INFORMATIK-BIBER SCHWEIZ
CASTOR INFORMATIQUE SUISSE
CASTORO INFORMATICO SVIZZERA

Exercices et solutions 2025

Années HarmoS 9/10

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

Éditeurs :

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

SSIE
www.svia-ssie-ssii.ch
schweizerischervereinfürinformatikinderausbildung//sociétésuissepourl'informatiquedansl'enseignement//societàsvizzeraperl'informaticanell'insegnamento

HASLERSTIFTUNG
ETH zürich
ABZ Ausbildungs- und Beratungszentrum für Informatikunterricht
Scuola universitaria professionale della Svizzera italiana SUPSI
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/`

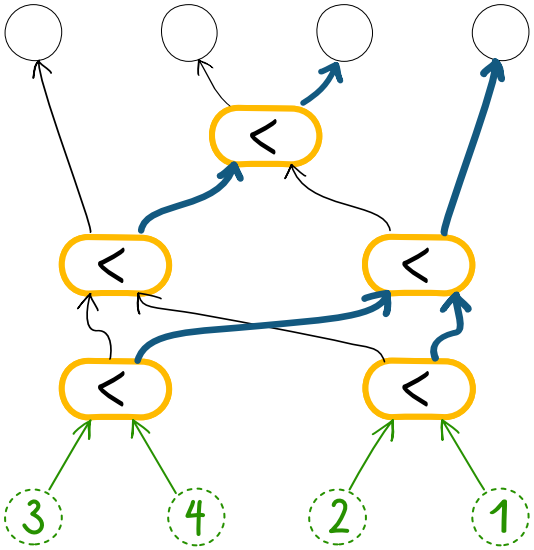
Table des matières

1. 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.

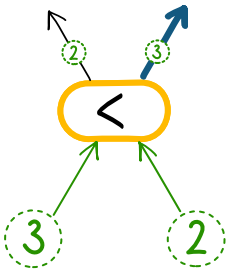
Les nombres traversent la machine de bas en haut en passant le long des flèches et des interrupteurs jusqu'à arriver aux champs de sortie.



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.

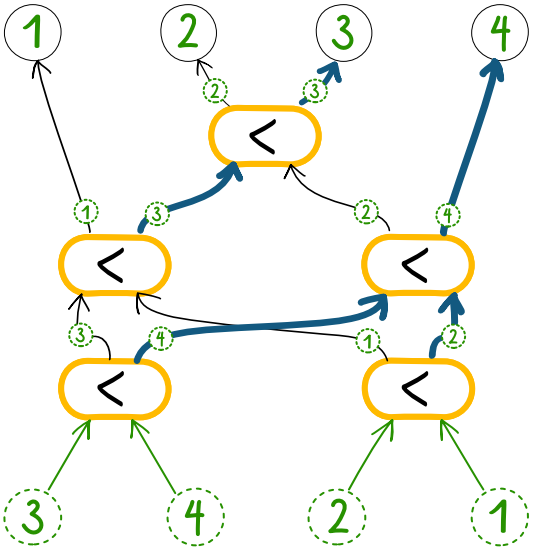
C) Elle donne les nombres dans leur ordre d'entrée ; le résultat de l'exemple est 3, 4, 2, 1.

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)`

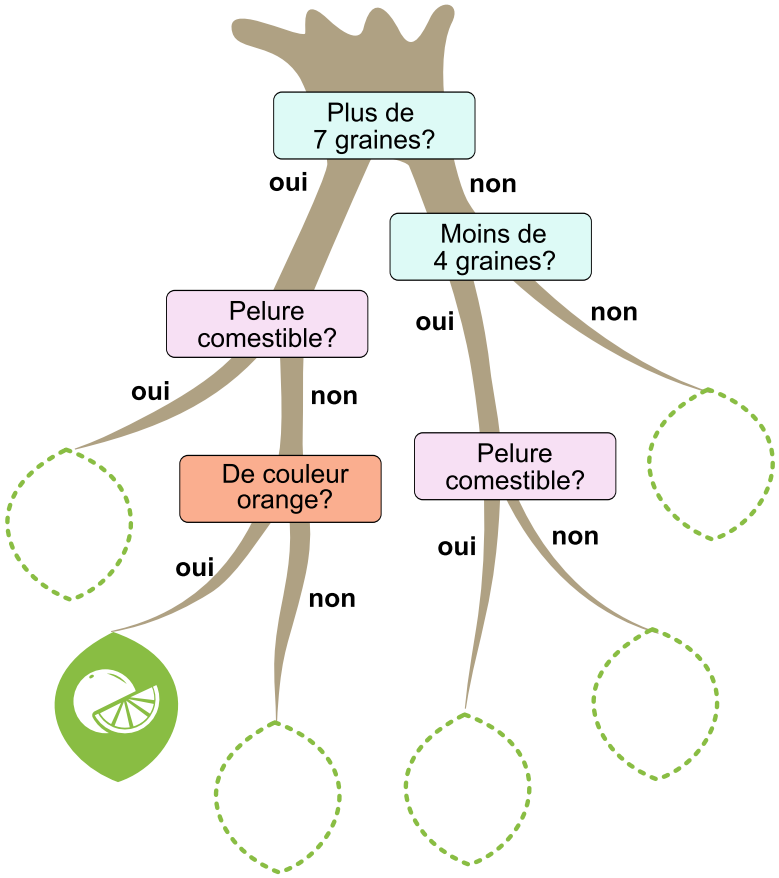
2. Arbre de décision

À l'école, une classe analyse des fruits. Trois propriétés sont examinées pour chaque fruit, et la valeur de ces propriétés détermine de quel fruit il s'agit. Les propriétés sont la couleur extérieure, le nombre de graines et si la pelure est comestible. La classe a noté les valeurs des propriétés et la sorte de fruit qu'ils en ont déduite dans un tableau :

Couleur	**Nombre de graines**	**Pelure comestible ?**	**Sorte de fruit**
vert	391	non	pastèque
jaune	5	oui	pomme
orange	9	non	orange
jaune	0	non	banane
rouge	5	oui	pomme
vert	0	oui	raisins
rouge	206	oui	fraise
vert	6	oui	pomme
orange	10	non	orange
rouge	173	oui	fraise

La sorte de fruit est déterminée à l'aide d'un arbre de décision. Un arbre de décision ressemble à un arbre à l'envers : la racine est en haut et les feuilles sont en bas. Des questions auxquelles on peut répondre par oui ou par non sont notées sur la racine et les embranchements.

Pour déterminer la sorte d'un fruit, on répond aux questions de l'arbre à l'aide des valeurs des propriétés comme cela : commence en haut à la racine. Réponds à la question et va sur la branche avec la réponse correspondante (oui ou non). Réponds à la question suivante et va sur la prochaine branche avec la bonne réponse. Continue ainsi jusqu'à ce que tu atteignes une feuille. La feuille te donne la sorte de fruit.

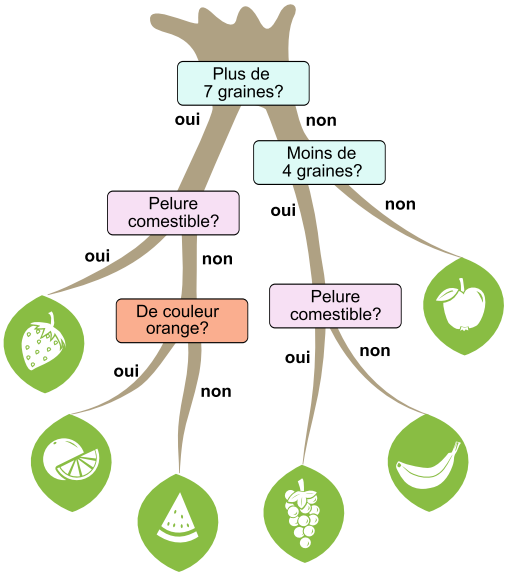


Malheureusement, l'arbre s'est cassé après 10 fruits : presque toutes les feuilles sont tombées.

Au bout de quelle branche se trouvait quelle feuille ?

Solution

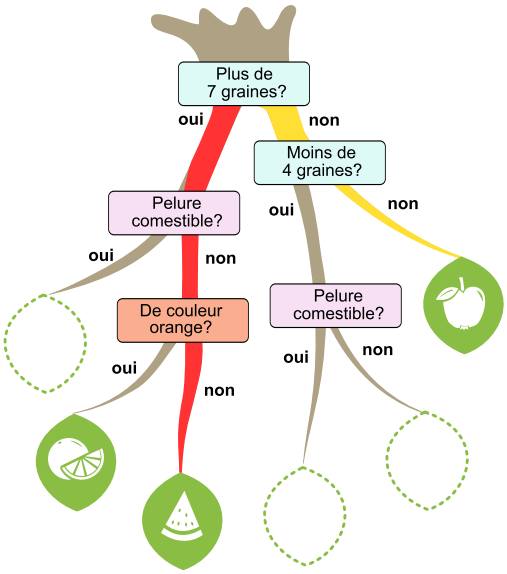
Voici la bonne réponse :



De gauche à droite, les fruits sur les feuilles sont : fraise, orange, pastèque, raisins, banane, pomme.

Comment trouve-t-on les bonnes branches pour les feuilles ?

Les sortes de fruits du tableau ont été déterminées grâce à l'arbre de décision. Parcourons donc le tableau ligne par ligne en regardant au bout de quelle branche les questions de l'arbre nous mènent. Au bout de la branche doit se trouver la feuille avec la sorte de fruit indiquée dans la dernière colonne du tableau. L'image montre les deux chemins correspondant aux deux premières lignes du tableau dans l'arbre de décision :



Première ligne (chemin rouge) : le fruit a 391 graines (Plus de 7 graines ? – oui), sa pelure n'est pas comestible (Pelure comestible ? – non), et le fruit est vert (De couleur orange ? – non). Ce chemin mène au bout de la troisième branche depuis la gauche. Il faut donc y mettre la feuille avec le fruit indiqué dans la dernière colonne de la première ligne : la pastèque.

Deuxième ligne (chemin jaune) : le fruit a 5 graines (Plus de 7 graines ? – non. Moins de 4 graines ? – non). Le chemin mène au bout de la branche tout à droite : il faut donc y mettre la feuille avec la pomme.

De la même manière, – la ligne 3 mène au bout de la deuxième branche depuis la gauche, où se trouve déjà l'orange ;

- la ligne 4 mène au bout de la deuxième branche depuis la droite, où il faut mettre la banane ;
- la ligne 5 mène à la pomme ;
- la ligne 6 mène au bout de la troisième branche depuis la droite, où il faut mettre les raisins,
- et la ligne 7 mène au bout de la branche tout à gauche, où il faut mettre la fraise.

C'est de l'informatique !

Un arbre de décision représente un moyen facile mais malin de trier des objets par sortes ou groupes (de les *classifier*) et de prendre des décisions basées sur cette classification. Les arbres de décision sont beaucoup utilisés en informatique : par exemple en médecine par des programme de diagnostique, par les assistants virtuels qui aident à trouver des produits ou dans les jeux vidéos lorque le programme doit décider comment un caractère réagit à une situation.

L'arbre de décision de cet exercice du castor était déjà donné, peut-être par l'enseignante ou l'enseignant de biologie. La personne qui l'a construit a réfléchi à quelles questions étaient importantes, dans quel ordre elles devaient être posées et comment les fourches de l'arbre devaient être arrangées pour que les fruits soient classifiés de la bonne manière.

En intelligence artificielle (IA), un domaine de l'informatique qui est devenu de plus en plus important ces dernières années, il faut aussi souvent examiner des données, les classifier puis prendre des décisions. Les outils utilisés pour cette classification – comme les arbres de décision – doivent être déterminés automatiquement à l'aide des données examinées. Les méthodes informatiques permettant de construire des structures de classification et de décision automatiquement relèvent de l'*apprentissage automatique* (*machine learning* en anglais) et/ou de la *science des données* (*data science*). Les arbres de décision peuvent eux aussi être « appris », ou construits automatiquement, par des algorithmes comme CART ou C4.5. Ces méthodes utilisent de nombreux exemples pour déterminer quelle question doit être posée quand pour prendre de bonnes décisions. Les arbres de décision automatiqes sont utilisés pour la reconnaisance du courrier indésirable, le diagnostique médical ou la reconnaissance de plantes, par exemple.

Cependant, la plupart des systemes d'IA connus n'utilisent **pas** d'arbres de décisions. Souvent, de grands modèles appelé « réseaux de neurones » sont utilisés. Ces structures ne contiennent pas de

questions concrètes, mais utilisent des méthodes statistiques complexes et difficiles à comprendre pour les être humains.

Mots clés et sites web

- Arbre de décision : `https://fr.wikipedia.org/wiki/Arbre_de_décision`
- Classification : `https://fr.wikipedia.org/wiki/Classification`

3. Guirlande lumineuse

Sophie a trois sortes de lampes dans son set d'électronique : des lampes rondes rouges, des lampes carrées bleues et des lampes pentagonales jaunes. Elle a relié plusieurs lampes pour former une étoile. Les flèches montrent comment les câbles sont placés.

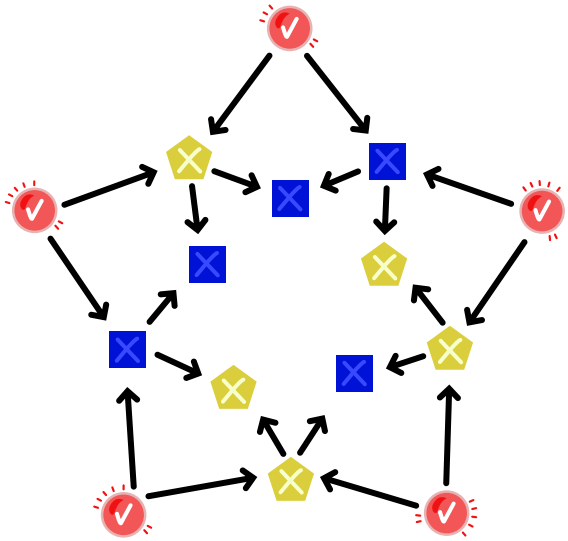
Les lampes bleues et jaunes sont contrôlées par les câbles dans le sens des flèches. Chaque lampe jaune ou bleue a donc exactement deux lampes de contrôle.

Voici comment fonctionnent les lampes :

- Sophie peut allumer et éteindre les lampes rouges elle-même.
- Une lampe bleue est allumée quand ses deux lampes de contrôle sont allumées ; elle est éteinte sinon.
- Une lampe jaune est allumée quand exactement une de ses lampes de contrôle est allumée ; elle est éteinte sinon.

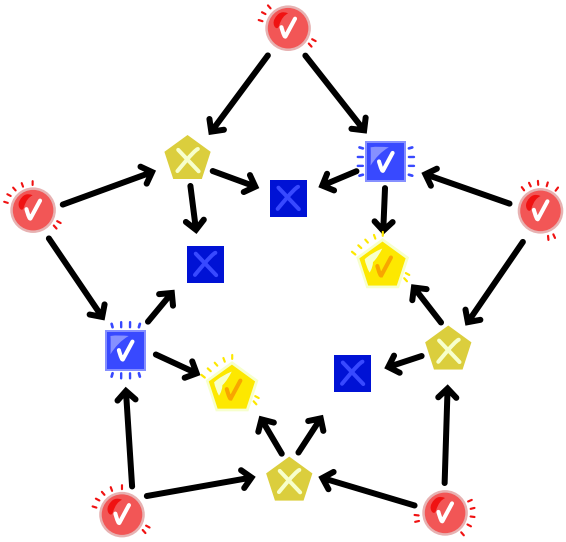
Sophie allume toutes les lampes rouges.

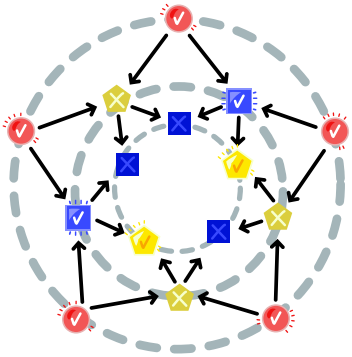
Quelles autres lampes sont ainsi allumées ?



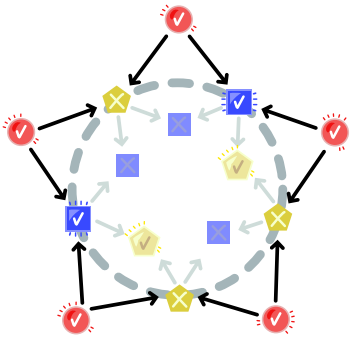
Solution

Voici la bonne réponse :

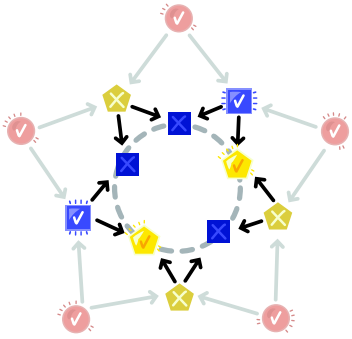




Les lampes forment trois cercles : un cercle extérieur avec toutes les lampes rouges, puis un cercle central et un cercle intérieur avec des lampes jaunes et bleues.



Toutes les lampes rouges, et donc toutes les lampes qui contrôlent le cercle central, sont allumées. Dans le cercle central, les lampes bleues sont donc allumées (car les *deux* lampes de contrôle sont allumées), et les lampes jaunes sont éteintes (car les *deux* lampes de contrôles sont allumées, et non pas *exactement une*).



C'est exactement l'inverse dans le cercle intérieur : chaque lampe a une lampe de contrôle bleue et une lampe de contrôle jaune, les deux placées dans le cercle central. Nous savons déjà qu'il n'y a toujours qu'une seule de ces deux lampes allumées. Les lampes jaunes du cercle central sont donc allumées, et les lampes bleues éteintes.

C'est de l'informatique !

Les lampes de l'étoile de Sophie ne peuvent être qu'allumées ou éteintes. On peut aussi dire que chaque lampe peut avoir deux valeurs : **allumée** ou **éteinte**, comme un feu de circulation peut avoir trois valeurs, « rouge », « orange » et « vert », ou une montre à affichage digital peut avoir 1440 valeurs de 00 :00 à 23 :59, ou encore comme l'affichage d'une caisse de supermarché peut avoir de nombreux montants comme valeur.

Avec les nombres, on peut calculer à l'aide d'*opérateurs* comme plus et moins. Il existe aussi des opérateurs pour les deux valeurs des lampes, et ces valeurs peuvent être appelées **allumé** et **éteint**, **oui** et **non**, **vrai** et **faux** ou **0** et **1**. Ces opérateurs calculent un résultat à partir de deux valeurs d'*entrée*. Deux de ces opérateurs sont utilisés dans cet exercice du castor :

- La valeur des lampes bleues est calculée avec l'opérateur *ET* (*AND* en anglais). Le résultat de l'opérateur ET est 1 si les deux entrées sont 1, et 0 sinon.
- La valeur des lampes jaunes est calculée avec l'opérateur *OU exclusif* (*XOR* en anglais). Le résultat de l'opérateur OU exclusif est 1 si les deux entrées sont différentes, et 0 sinon.

L'opérateur OU exclusif porte ce nom car il exclut que le résultat soit 1 lorsque les deux entrées sont 1. Il existe aussi un opérateur similaire qui fonctionne sans cette exclusion : le *OU* (*OR* en anglais).

En informatique, ces opérateurs et quelques autres sont appelés *opérateurs logiques*. Les opérateurs peuvent être construits avec des interrupteurs électroniques, qui sont des éléments de base des microprocesseurs. En effet, la plus petite unité informatique, le bit, a elle aussi deux valeurs. En les combinant de la bonne manière, on peut les utiliser pour faire des calculs compliqués et écrire n'importe quel programme.

Mots clés et sites web

- Logique : `https://fr.wikipedia.org/wiki/Logique`
- Algèbre de Boole : `https://fr.wikipedia.org/wiki/Algèbre_de_Boole_(logique)`
- Opérateur logique : `https://fr.wikipedia.org/wiki/Connecteur_logique`
- AND : `https://fr.wikipedia.org/wiki/Fonction_ET`
- XOR : `https://fr.wikipedia.org/wiki/Fonction_OU_exclusif`
- OR : `https://fr.wikipedia.org/wiki/Fonction_OU`

4. Architecture

Tu as les plots suivants :

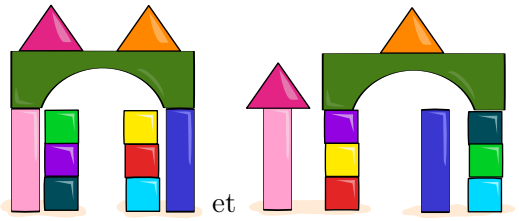
6 cubes — 2 parallélépipèdes — 1 pont — 2 pyramides

Ton ami te donne ces instructions pour réaliser des constructions avec les plots :

1. Prends 3 cubes.
2. Empile les 3 cubes les uns sur les autres pour construire une tour.
3. Construis une seconde tour avec les 3 autres cubes.
4. Pose les parallélépipèdes à côté des tours.
5. Pose le pont sur ta construction.
6. Prends les 2 pyramides et mets-les sur ta construction.

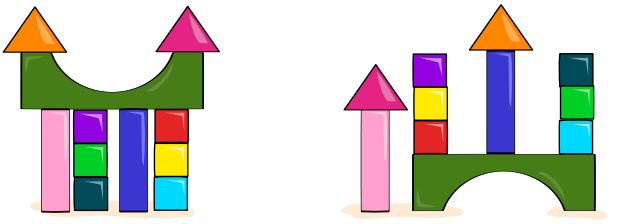
Tu dois suivre les 6 instructions dans l'ordre en construisant. Ces instructions te permettent de réaliser plusieurs constructions différentes.

Voici deux exemples :



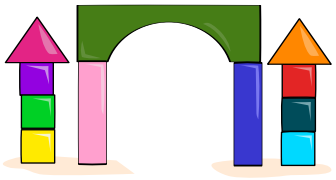
et

*Voici 4 autres constructions. L'une d'entre elles ne peut **PAS** être construite en suivant les instructions. Laquelle ?*

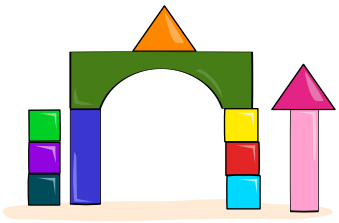


A)

B)



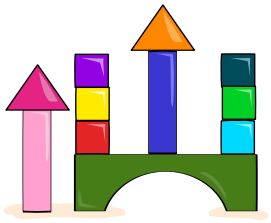
C)



D)

Solution

La bonne réponse est B.



Nous savons déjà que l'on peut réaliser plusieurs constructions différentes en suivant les mêmes instructions. Ces instructions ne sont donc pas *univoques*. Une chose est claire : l'ordre des instructions doit être respecté. Regardons plus en détail ce qu'il peut se passer lorsque l'on suit les instructions pas à pas. Nous utilisons un tableau pour décrire la construction après chaque instruction et vérifions quelle construction correspond à cette description.

Après l'instruction	Description	A	B	C	D
1 et 2	Il y a une première tour composée de 3 cubes.	✓	✗	✓	✓
3	Il y a 2 tours de 3 cubes chacune.	✓	✗	✓	✓
4	Il y a 2 tours et 2 parallélépipèdes.	✓	✗	✓	✓
5	Comme plus haut, et le pont est posé sur une partie de la construction réalisée plus tôt – donc sur des tours ou des parallélépipèdes.	✓	✗	✓	✓
6	Comme plus haut, et les pyramides sont posées sur une partie de la construction réalisée plus tôt (les tours, les parallélépipèdes ou le pont).	✓	✗	✓	✓

La construction de la réponse B ne correspond donc pas aux instructions. En particulier, l'instruction 5 n'a pas été suivie : elle dit que le pont doit être posé *sur* la construction réalisée avant. Dans la construction de la réponse B, le pont ne se trouve sur aucune autre pièce, mais *sous* des pièces (2 tours et un parallélépipède). La construction de la réponse B ne peut être réalisée que si le pont est construit avant les deux tours et le parallélépipède.

Le tableau montre que les constructions des réponses A, C et D peuvent être construites en suivant les instructions.

C'est de l'informatique !

Cet exercice montre l'importance d'instructions claires et univoques. Les instructions de montage et les recettes de cuisine en sont des exemples. Des instructions formulées de manière non univoque peuvent mener à des résultats différents des résultats prévus. Les instructions de cet exercice ne spécifient par exemple pas où ou dans quels sens les plots ou tours doivent être placés.

Les ordinateurs ont besoin d'instructions claires et univoques pour fonctionner comme les êtres humains le souhaitent. En informatique, de telles instructions sont appelées *algorithme* : une suite d'instructions décrivant pas à pas comment résoudre un problème ou effectuer une tâche. Chaque instruction d'un algorithme doit être univoque, car, contrairement aux êtres humains, les ordinateurs ne peuvent pas interpréter ou deviner. Ils ont donc besoin d'instructions univoques pour effectuer des tâches de manière prévisible. Si les instructions ne sont pas univoques, les logiciels peuvent ne pas bien fonctionner, les programmes ne pas être effectués comme souhaité ou donner des résultats imprévus.

Avant d'écrire un programme, la programmeuse ou le programmeur doit se demander quelles *contraintes* doivent être définies pour obtenir un résultat prévisible. Dans cet exercice du Castor, de telles contraintes pourraient par exemple définir l'orientation verticale ou horizontale des plots, l'emplacement des différentes parties de la construction ou comment les plots doivent être reliés à la construction existante.

Mots clés et sites web

- *Algorithme* : `https://fr.wikipedia.org/wiki/Algorithme`
- *Contraintes (Programmation par contraintes)* : `https://fr.wikipedia.org/wiki/Programmation_par_contraintes`

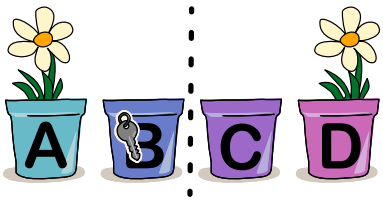
5. Pots de fleurs

Florian le castor décore l'entrée de sa maison avec des pots de fleurs. Dans certains pots, il y a **exactement une fleur**, et les autres pots sont **vides**.

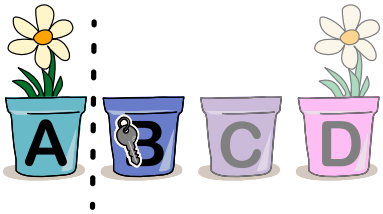
Une clé est cachée dans l'un des pots. Florian explique comment il fait pour trouver la clé.

«D'abord, je regarde tous les pots et compte combien de fleurs sont plantées dans les pots en tout. Si le nombre de fleurs est pair, la clé est dans la moitié gauche des pots, sinon elle est dans la moitié droite. Ensuite, je ne regarde que la moitié dans laquelle la clé se trouve, et je répète ce procédé jusqu'à ce qu'il ne reste qu'un seul pot. La clé est cachée dans ce pot.»

Florian montre comment trouver la clé cachée dans l'un des quatre pots A, B, C et D.



Il regarde les pots A, B, C et D. Il y a deux fleurs en tout, ce qui est un nombre **pair**. Cela veut dire que la clé est dans la moitié de **gauche**, dans le pot A ou B.



Il regarde les pots A et B. Il y a une fleur en tout, ce qui est un nombre **impair**. Cela veut dire que la clé est dans la moitié de **droite**, donc dans le pot B.

Florian a 8 pots de fleurs et cache la clé dans le pot C. Dans quels pots doit-il planter une fleur afin que l'on puisse trouver la clé avec sa méthode ?

Il y a plusieurs réponses possibles. 0 est aussi un nombre pair.

Solution

Voici une bonne réponse :

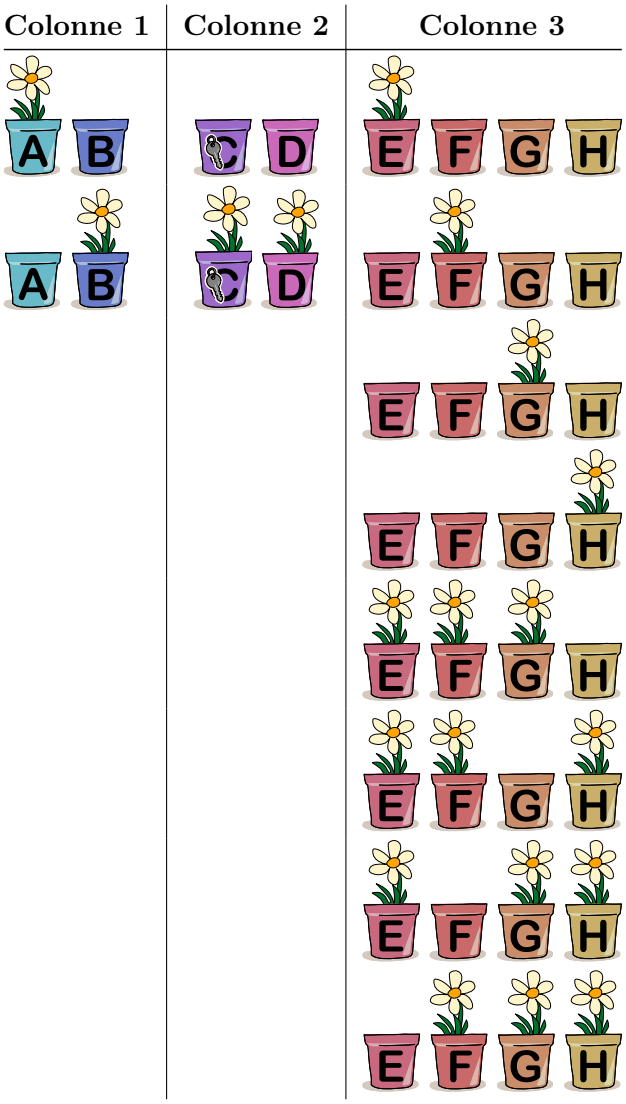
En voici une autre :

Il y a encore d'autres bonnes réponses : 32 en tout. On peut les trouver comme expliqué ci-dessous.

Le plus simple est de construire la solution « de haut en bas », c'est-à-dire en commençant par une petite partie de la réponse et en remontant jusqu'à la solution complète.

- Si la clé est dans le pot C, les derniers pots à être observés sont les pots C et D. Pour que la moitié gauche, C, soit choisie, il doit y avoir un nombre de fleurs pair dans les pots C et D. Il y a donc soit une fleur dans chaque pot, soit aucune fleur.
- L'étape d'avant considère les pots A à D. Comme la moitié droite (C et D) doit être choisie, il doit y avoir un nombre impair de fleurs dans les pots A à D. Comme il y en a un nombre pair dans les pots C et D, il doit y avoir soit une fleur dans le pot A, soit une fleur dans le pot B.
- La première étape considère tous les pots. Comme la clé se trouve dans la moitié de gauche (A à D), il faut un nombre de fleurs total pair. Comme la moitié gauche contient un nombre impair de fleurs, la moitié de droite (E à H) doit aussi contenir un nombre impair de fleurs. On peut doic choisir entre une ou trois fleurs à répartir dans les pots E, F, G et H.

Le tableau suivant récapitule toutes les réponses possibles : on peut construire une réponse complète en choisissant une option dans chaque colonne et en les combinants. On obtient ainsi toutes les $2 \times 2 \times 8 = 32$ solutions.



C'est de l'informatique !

Florian *code* la position de la clé à l'aide d'une suite de fleurs dans ses pots. Ses amis peuvent décoder cette représentation parce qu'ils connaissent la méthode de codage de Florian. Pour que les ordinateurs puissent traiter des données, elle ne sont pas enregistrée dans un forme naturelle et facilement compréhensible pour l'être humain, mais sont codées sous une forme lisible par les ordinateurs. Beaucoup de méthodes de codage existent en informatique. Certaines servent à économiser l'espace de stockage : on parle alors de *compression*. Si une *clé* est nécessaire pour déchiffrer les données codées, on parler alors de *chiffrement*. Dans tous les cas, il est important que le code soit univoque pour pouvoir retrouver exactement les données originales. Chaque combinaison de pots avec et sans fleur détermine de manière univoque dans quel pot se trouve la clé : le code de Florian dans cet exercice du castor remplit donc cette condition.

La méthode de Florian pour trouver la clé parmi les pots de fleurs fonctionne de manière semblable à la *recherche dichotomique* lors de laquelle l'espace de recherche est divisé par deux à chaque étape.

Cela permet de vite atteindre l'objectif. S'il y a deux fois plus de pots, il ne faut qu'une étape supplémentaire.

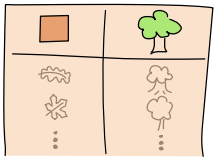
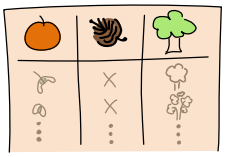
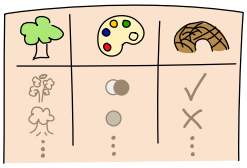
Dans cet exercice, le meilleur moyen de trouver la solution est d'utiliser un approche ascendante (ou *bottom-up*). Pour cela, on commence par considérer les plus petites parties de la réponse, étant donné que les plus grandes en dépendent. Au lieu d'essayer toutes les possibilités de planter des fleurs – $2^8 = 256$ possibilités ici – on arrive rapidement à une bonne réponse en combinant les réponses partielles.

Mots clés et sites web

- Approche ascendante/descendante : `https://fr.wikipedia.org/wiki/Approches_ascendante_et_descendante`
- Code : `https://fr.wikipedia.org/wiki/Code_(information)`
- Recherche dichotomique : `https://fr.wikipedia.org/wiki/Recherche_dichotomique`
- Recherche exhaustive : `https://fr.wikipedia.org/wiki/Recherche_exhaustive`

6. 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.

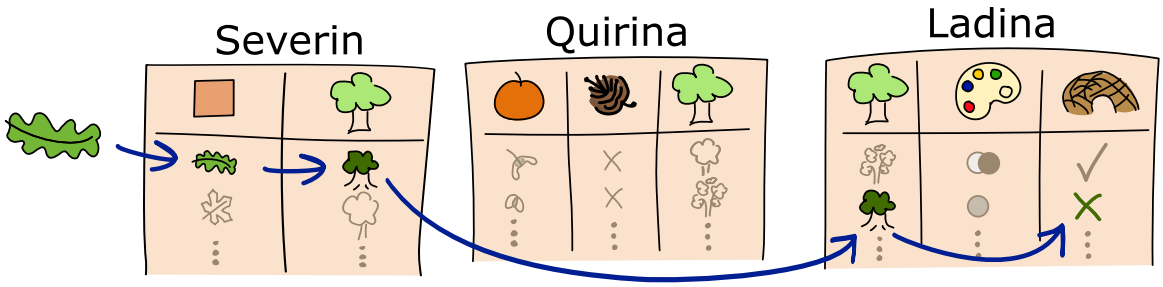
Solution

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.

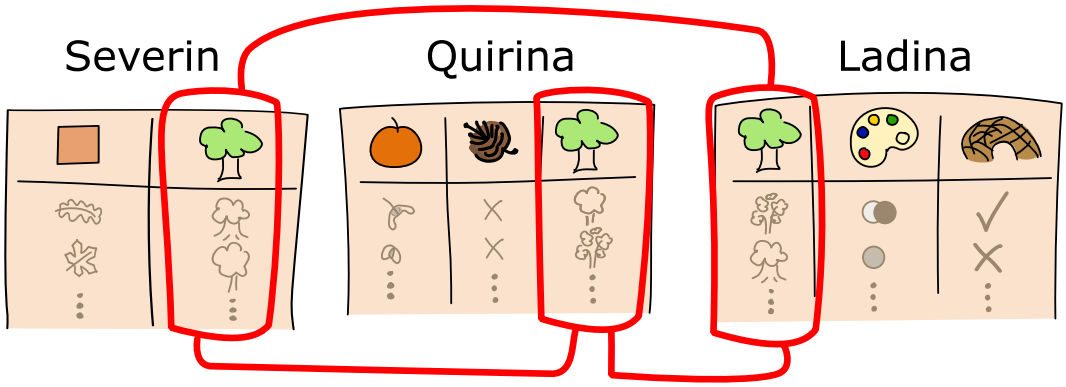


C'est de l'informatique !

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 :



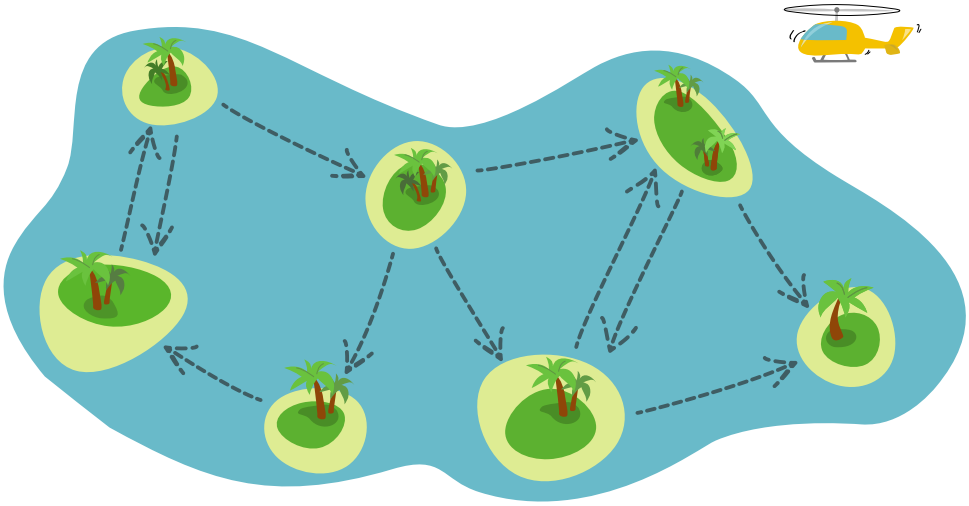
Les bases de données relationnelles sont si importantes qu'il existe un langage spécial pour les requêtes et autres opérations sur les bases de données, le *SQL (Structured Query Language)*.

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`

7. 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

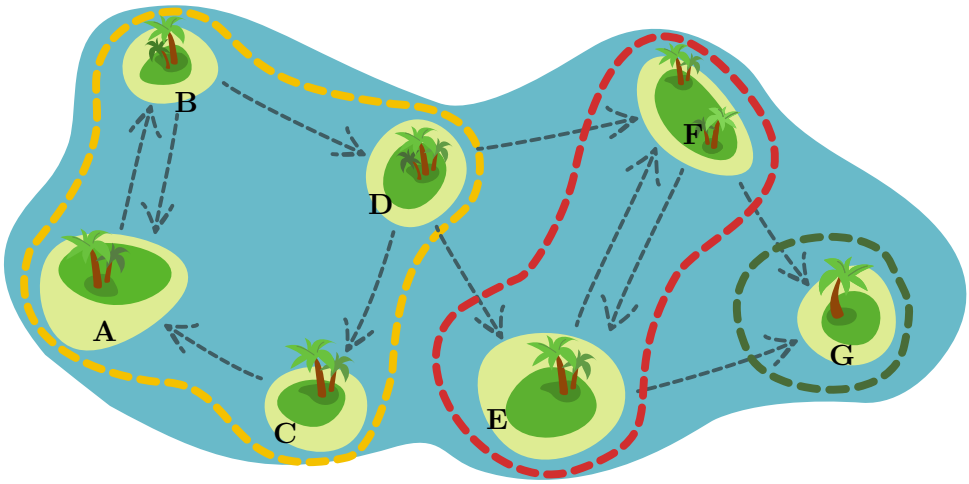
Solution

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

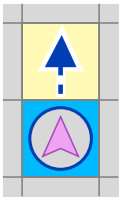
- Graphe orienté : `https://fr.wikipedia.org/wiki/Graphe_orienté`
- Composante fortement connexe : `https://fr.wikipedia.org/wiki/Composante_fortement_connexe`
- Algorithme de Tarjan : `https://fr.wikipedia.org/wiki/Algorithme_de_Tarjan`

8. Lefty II

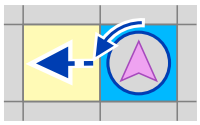
Lefty est un robot qui se déplace sur une grille à cases carrées. Il peut y avoir des murs rouges entre les cases. Lefty doit atteindre le but vert.

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

avancer d'une case

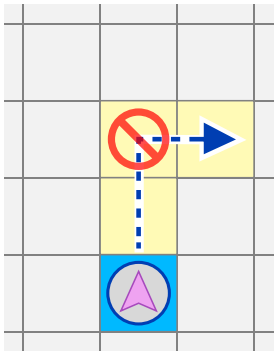


se tourner vers la gauche
puis avancer d'une case

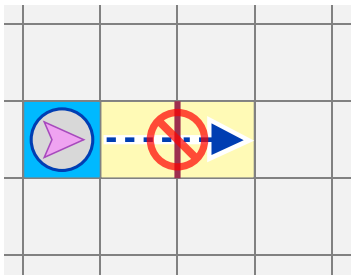


Lefty ne peut pas tout faire. Par exemple, il ne peut…

… **pas** simplement tourner à droite

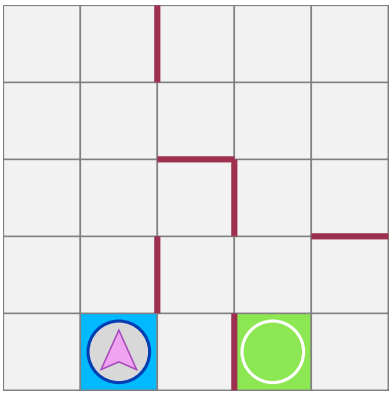


… **pas** traverser des murs



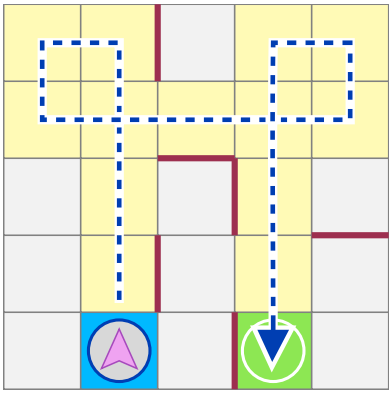
Par quelles cases Lefty ***doit****-il passer pour atteindre le but ?*

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



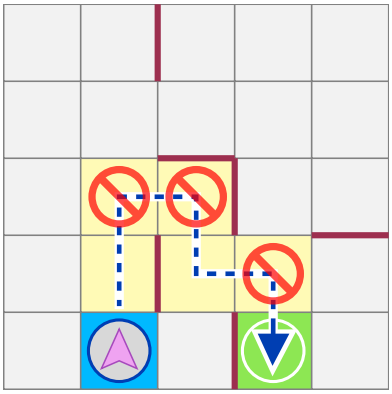
Solution

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.

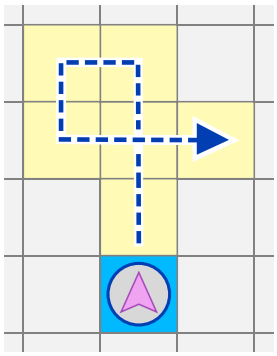


C'est de l'informatique !

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

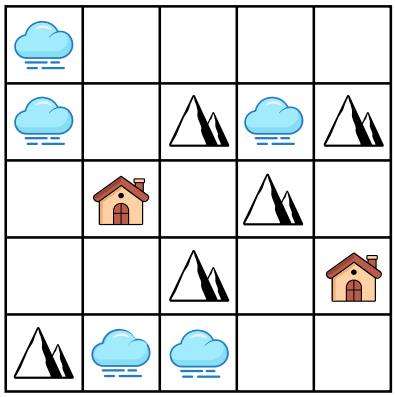
- Processeur : `https://fr.wikipedia.org/wiki/Processeur`
- jeu d'instructions : `https://fr.wikipedia.org/wiki/Jeu_d'instructions`
- CISC : `https://fr.wikipedia.org/wiki/Microprocesseur_à_jeu_d'instructions_étendu`
- RISC : `https://fr.wikipedia.org/wiki/Processeur_à_jeu_d'instructions_réduit`

9. Jour de brouillard

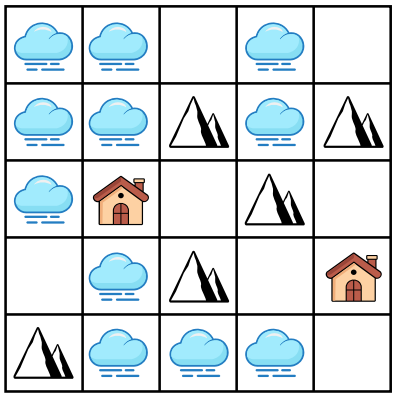
Aujourd'hui, il y a du brouillard au pays montagneux. Le brouillard s'étend d'heure en heure.

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.

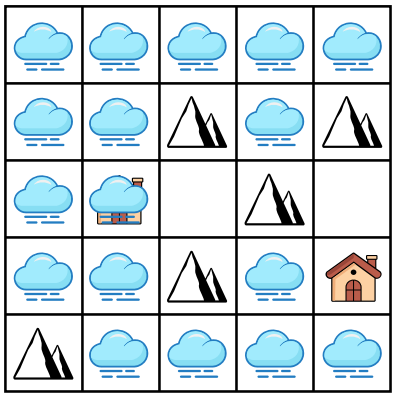
Voici un exemple :



À l'aube

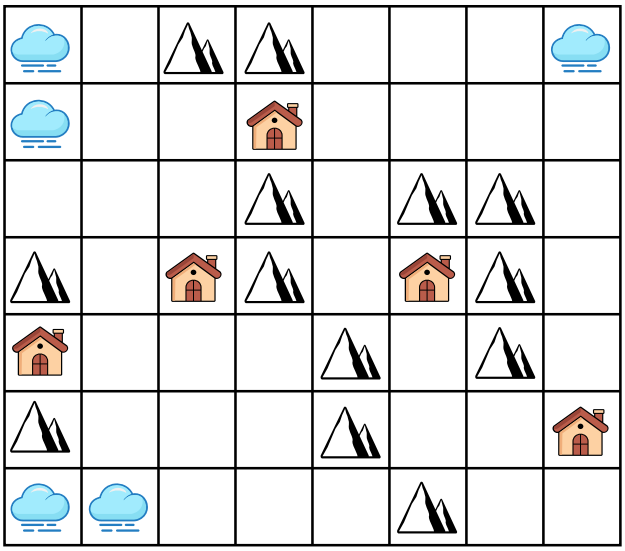


1 heure plus tard



2 heures plus tard

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

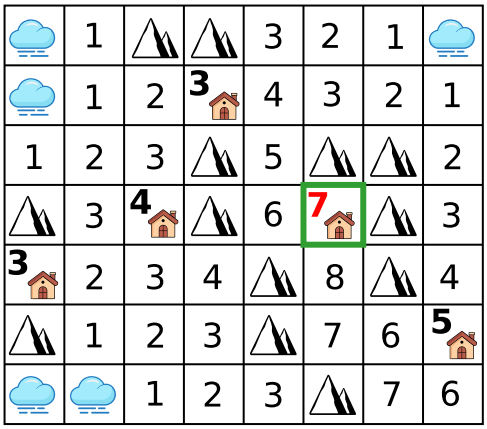


Solution

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.

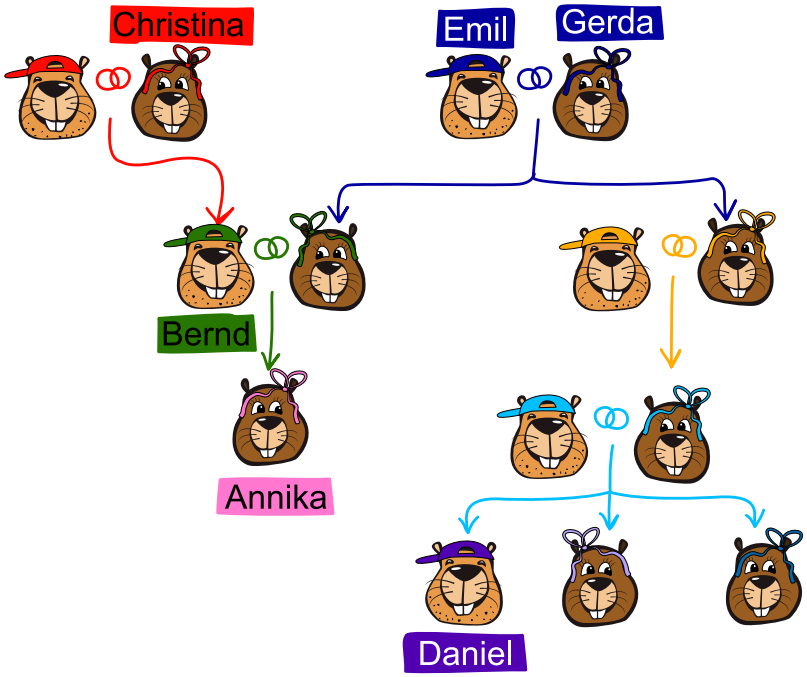
Les ensembles connexes sont intéressants pour différents domaines de l'informatique. Une zone de couleur unie sur une image (numérique) est un ensemble de pixels connexe de la même couleur ; on peut le déterminer à l'aide d'un *algorithme de remplissage par diffusion* qui fonctionne de manière similaire 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`

10. 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.

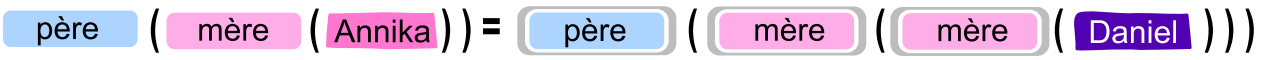
Complète l'équation suivante pour qu'elle décrive le lien de parenté entre Daniel et Annika.

père mère

père (mère (Annika)) = (((Daniel)))

Solution

Voici la bonne réponse :



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`

11. 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.

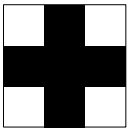
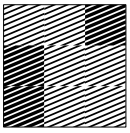


Image secrète


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.

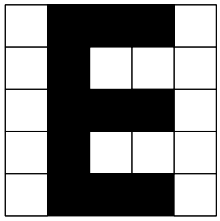
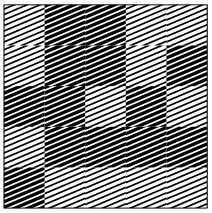
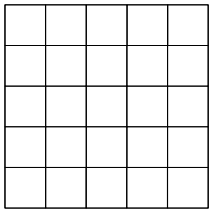


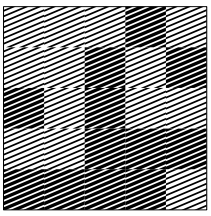
Image secrète


Transparent 1


Transparent 2

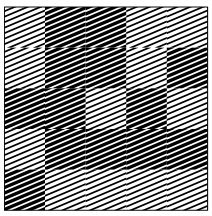
Solution

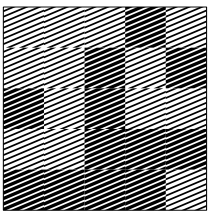
Voici la bonne réponse :

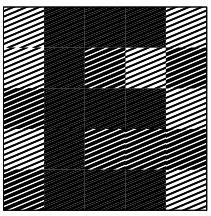


Sur cette image du transparent 2, chaque pixel a été imprimé en suivant la règle décrite plus haut – en fonction de l'image secrète et du transparent 1. L'image du transparent 2 n'est différente du transparent 1 qu'aux endroits où l'image secrète a un pixel noir.

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 !

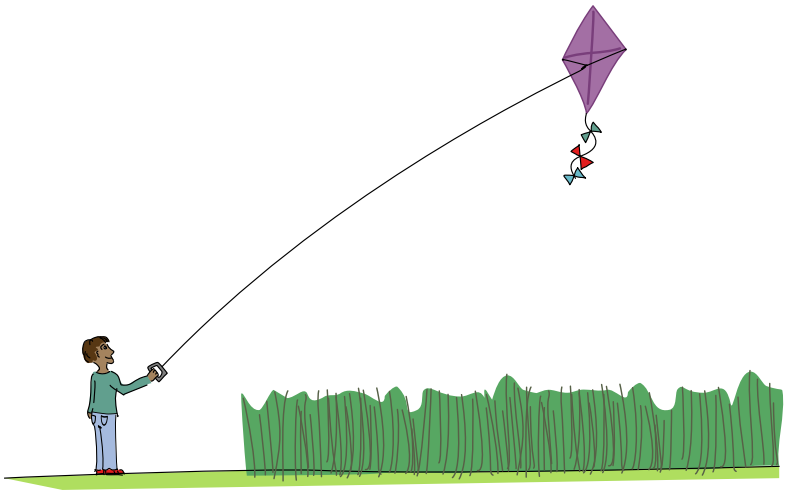
Le service de messagerie utilise un *procédé cryptographique* basé sur une image – ce qui est appelé *cryptographie visuelle*. La technique de cet exercice du castor a été développée en 1994 par les scientifiques israéliens Moni Naor et Adi Shamir. Le procédé est très sûr pour un unique transparent : 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`

12. Cerf-volant perdu

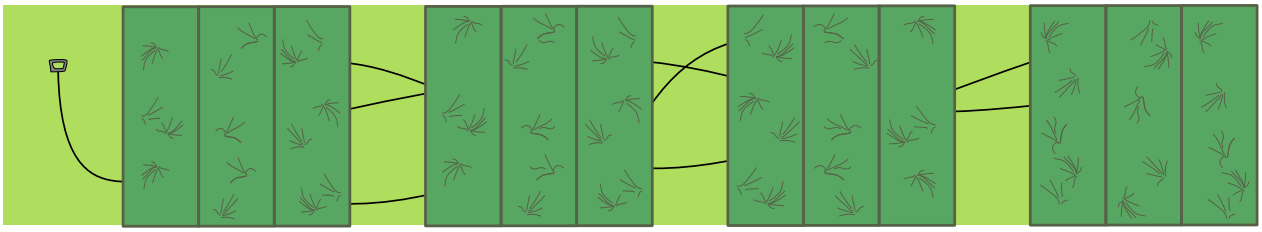


Pas de chance! Asterios a perdu son cerf-volant dans la prairie. Le fil s'est emmêlé dans les hautes herbes et c'est difficile de retrouver le cerf-volant.

La prairie est divisée en 15 cases qui peuvent être examinées une à une.

Asterios a déjà examiné 3 cases de la prairie. Il regarde attentivement comment le fil est arrangé dans ces cases et remarque qu'il ne doit plus examiner qu'une seule case pour savoir exactement dans quelle case est son cerf-volant.

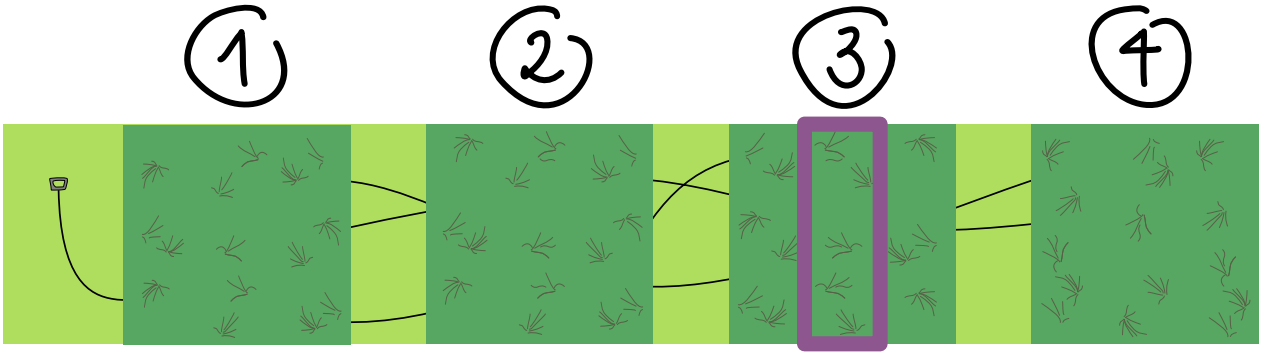
De quelle case s'agit-il ?



Solution

Voici la bonne réponse :

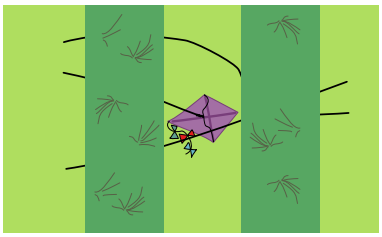
Asterios doit examiner la case encadrée en violet pour savoir exactement dans quelle case se trouve son cerf-volant.



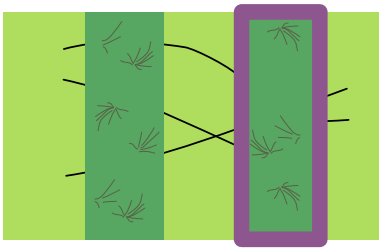
Tu peux trouver la solution en deux étapes. D'abord, tu détermines dans lequel des quatre blocs 1, 2, 3 ou 4 doit se trouver le cerf-volant. Pense au fait que le fil commence à gauche et que le cerf-volant se trouve à l'autre extrémité.

Le cerf-volant ne peut pas être dans le bloc 4, car le fil y entre et en ressort. Le cerf-volant doit se trouver à la droite du bloc 2, car on voit trois parties de fil entre le bloc 2 et le bloc 3. Deux parties du fil forment une boucle à droite de la prairie, et la troisième doit donc mener au cerf-volant.

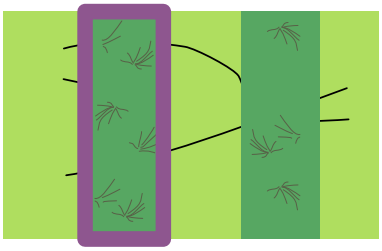
Lorsqu'Asterios examine la case au milieu du bloc 3, il peut tomber sur l'un des trois cas suivants :



Cas 1 : Le cerf-volant se trouve dans la case du milieu et la recherche est terminée.



Cas 2 : Il n'y pas de cerf-volant, mais trois parties de fil qui entrent dans la case de droite. Comme il n'y a que deux parties de fil qui sortent de la case de droite, le cerf-volant doit s'y trouver.



Cas 3 : Il n'y pas de cerf-volant, mais deux parties de fil qui y entrent depuis la case de gauche. Le cerf-volant doit donc être dans la case de gauche, car les deux parties de fil font partie de la boucle à droite de la prairie.

Dans chacun de ces cas, nous savons dans quelle case se trouve le cerf-volant après avoir examiné la case au milieu du bloc 3.

C'est de l'informatique !

Lors d'un parcours systématique des cases, on fait une *recherche* déterministe : chaque nouvelle information – les observations dans une case examinée – permet de tirer des conclusions sur les cases voisines. La *topologie* du fil joue un rôle important pour tirer ces conclusions : le fil forme des boucles fermées, et chaque zone séparée par deux droites (comme les cases de cet exercice) contient soit un nombre pair de segments, soit le virage à l'extrémité d'une boucle.

Les propriétés topologiques décrivent des structures formées par l'arrangement et la connexion d'éléments – indépendamment des tailles, distances et angles. il ne s'agit donc pas de connaître la longueur ou l'orientation d'un objet, mais de savoir comment les objets sont reliés ensemble et combien il y a de passages ou de chemins.

L'interprétation topologique d'un parcours systématique n'est pas seulement une réflexion intéressante, mais a aussi beaucoup d'applications en informatique :

Par exemple, un réseau routier peut être considéré comme un système de points et de connexions – les rues et les croisements. Une telle structure aide les algorithmes à trouver des chemins, reconnaître des déviations et déterminer l'itinéraire le plus rapide d'un point à un autre.

Un autre exemple est la recherche de problèmes dans les réseaux électriques : les circuits parallèles, boucles ou interruptions montrent souvent où se trouve le problème même sans mesure exacte, simplement en observant la structure logique du réseau.

Mots clés et sites web

- Topologie : `https://fr.wikipedia.org/wiki/Topologie`
- Algorithme de recherche : `https://fr.wikipedia.org/wiki/Algorithme_de_recherche`

13. Couronne de l'Avent

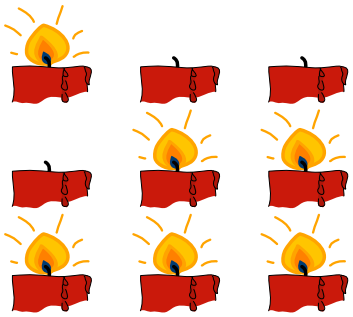
Il existe une tradition qui consiste à allumer des bougies lors des quatre dimanches de l'Avent : une bougie le premier dimanche, deux bougies le deuxième dimanche, et ainsi de suite.

Chris adore cette tradition. Ses quatre bougies ont la même taille avant d'avoir été allumées. Pour passer un Noël magnifique, Chris aimerait que ses bougies aient aussi toutes la même taille après le dernier dimanche de l'Avent. Pour cela, il devrait allumer chaque bougie le même nombre de fois.

Malheureusement, Chris ne trouve pas comment passer un Noël magnifique.

Si la tradition ne concernait que 3 dimanches (et 3 bougies), ce serait possible : Chris allumerait chaque bougie deux fois.

1er dimanche
2^{e} dimanche
3^{e} dimanche

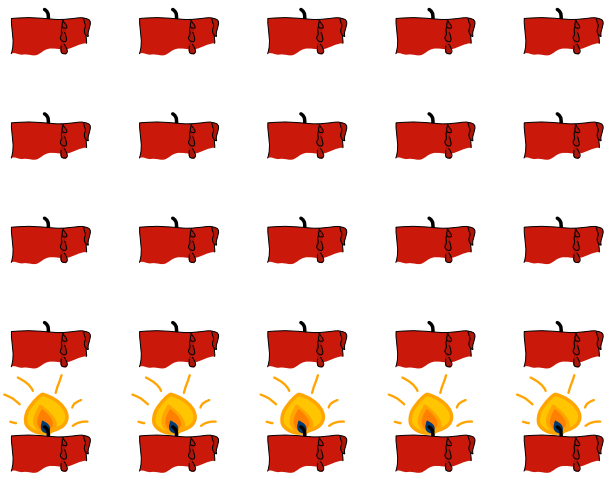


Ce serait aussi possible avec 5 dimanches (et 5 bougies).

Montre à Chris comment allumer chaque bougie le même nombre de fois.

Nous avons déjà allumé les bougies pour le cinquième dimanche.

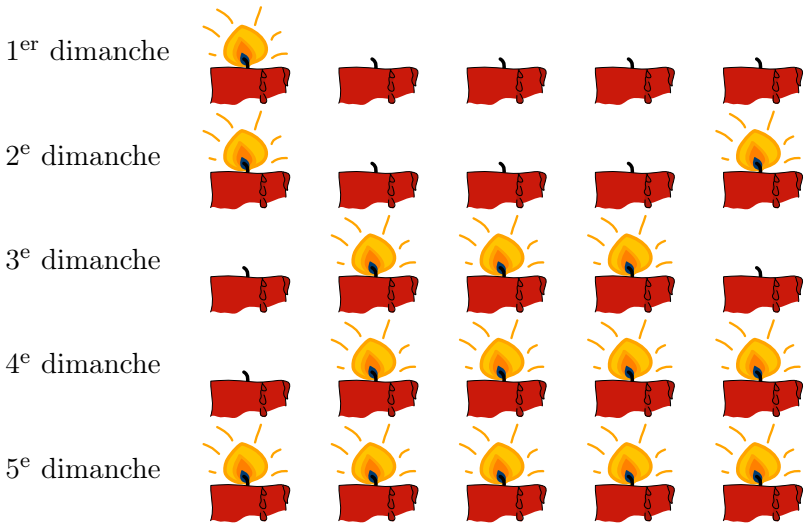
1er dimanche
2^{e} dimanche
3^{e} dimanche
4^{e} dimanche
5^{e} dimanche



Solution

Voici une bonne réponse :

Chris peut allumer les bougies comme cela pour qu'elles soient toutes allumées le même nombre de fois :



Il y a beaucoup de bonnes réponses possibles. Dans tous les cas, on peut procéder comme ceci :

1. On regroupe les dimanches par paires de manière à ce que leurs numéros additionnés donnent le nombre total de dimanches ; pour cinq dimanches, ce sont les paires 1 et 4 ainsi que 2 et 3.
2. Pour chacune de ces paires, on allume les bougies de manière à ce que les ensembles de bougies allumées à chacun des deux dimanches soient disjoints – par exemple la bougie 1 pour le premier dimanche et les bougies 2 à 5 pour le quatrième dimanche (si on considère chaque bougie comme un bit avec les valeurs 1 = allumée et 0 = éteinte, les suites de bits pour les deux dimanches d'une paire doivent être complémentaires). Comme cela, chaque bougie est allumée une fois l'un des dimanches de chaque paire.
3. En plus de cela, chaque bougie est allumée encore une fois le cinquième dimanche.

Toutes les bougies sont donc allumées le même nombre de fois.

C'est de l'informatique !

Au premier coup d'œil, cet exercice du castor semble être un problème mathématique. On peut effectivement démontrer que le problème de Chris peut être résolu pour chaque nombre impair de dimanches (on peut constater que la stratégie décrite dans l'explication fonctionne pour tous les nombres impairs).

Cependant, si on veut savoir concrètement comment allumer les bougies, il faut écrire un algorithme qui détermine l'ordre d'allumage des bougies – ou, encore mieux, qui liste toutes les solutions possibles. Cet algorithme est basé sur l'explication de la solution ci-dessus. Si n est le nombre (impair) de dimanches :

1. Allume les n bougies le n-ième dimanche.
2. Pour $i = 1$ jusqu'à $(n-1)/2$:
   a) Le dimanche i, allume les i premières bougies, et les dernières $n-i$ bougie le dimanche $n-i$.

On note que le point 2a de cet algorithme peut être effectué de toutes les manières satisfaisant les conditions décrites dans le point 2 de l'explication ci-dessus.

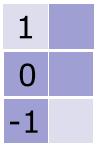
Le développement d'algorithmes pour la résolution de problèmes est l'une des tâches les plus importantes des informaticiens et informaticiennes. Si un algorithme a des bases mathématiques solides, il est plus facile de démontrer qu'il a les propriétés désirées que sans de telles bases. Pour l'algorithme ci-dessus, on peut démontrer qu'il fonctionne pour tous les nombres de dimanches impairs.

Mots clés et sites web

- Algorithme : `https://fr.wikipedia.org/wiki/Algorithme`
- Démonstration : `https://fr.wikipedia.org/wiki/Démonstration_(logique_et_mathématiques)`

14. Filtre d'image

Les images numériques sont souvent composées de pixels. Sandra crée des *cartes de teinte* pour de telles images pixelisées. Pour cela, elle commence par ajouter aux images un cadre de pixels blancs. Ensuite, elle détermine une valeur de teinte pour chaque pixel de l'image comme suit :

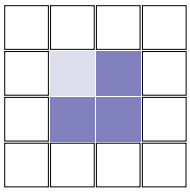


1 si le pixel est plus clair que son voisin de droite ;

0 si le pixel a la même teinte que son voisin de droite ;

-1 si le pixel est plus foncé que son voisin de droite.

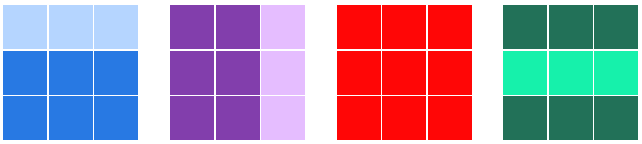
Voici une image composée de 4 pixels (plus le cadre de pixels blancs) et la carte de teinte correspondante.



1	-1
0	-1

Tu vois ci-dessous 4 images de 9 pixels chacune. Trois d'entre elles ont la même carte de teinte.

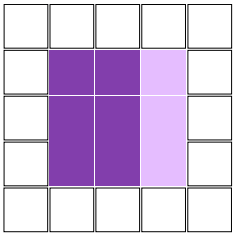
Quelle est l'image qui a une carte de teinte ***différente*** *?*



A) B) C) D)

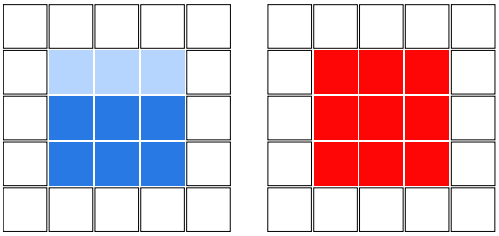
Solution

La bonne réponse est B :



0	-1	-1
0	-1	-1
0	-1	-1

Les trois autres images ont toutes la même carte de teinte :



0	0	-1
0	0	-1
0	0	-1

Pour trouver la bonne réponse, on peut déterminer la carte de teinte de chacune des quatre images et les comparer. On peut aussi observer que les cartes de teinte ne considèrent que la différence de teinte entre les pixels sur la même ligne, donc horizontalement. Comme les trois images A, C et D n'ont pas de différence de teinte sur l'axe horizontal, leur carte de teinte doit être la même.

C'est de l'informatique !

En informatique, les images peuvent être représentées dans plusieurs formats différents. De manière générale, les images sont enregistrées soit sous forme d'*image vectorielle* ou d'*image matricielle*. Ces dernières sont des matrices de *pixels* (de l'anglais « picture element »). Dans le cas le plus simple, chaque pixel peut être noir ou blanc et peut alors être représenté par un seul bit prenant la valeur 0 ou 1. Les pixels de couleur sont représentés par plusieurs valeurs qui quantifient, par exemple, la proportion des couleurs rouge, vert et bleu dans la couleur du pixel en question.

La carte de teinte de cet exercice du castor se rapproche du concept de la *convulotion* d'une image avec un *filtre*. Suivant le filtre utilisé, différentes propriétés de l'image peuvent être mises en évidence par une telle convolution, comme par exemple les angles, les arêtes ou les zones de teinte unie. Cela facilite l'interprétation des informations contenues dans l'image par les ordinateurs.

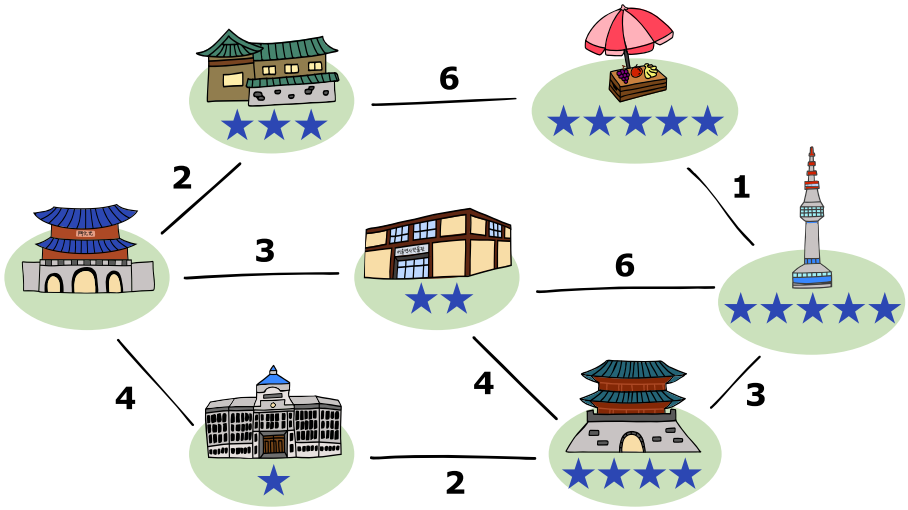
Les *réseaux neuronaux convolutifs* (CNN) utilisent le concept de convolution pour la reconnaissance des images et font souvent partie des systèmes d'intelligence artificielle. Pour reconnaître des objets complexes dans une image, un CNN apprend à élaborer ses propres filtres avec lesquels traiter l'image. Cela lui permet, par exemple, de différencier des images de chats d'images de chiens ou de détecter des tumeurs en imagerie médicale.

Mots clés et sites web

- Vision par ordinateur : `https://fr.wikipedia.org/wiki/Vision_par_ordinateur`
- Pixel : `https://fr.wikipedia.org/wiki/Pixel`
- Convolution : `https://fr.wikipedia.org/wiki/Produit_de_convolution`
- Filtre : `https://fr.wikipedia.org/wiki/Noyau_(traitement_d'image)`
- Réseau neuronal convolutif :
  `https://fr.wikipedia.org/wiki/Réseau_neuronal_convolutif`

15. Visite de Séoul

À Séoul, en Corée, il y a des bus qui relient les sites importants pour les touristes. L'image montre les sites les plus importants de Séoul. Plus un site est apprécié, plus il a d'étoiles. Les lignes montrent les connexions par bus. La longueur de chaque ligne en kilomètres est notée à côté de la ligne.



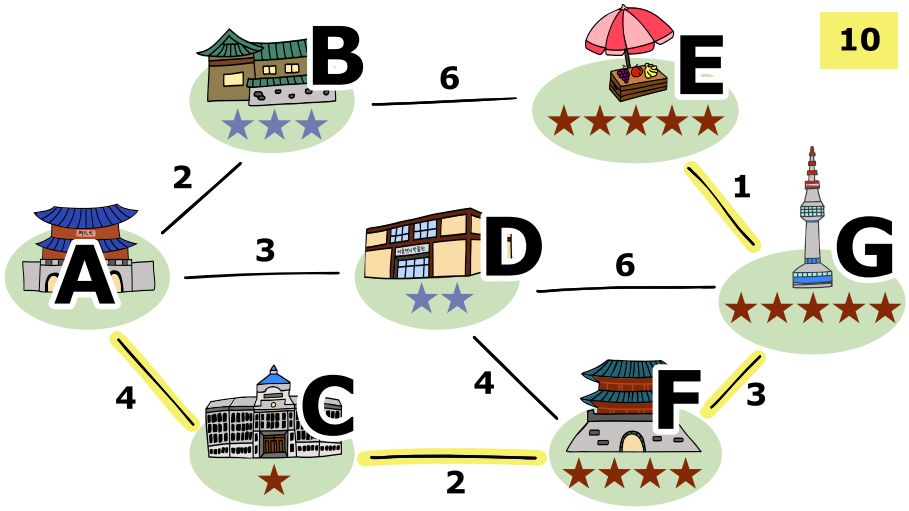
Lotte commence par visiter le palais  . Depuis là, elle aimerait aller visiter d'autres sites en bus. Elle a un billet avec lequel elle peut rouler au maximum 10 kilomètres. Elle aimerait l'utiliser pour visiter des sites avec le plus grand nombre d'étoiles possible au total. Elle ne visite un endroit qu'une seule fois et ne doit pas revenir au palais.

Quelles connexions Lotte doit-elle emprunter pour récolter le plus grand nombre d'étoiles ?

Solution

Pour Lotte, nous voulons trouver un itinéraire de bus qui parte du palais, fasse au maximum 10 kilomètres et passe par les sites ayant le plus d'étoiles possible. Pour déterminer le meilleur itinéraire, nous devons prendre la distance des sites au palais et leur nombre d'étoiles en considération.

Nous commençons par assigner les lettres A à G aux sites.



Nous pouvons maintenant décrire un arrêt sur un site par :

- la lettre du site,
- la distance totale du site au départ (A) et
- le nombre total d'étoiles récoltées sur le chemin jusqu'à ce site.

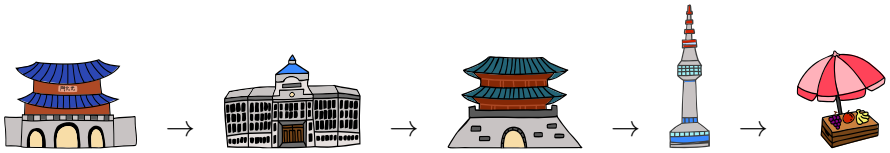
Nous appelons un itinéraire *valide* s'il fait 10 km au maximum. Un itinéraire valide serait par exemple :

- de A à B : B est à 2 km de A et a 3 étoiles ; cet arrêt peut être décrit par B(2,3) ;
- de B à E : la distance augmente de 6 km pour atteindre 8 km en tout, et le nombre d'étoiles augmente de 5 pour atteindre 8. L'arrêt est donc décrit par E(8,8) ;
- et finalement de E à G : on ajoute 1 km et 5 étoiles, et cet arrêt est décrit par G(9, 13).

Voici toutes les routes valides :

```
A → B(2,3) → E(8,8) → G(9,13)
A → D(3,2) → G(9,7) → E(10,12)
A → D(3,2) → F(7,6) → G(10,11)
A → D(3,2) → F(7,6) → C(9,7)
A → C(4,1) → F(6,5) → D(10,7)
A → C(4,1) → F(6,5) → G(9,10) → E(10,15)
```

La dernière route valide est la meilleure, car son dernier arrêt E(10,15) a le plus grand nombre d'étoiles. Lotte doit donc suivre l'itinéraire suivant avec son ticket de métro pour récolter le maximum d'étoiles :



C'est de l'informatique !

Le but de Lotte dans cet exercice du castor est de récolter le plus d'étoiles possible le long de son chemin. Elle cherche donc à *optimiser* une valeur – le nombre d'étoiles – en trouvant le plus grand nombre possible. Lotte a donc un *problème d'optimisation*. La solution de son problème est contrainte par son ticket de bus qui limite la longueur de son chemin.

En informatique, il existe de nombreuses méthodes pour résoudre les problèmes d'optimisation, qu'ils soient avec ou sans contraintes. Des outils informatiques sont donc souvent utilisés pour résoudre ce type de problèmes. Les systèmes de navigation sont utiles pour déterminer des itinéraires optimaux, et ils permettent souvent aux utilisateurs de changer la valeur à optimiser : l'itinéraire doit-il être le plus court possible ou consommer le moins d'énergie ? Des contraintes peuvent aussi être spécifiées, par exemple, on peut éviter les autoroutes, les routes à péages ou les ferrys.

Les méthodes informatiques pour trouver des itinéraires utilisent des structures de données qui peuvent être représentées de manière semblable au réseau de bus de cet exercice : des *graphes*. Ceux-ci sont composés d'un ensemble de *nœuds* et d'un ensemble de paires de nœuds, les *arêtes*. Les graphes permettent de modéliser les relations entre des objets ; par exemple les lignes de transports entre des endroits. Les distances peuvent être associées aux arêtes en tant que *poids*. En informatique, il existe beaucoup de méthodes pour résoudre les problèmes dont les données sont sous forme de graphe – par exemple le *parcours en profondeur* avec lequel nous avons résolu le problème de Lotte.

Mots clés et sites web

- Optimisation : `https://fr.wikipedia.org/wiki/Optimisation_(mathématiques)`
- Théorie des graphes : `https://fr.wikipedia.org/wiki/Théorie_des_graphes`
- Parcours en profondeur : `https://fr.wikipedia.org/wiki/Algorithme_de_parcours_en_profondeur`

16. Lacs de montagne

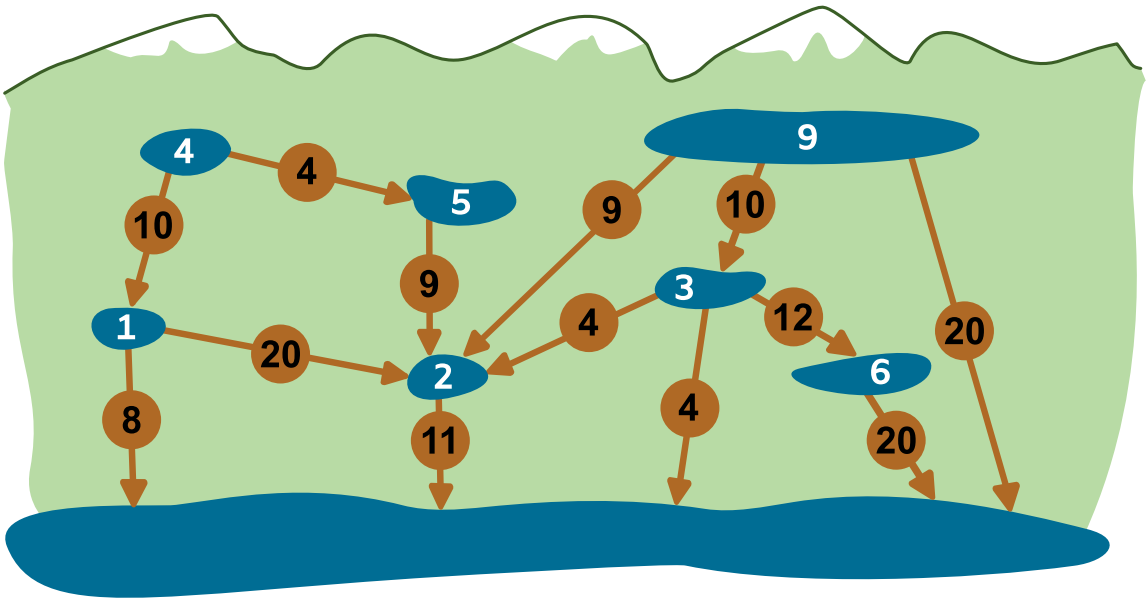
Dans la montagne au-dessus du réservoir du barrage, il y a plusieurs petits lacs naturels. Ils pourraient déborder s'il pleut beaucoup, ce qui est dangereux. C'est pour cela que des canaux doivent être construits entre certains lacs. Ces canaux doivent amener toute l'eau des lacs en trop dans le réservoir du barrage. Leur construction doit coûter le moins cher possible.

Chaque lac de montagne a un nombre qui indique combien d'eau en surplus doit en être déviée.

Une flèche dénote chaque endroit auquel un canal peut être construit. Elle montre la direction de l'eau, et le nombre indique la capacité du canal (donc combien d'eau il peut transporter) ainsi que son prix.

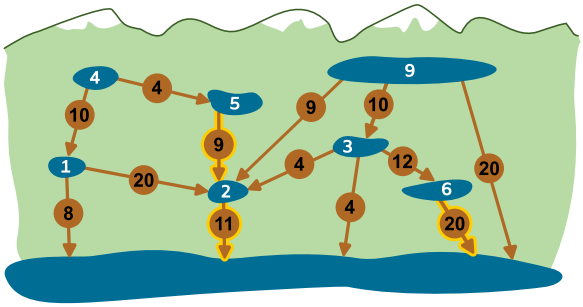
Fais attention : lorsqu'un canal transporte l'eau d'un lac de montagne à un autre, l'eau en trop des deux lacs s'accumule dans le deuxième lac.

Où faut-il construire des canaux ?

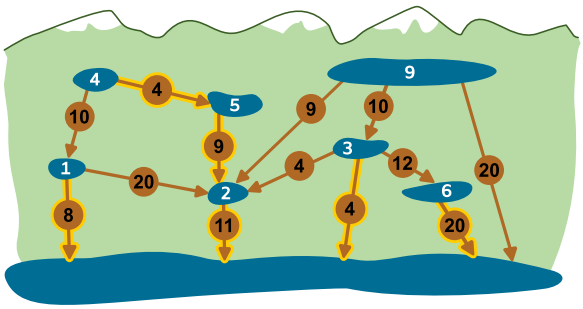


Solution

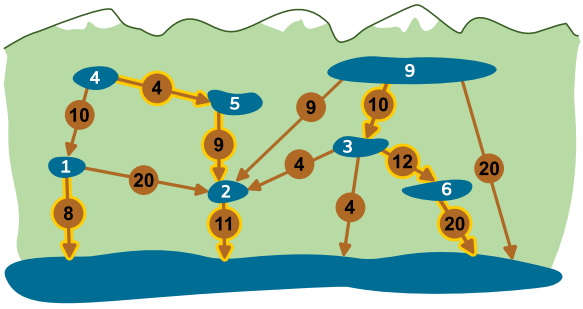
Voici la bonne réponse :



Pour certains lacs, il n'y a qu'un seul endroit auquel un canal peut être construit. Il s'agit des lacs avec la quantité d'eau en trop 2 en bas au centre (le lac 2 en bref), 5 et 6. Un canal *doit* donc être construit à ces endroits. Ces canaux ont une capacité assez grande pour ces trois lacs, même si l'on prend en compte que l'eau en trop du lac 5 doit aussi être déviée par le lac 2 et qu'un surplus de 7 doit donc être dévié. La construction de ces canaux coûte 11 + 9 + 20 = 40.



Les lacs 1 et 4 ont deux endroits de construction possible chacun. (a) Si le canal de 4 à 5 est construit, le canal du lac 2 vers le grand lac doit pouvoir contenir 4 + 5 + 2 = 11 de surplus d'eau. Il est alors plein, ce qui signifie que l'eau du lac 1 ne peut pas être déviée dans le lac 2, mais doit être déviée dans le lac du barrage. Le coût total est 4 + 8 = 12. (b) L'autre possibilité est de construire un canal entre les lacs 4 et 1. Si l'on construit ensuite le canal du lac 1 au lac 2, il y aurait 4 + 1 + 7 = 12 de surplus dans le lac 2, ce qui est plus que la capacité du canal du lac 2 vers le lac du barrage. Dans ce cas, il faut donc construire le canal du lac 1 directement vers le lac du barrage. Le coût total est 10 + 8 = 18, donc plus que l'alternative.



Il reste les lacs 3 et 9. (a) Le lac 9 ne peut pas être dévié dans le lac 2, parce que cela dépasserait la capacité du canal du lac 2 vers le lac du barrage (même sans le canal du lac 4 au lac 5). (b) Si le lac 9 est dévié directement dans le lac du barrage, la solution la moins chère pour les lacs 3 et 9 coûte 20 + 4 = 24. (c) Si le lac 9 est dévié dans le lac 3, il y a un surplus de 12 qui ne peut être dévié que dans le lac 6. Le surplus de 18 peut être évacué par le canal vers le barrage déjà construit. Le coût pour les lacs 3 et 9 est alors de 10 + 12 = 22, ce qui est moins cher.

Les canaux choisis peuvent dévier toute l'eau en surplus dans le lac du barrage pour un coût minimal de $40 + 12 + 22 = 74$.

C'est de l'informatique !

Les lacs (naturels et du barrage) et les endroits auxquels les canaux peuvent être construits peuvent être représentés dans un *graphe*. Un graphe a des *nœuds* (ici, les lacs) et des arêtes (ici, les endroits où des canaux peuvent être construits) qui les relient. Les arêtes peuvent avoir une direction et un *poids*, comme la capacité des canaux dans cet exercice.

C'est très utile de représenter un problème à l'aide d'une structure connue (comme un graphe) lorsqu'il existe des algorithmes informatiques permettant de traiter ces structures pour y résoudre des problèmes. Dans le domaine des graphes, beaucoup de problèmes sont décrits et il existe un grand nombre d'algorithmes efficaces permettant de les résoudre. C'est le cas pour les problèmes de flot, comme le *problème du flot de coût minimum* qui ressemble au problème de cet exercice du castor.

Mots clés et sites web

- Théorie des graphes : `https://fr.wikipedia.org/wiki/Théorie_des_graphes`
- Graphe orienté : `https://fr.wikipedia.org/wiki/Graphe_orienté`
- Réseau de flot : `https://fr.wikipedia.org/wiki/Réseau_de_flot`
- Problème du flot de coût minimum : `https://fr.wikipedia.org/wiki/Problème_du_flot_de_coût_minimum`

17. Parking

Il y a 9 invités qui viennent en voiture à une fête. Devant la salle de fête, 9 voitures peuvent se parquer dans 3 colonnes avec 3 voitures les unes derrière les autres dans chaque colonne. Les invités arrivent dans l'ordre suivant :

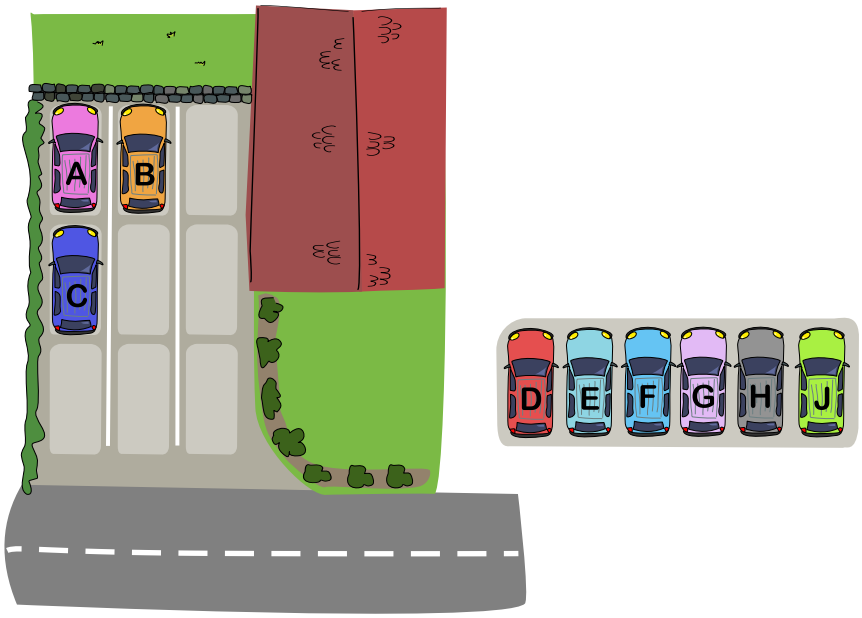
Anja, **B**rigitte, **C**lara, **D**avid, **E**lia, **F**rank, **G**abi, **H**ugo et enfin **J**ulia.

Pour se parquer, chaque invité choisit une colonne et s'y parque le plus en avant possible.

Les invités veulent partir de la fête dans l'ordre suivant :

Gabi, **D**avid, **B**rigitte, **E**lia, **J**ulia, **C**lara, **H**ugo, **A**nja et enfin **F**rank.

Les voitures d'**A**nja, **B**rigitte et **C**lara sont déjà parquées. Les autres invités se parquent les uns après les autres. Ils veulent se parquer de manière à ce que personne ne soit bloqué par une autre voiture au moment de partir.

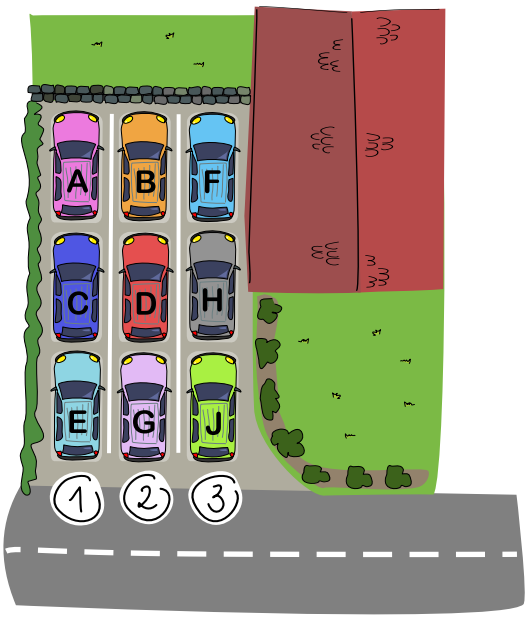


Montre aux invités comment ils peuvent se parquer !

Place les 6 voitures restantes dans les colonnes du parking. Tu dois considérer l'ordre d'arrivée ***et*** *l'ordre de départ.*

Solution

Voici la bonne réponse :



Nous commençons par observer que :

a) **F**rank part en dernier et doit donc se parquer tout en avant.

b) **G**abi part en premier et doit donc être sur une des trois places de derrière.

c) **B**rigitte part en troisième, donc **D**avid et **G**abi doivent se parquer derrière elle.

À partir de cela, nous pouvons faire les déductions suivantes :

- Le point a) signifie que **F**rank doit se parquer à côté de **B**rigitte, sur le devant de la colonne 3.
- **D**avid arrive après **A**nna, **B**rigitte et **C**lara. Le point c) signifie que **D**avid doit se parquer derrière **B**rigitte dans la colonne 2.
- Le point c) signifie que **G**abi doit se parquer derrière **D**avid dans la colonne 2.
- Il ne reste que la place derrière **C**lara dans la colonne 1 pour **E**mil. Il part en quatrième (après **G**abi, **D**avid et **B**rigitte) et doit donc être parqué tout à l'arrière d'une colonne. Quand il arrive, il n'y a que **F**rank dans la colonne 3 et **E**mil devrait se mettre sur la place du milieu ; il ne lui reste donc que la colonne 1 comme possibilité, vu que la colonne 2 est pleine.
- Il ne reste que les deux places dans la colonne 3 pour **H**ugo (au milieu) et **J**ulia (derrière). Heureusement que **J**ulia veut partir avant **H**ugo, car il n'y aurait pas de bonne solution sinon !

Comme chaque voiture ne peut se parquer qu'à une seule place, il n'y a qu'une seule solution.

C'est de l'informatique !

Les trois voitures parquées dans chaque colonne ne peuvent partir que dans l'ordre inverse de leur ordre d'arrivée. C'est comme avec une pile d'assiettes de laquelle on ne peut enlever que l'assiette du dessus sans risque.

Les *piles* existent aussi en informatique : ce sont des structures de données. Elles fonctionnent comme des piles d'assiettes ou les colonnes du parking de cet exercice du castor : l'opération *empiler* (*push* en anglais) ajoute un élément de données sur la pile. La fonction *top* nous dit quel est le dernier élément ajouté, et l'opération *dépiler* (*pop* en anglais) l'enlève de la pile. Il existe des modèles de calcul utilisant les piles en informatique théorique, par exemple les *automates à pile* : ils reconnaissent les *langages algébriques* utilisés par les ordinateurs, comme les langages de programmation ou le HTML.

Les systèmes d'exploitation des ordinateurs utilisent plusieurs piles (comme les trois colonnes de cet exercice) pour répartir des tâches entre plusieurs processeurs, par exemple. En ajoutant une deuxième pile à un automate à pile, celui-ci peut effectuer n'importe quel calcul, comme une machine de Turing. La deuxième pile fait toute la différence !

Mots clés et sites web

- Pile : `https://fr.wikipedia.org/wiki/Pile_(informatique)`
- Problème de partition : `https://fr.wikipedia.org/wiki/Problème_de_partition`
- Multiprocesseur : `https://fr.wikipedia.org/wiki/Multiprocesseur`
- Automate à pile : `https://fr.wikipedia.org/wiki/Automate_à_pile`
- Langage algébrique : `https://fr.wikipedia.org/wiki/Langage_algébrique`

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.

18. Encore plus de bois

Linus le castor a besoin de beaucoup de bois. Malheureusement, Linus ne rame pas encore très bien. Quand il commence à ramer, il continue toujours jusqu'à arriver juste devant le prochain rocher. Aide Linus à trouver un chemin pour ramasser le plus grand nombre possible de bûches.

Tu peux utiliser ces instructions :

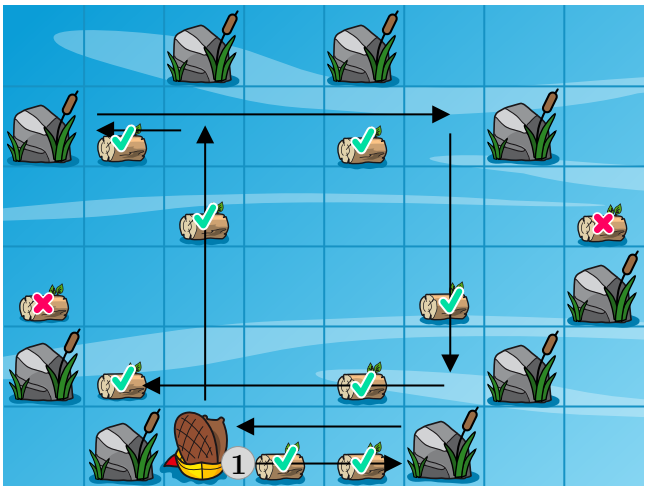
Instruction	**Effet**
`turnRight()` / `turnLeft()`	Linus se tourne sur place de 90 degrés sur sa droite ou sur sa gauche
`paddle()`	Linus rame et avance en ligne droite jusqu'à ce qu'un rocher l'arrête devant lui et ramasse toutes les bûches sur son chemin

Écris un programme pour ramasser le plus grand nombre possible de bûches.

Solution

La solution correcte est la suivante :

```
turnRight()
paddle()
turnRight()
turnRight()
paddle()
turnRight()
paddle()
turnLeft()
paddle()
turnRight()
turnRight()
paddle()
turnRight()
paddle()
turnRight()
paddle()
```



Comme nous pouvons avancer uniquement avec la commande `paddle()`, chaque mouvement vers l'avant s'arrête automatiquement juste avant un rocher. En raison de cette contrainte, il est possible de ramasser au maximum 8 bûches.

Si, en plus de la commande `paddle()`, nous disposions d'une autre commande permettant à Linus de faire un seul pas, il serait possible de ramasser davantage de bûches. Mais pour cela, il nous faudrait aussi une autre commande pour ramasser les bûches.

C'est de l'informatique !

Le problème comporte plusieurs parties : dans la première partie, nous nous occupons de la recherche d'un chemin. En mathématiques comme en informatique, ce type de situation est souvent décrit comme un problème de graphe. Ce type de problème est fréquemment lié à des problèmes d'optimisation. En effet, il ne s'agit pas seulement de trouver le bon chemin, mais aussi de déterminer le nombre maximal de bûches que l'on peut ramasser en suivant ce chemin. Des problèmes d'optimisation bien connus sont, par exemple, le problème du voyageur de commerce.

Dans la deuxième partie, consacrée à la programmation, nous nous demandons comment décrire précisément le chemin trouvé auparavant. Il est d'abord essentiel de comprendre le fonctionnement de la commande `paddle()`. Cette commande cache en réalité une boucle conditionnelle. Cela signifie que les mêmes instructions sont répétées tant qu'une certaine condition est remplie.

Avant chaque mouvement, Pétunia le castor vérifie si le chemin est encore libre. Si c'est le cas — c'est-à-dire s'il n'y a pas de rocher sur la case directement devant elle — elle avance d'une case et recommence le même processus de contrôle. Grâce à cela, il devient possible de parcourir n'importe quelle distance jusqu'à un rocher.

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`
- TSP : `https://fr.wikipedia.org/wiki/Problème_du_voyageur_de_commerce`

A. Auteur·e·s des exercices

James Atlas
Masiar Babazadeh
Wilfried Baumann
Leonardo Cavalcante
Špela Cerar
Andrew Csizmadia
Christian Datzko
Diane Dowling
Nora A. Escherle
Gerald Futschek
Silvan Horvath
Alisher Ikramov
Thomas Ioannou
Asterios Karagiannis
Blaž Kelvišar
David Khachatryan
Doyong Kim
Jihye Kim
Dong Yoon Kim
Vaidotas Kinčius
Stefan Koch
Lukas Lehner
Gunwoong Lim
Linda Mannila
Anna Morpurgo
A-Yeong Park
Suchan Park
Gabriela Gomez Pasquali
Elsa Pellet
Jean-Philippe Pellet
Zsuzsa Pluhár
Wolfgang Pohl
Pedro Ribeiro
Kirsten Schlüter
Margareta Schlüter
Dirk Schmerenbeck
Jacqueline Staub
Susanne Thut
Christine Vender
Florentina Voboril
Michael Weigend
Philip Whittington
Kyra Willekes
Hsu Sint Sint Yee

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/