par déterminer qui est la mère de Daniel, puis la mère de sa mère, et enfin le père de la mère de sa mère. La composition de fonctions est possible dans la plupart des langages de programmation, mais est spécialement importante en *programmation fonctionnelle*.

Mots clés et sites web

- Fonction : `https://fr.wikipedia.org/wiki/Routine_(informatique)`
- Composition de fonctions : `https://fr.wikipedia.org/wiki/Composition_de_fonctions`

15. Message secret

Une image secrète composée de pixels noirs et blancs doit être transmise de manière sûre. Pour cela, le service de messagerie crée deux images composées de pixels foncés et clairs sur des feuilles transparentes. L'image secrète n'est révélée que lorsque les deux feuilles transparentes sont superposées.

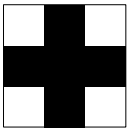
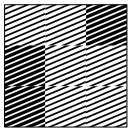


Image secrète


Transparent 1


Transparent 2


Transparents 1 et 2 superposés

Les images des les feuilles transparentes sont créées de la manière suivante : tout d'abord, un motif aléatoire de pixels clairs et foncés est imprimé sur la première feuille transparente. La couleur des pixels de la deuxième feuille transparente est déterminée par la règle suivante en fonction de la couleur des mêmes pixels sur l'image originale et sur la première feuille transparente :

- Si le pixel de l'image originale est noir, les pixels sur le transparent 1 et le transparent 2 doivent être de couleurs différentes (l'un clair et l'autre foncé).
- Si le pixel de l'image originale est blanc, les pixels sur le transparent 1 et le transparent 2 doivent être la même couleur (les deux clairs ou les deux foncés).

Le premier transparent a déjà été imprimé pour l'image ci-dessous. Détermine la couleur des pixels du deuxième transparent.

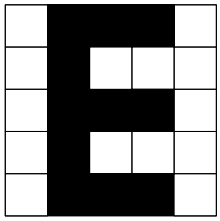
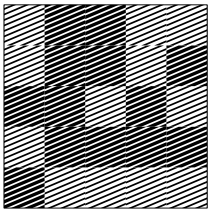
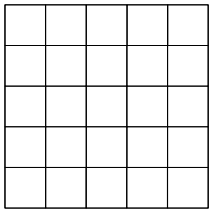


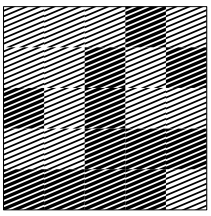
Image secrète


Transparent 1


Transparent 2

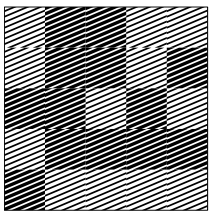
Solution

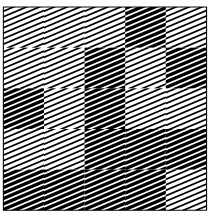
Voici la bonne réponse :

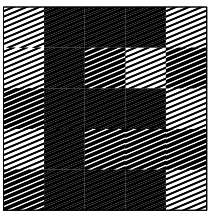


Sur cette image du transparent 2, chaque pixel a été imprimé en suivant la règle décrite plus haut – en fonction de l'image secrète et du transparent 1. L'image du transparent 2 n'est différente du transparent 1 qu'aux endroits où l'image secrète a un pixel noir.

Tu peux voir ici comment la superposition du transparent 1 et de ce transparent 2 révèle l'image secrète :


Transparent 1


Transparent 2


Transparents 1 et 2 superposés

C'est de l'informatique !

Le service de messagerie utilise un *procédé cryptographique* basé sur une image – ce qui est appelé *cryptographie visuelle*. La technique de cet exercice du castor a été développée en 1994 par les scientifiques israéliens Moni Naor et Adi Shamir. Le procédé est très sûr pour un unique transparent : comme il est basé sur une matrice de pixels aléatoire, aucune information ne peut être gagnée à partir d'un seul transparent, même à l'aide d'outils informatiques. Le déchiffrement n'est possible qu'avec les deux transparents, et est alors très simple.

Pour la transmission, un transparent 1 aléatoire mais fixe pourrait être enregistré comme *clé* chez l'expéditeur et chez le destinataire : il ne faudrait plus que générer et envoyer le deuxième transparent pour chaque image secrète. Cependant, le procédé n'est plus si sûr si la clé, donc le transparent 1, est réutilisée. C'est un problème général des méthodes de cryptographie qui fonctionnent d'après le principe du *masque jetable*. Dans ces méthodes, la clé doit être au moins aussi longue que le message secret et doit être générée aléatoirement. Le message chiffré est généré à l'aide d'une combinaison réversible des signes du message secret et de la clé. En informatique, l'opération *XOR* est souvent utilisée pour les messages en bits. La combinaison des pixels clairs et sombres de cet exercice correspond exactement à cette opération.

Mots clés et sites web

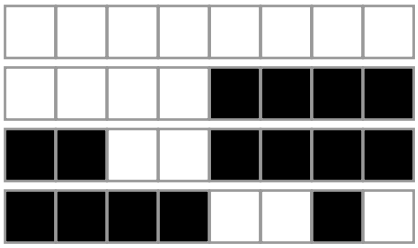
- Cryptographie visuelle : `https://fr.wikipedia.org/wiki/Cryptographie_visuelle`
- Masque jetable : `https://fr.wikipedia.org/wiki/Masque_jetable`
- XOR : `https://fr.wikipedia.org/wiki/Fonction_OU_exclusif`

16. Noir et blanc

Sarah aimerait représenter des suites de carrés noirs et blancs avec des lettres. Pour cela, elle applique l'algorithme suivant à une suite de lettres :

- Si tous les carrés de la suite sont blancs, écris `B`.
- Si tous les carrés de la suite sont noirs, écris `N`.
- Si la suite contient des carrés noirs et des carrés blancs, écris `x` et :
  - — applique l'algorithme à la moitié gauche de la suite, puis
  - — applique l'algorithme à la moitié droite de la suite.

Voici les représentations données par l'algorithme pour quelques suites de carrés :



`B`

`xBN`

`xxNBN`

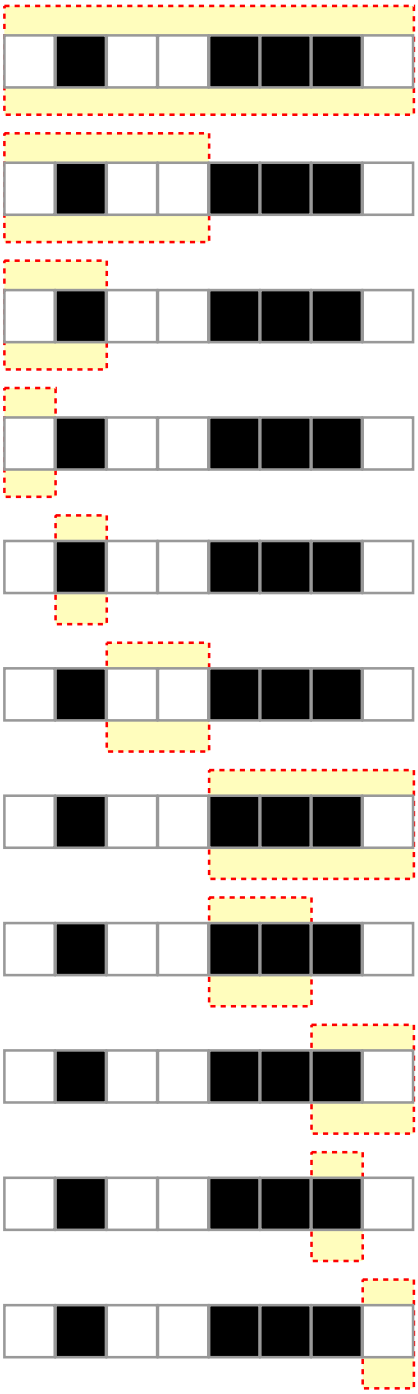
`xNxBxNB`

Quelle est la représentation donnée par l'algorithme de Sarah pour la suite de carrés suivante ?

Solution

Voici la bonne réponse : `xxxBNBxNxNB`.

Nous appliquons l'algorithme à la suite de carrés et déterminons la réponse pas à pas. La suite de carrés traitée par l'algorithme à chaque pas est encadrée en jaune sur les images ci-dessous.



`x` - La suite contient des carrés noirs et des carrés blancs. L'algorithme écrit `x` et s'applique lui-même à la moitié de gauche de la suite.

`xx` - La suite contient des carrés noirs et des carrés blancs. L'algorithme écrit `x` et s'applique lui-même à la moitié de gauche de la suite.

`xxx` - La suite contient des carrés noirs et des carrés blancs. L'algorithme écrit `x` et s'applique lui-même à la moitié de gauche de la suite.

`xxxB` – Tous les carrés de la suite sont blancs. L'algorithme écrit `B` et a fini de traiter cette suite. Il traite maintenant la partie droite de la suite précédente.

`xxxBN` - Tous les carrés de la suite sont noirs. L'algorithme écrit `N` et a fini de traiter cette suite. Il traite maintenant la partie droite de la suite précédente.

`xxxBNB` - Tous les carrés de la suite sont blancs. L'algorithme écrit `B` et a fini de traiter la moitié gauche de la suite complète. Il traite maintenant la moitié droite.

`xxxBNBx` - La suite contient des carrés noirs et des carrés blancs. L'algorithme écrit `x` et s'applique lui-même à la moitié de gauche de la suite.

`xxxBNBxN` - Tous les carrés de la suite sont noirs. L'algorithme écrit `N` et a fini de traiter cette suite. Il traite maintenant la partie droite de la suite précédente.

`xxxBNBxNx` - La suite contient des carrés noirs et des carrés blancs. L'algorithme écrit `x` et s'applique maintenant lui-même à la moitié de gauche de la suite.

`xxxBNBxNxN` - Tous les carrés de la suite sont noirs. L'algorithme écrit `N` et a fini de traiter cette suite. Il traite maintenant la partie droite de la suite précédente.

`xxxBNBxNxNB` - Tous les carrés de la suite sont blancs. L'algorithme écrit `B` et a fini de traiter la moitié droite de la suite complète. Il a terminé.

C'est de l'informatique !

L'algorithme de Sarah a une propriété très particulière : lorsque la suite de carrés contient des carrés noirs et des carrés blancs, il s'applique lui-même à la moitié gauche, puis à la moitié droite de la suite. La tâche consistant à décrire la suite de carrés à l'aide de lettres est ainsi divisée en deux tâches plus petites. Cette méthode est utile lorsque les tâches partielles sont plus faciles à réaliser que la tâche complète. En informatique, cette méthode s'appelle *diviser pour régner* d'après la méthode de gouvernance antique de l'Empire romain « Divide ut imperes ». Lorsqu'une méthode traite les problèmes partiels de la même manière que le problème complet, donc en s'appliquant elle-même aux problèmes partiels et en les divisant à leur tour en problèmes plus petits, on dit qu'elle est *récursive* – comme l'algorithme de Sarah dans cet exercice. La récursivité est très souvent utilisée en informatique, que ce soit pour trier des données, construire des systèmes de fichiers ou beaucoup d'autres tâches.

L'algorithme de Sarah décrit ou *encode* la suite de carrés avec des lettres. Un encodage doit être réversible, il doit donc exister une méthode permettant de convertir la suite de lettres en la suite de carrés originale. C'est possible sans problème si la description en lettres est complétée pour avoir la même longueur que la suite de carrés. Réfléchis à comment le faire ! Peut-être as-tu besoin d'un algorithme récursif ?

Idéalement, l'encodage fait par l'algorithme de Sarah est beaucoup plus court que la suite de carrés originale. Par exemple, une suite de 1024 carrés blancs est encodée par un unique `B`. L'algorithme de Sarah ne fait donc pas que de l'encodage, mais aussi de la *compression de données* (une représentation des données permettant d'économiser de la place) en suivant un principe similaire à ceux utilisés pour la compression des images comme le JPEG.

Mots clés et sites web

- Diviser pour régner : `https://fr.wikipedia.org/wiki/Diviser_pour_régner_(informatique)`
- Récursivité : `https://fr.wikipedia.org/wiki/Récursivité`
- Compression de données : `https://fr.wikipedia.org/wiki/Compression_de_données`
- Quadtree : `https://fr.wikipedia.org/wiki/Quadtree`

Tâches de programmation

Les tâches qui suivent sont des tâches de programmation et font partie des tâches bonus du concours.

Alors que les tâches de base n'ont aucun prérequis en informatique, ces tâches-ci se résolvent plus facilement en ayant des connaissances en programmation.

Comme la programmation sur papier n'est pas très pratique, un code QR est fourni pour chaque tâche pour la résoudre en ligne de façon interactive.

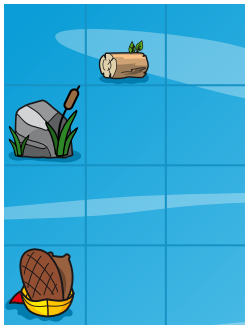
17. Le banc de sable fou

Benno le castor veut ramasser une bûche dans le lac. Benno a découvert que les mouvements de sable dans le lac déplacent toujours le rocher à différents endroits. Écris les instructions pour que Benno puisse ramasser la bûche, peu importe où se trouve le rocher. Clique sur les cercles sous le lac pour voir les différentes positions où le rocher peut se trouver.

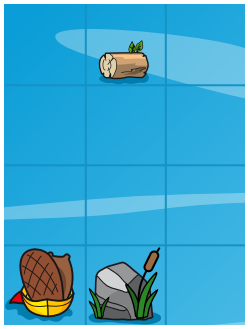
Tu peux utiliser ces instructions :

Instruction	**Effet**
`move()`	Benno avance d'exactement une case dans la direction de son regard
`turnRight()` / `turnLeft()`	Benno se tourne sur place de 90 degrés sur sa droite ou sur sa gauche
`removeLog()`	Benno ramasse la bûche de la case où il se trouve

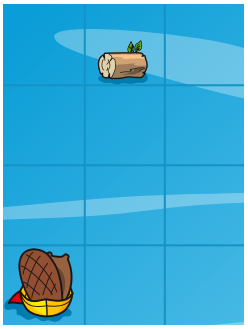
Lac 1



Lac 2



Lac 3



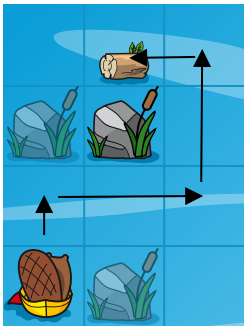
Lac 4

Écris un programme pour que Benno puisse toujours ramasser la bûche.

Solution

La solution correcte est la suivante :

```
move()
turnRight()
move()
move()
turnLeft()
move()
move()
turnLeft()
move()
removeLog()
```



Comme le banc de sable se déplace, il n'est pas possible de prendre le chemin direct vers la bûche :

```
move()
move()
move()
turnRight()
move()
removeLog()
```

Cette solution semble, à première vue, être la plus courte, car elle permet d'atteindre la bûche non seulement dans le lac 1, mais aussi dans les lacs 2 et 3. Cependant, il est impossible de ramasser la bûche dans le quatrième lac, car le castor avance directement contre un rocher. Bien sûr, nous pourrions maintenant adapter le programme pour ramasser la bûche dans le quatrième lac. Mais il pourrait alors arriver que la solution ne permette plus de ramasser la bûche dans l'un des autres lacs.

Une bonne stratégie de résolution, lorsqu'il y a plusieurs lacs, est donc de commencer par les observer, afin de planifier le trajet en tenant compte de la position des rochers et des bûches dans tous les lacs.

C'est de l'informatique !

L'informatique s'occupe souvent d'abstraction, c'est-à-dire de la simplification de systèmes et de processus complexes. Programmer permet de décomposer des problèmes compliqués en plus petites

parties et de les résoudre de manière systématique. La programmation enseigne une manière de penser structurée, qui aide à aborder les problèmes de façon méthodique.

Souvent, nous cherchons une solution qui puisse être utilisée pour plusieurs problèmes similaires. Dans cet exemple, il s'agit donc de trouver une solution qui fonctionne non seulement pour un seul cas, mais pour plusieurs situations différentes.

Mots clés et sites web

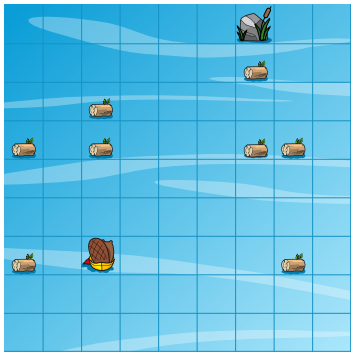
- Programmation : `https://fr.wikipedia.org/wiki/Programmation_informatique`
- Séquence : `https://fr.wikipedia.org/wiki/Structure_de_contrôle`

18. La bûche précieuse

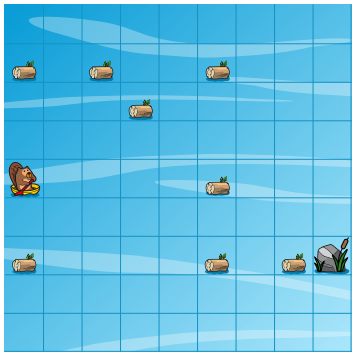
Pétunia le castor est à la recherche de bûches précieuses dans le Seeland. La bûche la plus précieuse se trouve toujours juste à côté d'un rocher. Aide Pétunia à écrire les instructions pour qu'elle atteigne, dans les trois lacs, la case avec la bûche précieuse près du rocher. Utilise le moins d'instructions possible. Clique sur les cercles sous le lac pour passer d'un lac à l'autre.

Tu peux utiliser ces instructions :

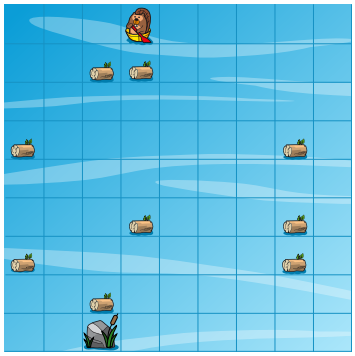
Instruction	**Effet**
`move()`	Pétunia avance d'exactement une case dans la direction de son regard
`turnRight()` / `turnLeft()`	Pétunia se tourne sur place de 90 degrés sur sa droite ou sur sa gauche
`goToLog()`	Pétunia avance en ligne droite devant elle jusqu'à ce qu'elle arrive sur une case avec une bûche



Lac 1



Lac 2



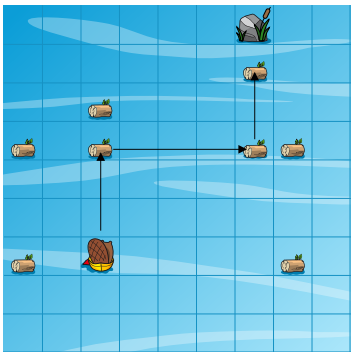
Lac 3

Écris un programme pour atteindre la bûche précieuse près du rocher dans les trois lacs. Essaie d'utiliser le moins d'instructions possible.

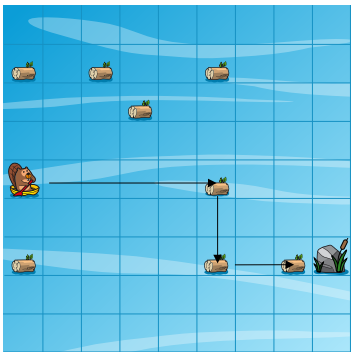
Solution

La solution correcte est la suivante :

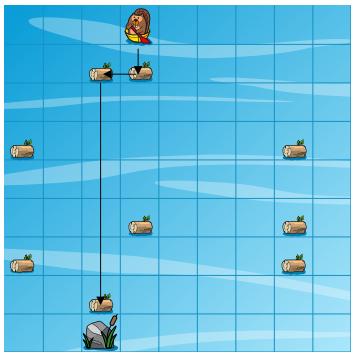
```
goToLog()
turnRight()
goToLog()
turnLeft()
goToLog()
```



Lac 1



Lac 2



Lac 3

Pour utiliser le moins d'instructions possible, il est nécessaire d'utiliser la commande `goToLog()`. Elle permet d'atteindre chaque bûche depuis la précédente, peu importe la distance qui les sépare.

Si, pour le lac 1, nous utilisions à la place la commande move(), nous arriverions également directement à la bûche précieuse :

```
move()
move()
move()
turnRight()
move()
move()
move()
move()
turnLeft()
move()
move()
```

Cependant, cette solution n'est pas celle qui utilise le moins d'instructions, car le programme est nettement plus long que la solution recherchée.

Un autre problème apparaît aussi : avec ce programme plus long, le castor n'atteint plus la bûche précieuse dans les lacs 2 et 3. Ce n'est qu'avec la commande `goToLog()` que nous parvenons à atteindre la bûche dans les trois lacs avec la même suite d'instructions, quelle que soit la distance entre le castor et la bûche dans chaque lac.

C'est de l'informatique !

L'informatique s'occupe souvent d'abstraction, c'est-à-dire de la simplification de systèmes et de processus complexes. Programmer permet de décomposer des problèmes compliqués en plus petites parties et de les résoudre de manière systématique. La programmation enseigne une manière de penser structurée, qui aide à aborder les problèmes de façon méthodique.

Souvent, nous cherchons une solution qui puisse être utilisée pour plusieurs problèmes similaires. Dans cet exemple, il s'agit donc de trouver une solution qui fonctionne non seulement pour un cas particulier, mais pour plusieurs situations différentes. Les solutions programmées qui ne fonctionnent que pour un seul cas précis sont appelées hardcode. Souvent, nous essayons d'éviter qu'une solution soit hardcodée et nous écrivons à la place des programmes utilisables pour plusieurs problèmes similaires.

Au lieu de compter combien de cases Pétunia doit avancer, nous utilisons la commande `goToLog()`, qui repose sur une structure de boucle (ici : avance tant que tu n'es pas sur une case avec une bûche). Cette approche rend inutile le comptage des cases et permet d'écrire une version plus courte et plus flexible du programme.

Mots clés et sites web

- Programmation : `https://fr.wikipedia.org/wiki/Programmation_informatique`
- Séquence : `https://fr.wikipedia.org/wiki/Structure_de_contrôle`
- Loop : `https://en.wikipedia.org/wiki/Control_flow#loop-statement`

A. Auteur·e·s des exercices

Masiar Babazadeh

Wilfried Baumann

Gi Soong Chee

Byeonggyu Cho

Vladimir Costas

Valentina Dagienė

Christian Datzko

Nora A. Escherle

Abeer Eshra

Gerald Futschek

Silvan Horvath

Alisher Ikramov

David Khachatryan

Doyong Kim

Jihye Kim

Vaidotas Kinčius

Stefan Koch

Lukas Lehner

Taina Lehtimäki

Gunwoong Lim

Mattia Monga

Anna Morpurgo

Kamohelo Motloung

Justina Oostendorp

Elsa Pellet

Jean-Philippe Pellet

Emiliano Pereiro

Zsuzsa Pluhár

Wolfgang Pohl

Pedro Ribeiro

Kirsten Schlüter

Dirk Schmerenbeck

Vipul Shah

Jacqueline Staub

Nikolaos Stratis

Susanne Thut

Christine Vender

Florentina Voboril

Michael Weigend

Philip Whittington

Kyra Willekes

B. Partenaires académiques

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

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

La Scuola universitaria professionale della Svizzera italiana (SUPSI)
`http://www.supsi.ch/`

Pädagogische Hochschule Zürich
`https://www.phzh.ch/`

Universität Trier
`https://www.uni-trier.de/`

C. Sponsoring

Fondation Hasler
http://www.haslerstiftung.ch/

Abraxas Informatik AG
https://www.abraxas.ch

Amt für Kindergarten, Volksschule und Beratung, Bildungs- und Kulturdirektion, canton de Berne
https://www.bkd.be.ch/de/start/ueber-uns/die-organisation/amt-fuer-kindergarten-volksschule-und-beratung.html

Amt für Wirtschaft, canton de Zurich
https://www.zh.ch/de/volkswirtschaftsdirektion/amt-fuer-wirtschaft.html

Fondation d'Informatique Suisse
https://informatics-foundation.ch

cyon
https://www.cyon.ch

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

Wealth Management IT and UBS Switzerland IT
http://www.ubs.com/