



INFORMATIK-BIBER SCHWEIZ
CASTOR INFORMATIQUE SUISSE
CASTORO INFORMATICO SVIZZERA

Exercices et solutions 2025

Années HarmoS 11/12

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

Éditeurs:

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

010100110101011001001001
010000010010110101010011
0101001101001001010000101
00101101010101001101010011
0100100101001001001001001

SS!E

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





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 1^{er} décembre 2025.



Les exercices sont protégés par une licence Creative Commons Paternité – Pas d'Utilisation Commerciale – Partage dans les Mêmes Conditions 4.0 International. Les auteur·e·s sont cité·e·s en p. 76.



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.bebbras.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

Ont collaboré au Castor Informatique 2025	i
Préambule	iii
Table des matières	v
1. Pots de fleurs	1
2. De la feuille à la hutte	5
3. Archipel des castors	9
4. Jour de brouillard	13
5. Arbre généalogique	17
6. Message secret	19
7. Cerf-volant perdu	23
8. Couronne de l'Avent	27
9. Filtre d'image	31
10. Transport de farine	35
11. Noir et blanc	39
12. Adresses e-mail	43
13. Acide castorique	47
14. Transports publics	51
15. Lacs de montagne	55
16. Parking	59
17. Castor Jones	63
18. Session d'examens	67
19. Autour des rochers	73
A. Auteur-e-s des exercices	76
B. Partenaires académiques	77
C. Sponsoring	78



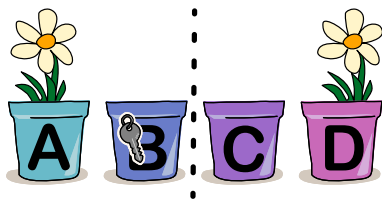
1. Pots de fleurs

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**.

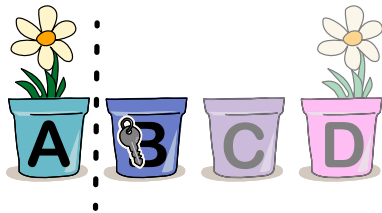
Une clé est cachée dans l'un des pots. Florian explique comment il fait pour trouver la clé.

«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 donc 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 combinant. On obtient ainsi toutes les $2 \times 2 \times 8 = 32$ solutions.



Colonne 1	Colonne 2	Colonne 3
 	 	   
 	 	   
		
		   
		   
		   
		   
		   
		   
		  
		   

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 une 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.

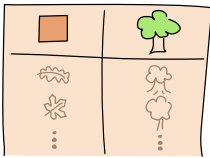



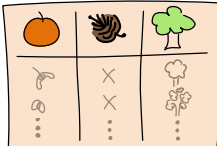



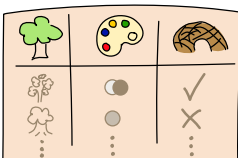



Mots clés et sites web

- Approche ascendante/descendante:
https://fr.wikipedia.org/wiki/Approches_ascendante_et_descendante
- Code: [https://fr.wikipedia.org/wiki/Code_\(information\)](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



2. 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.



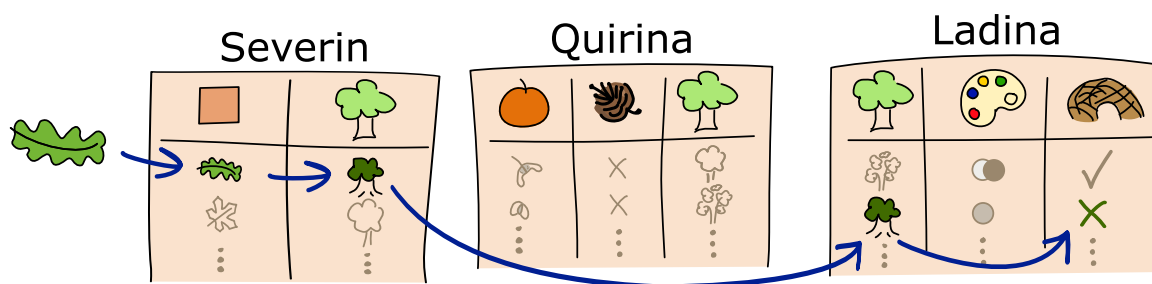
Solution

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.



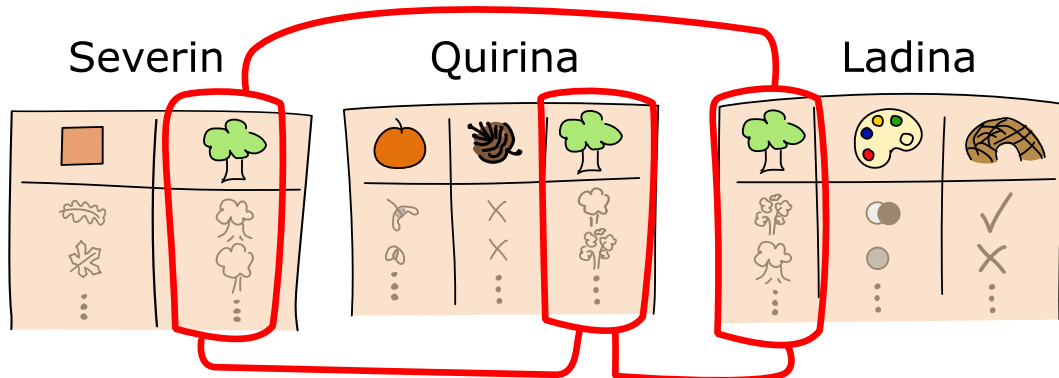
C'est de l'informatique !

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

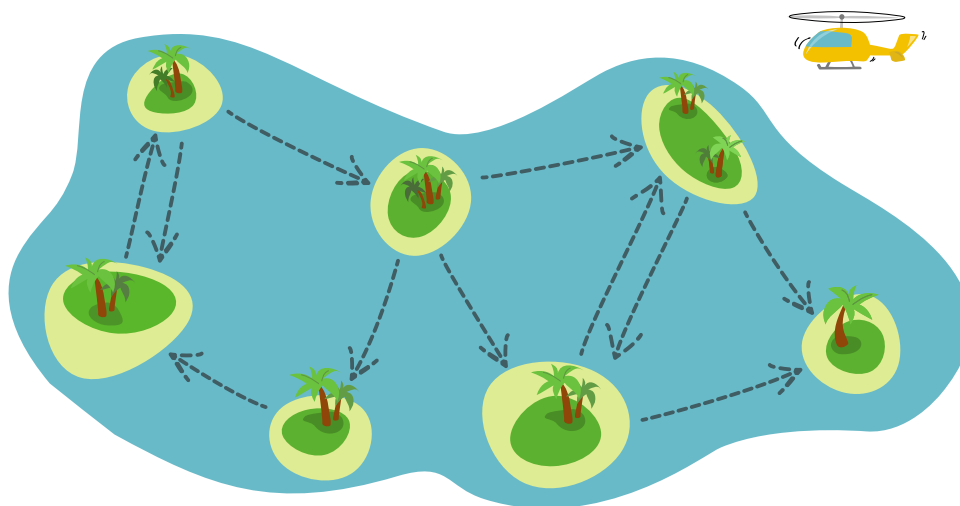
- Base de données relationnelle:
https://fr.wikipedia.org/wiki/Base_de_données_relationnelle
- Clé: [https://fr.wikipedia.org/wiki/Clé_\(structure_de_données\)](https://fr.wikipedia.org/wiki/Clé_(structure_de_données))
- SQL: <https://fr.wikipedia.org/wiki/SQL>
- Attribut: https://fr.wikipedia.org/wiki/Base_de_données#Attribut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





3. 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



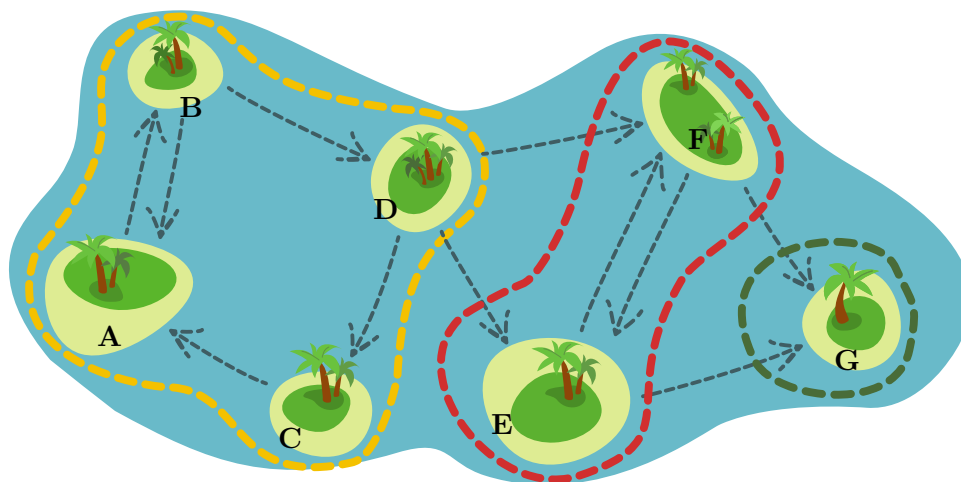
Solution

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



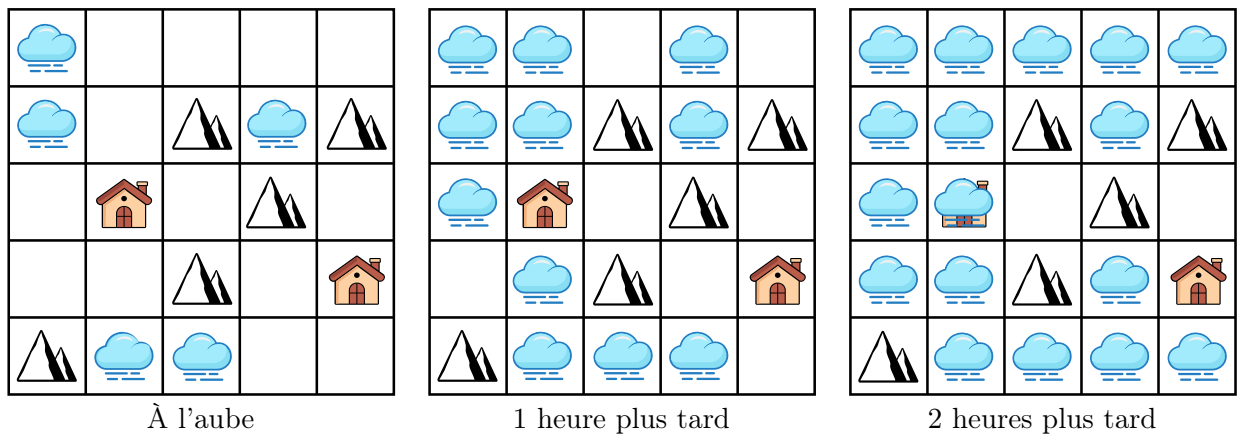


4. Jour de brouillard

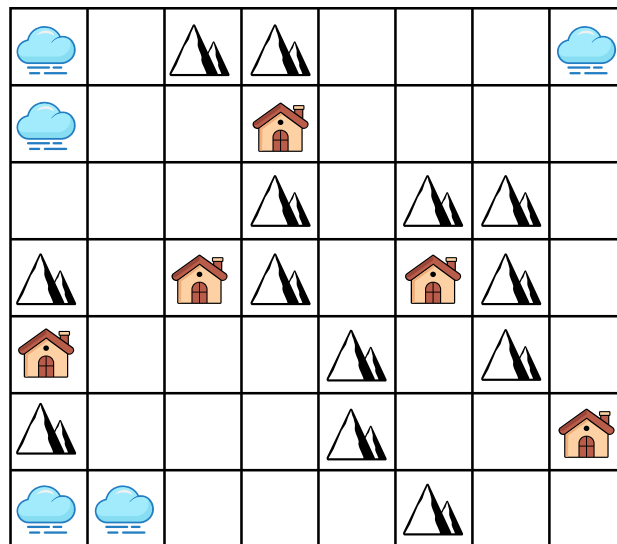
Aujourd'hui, il y a du brouillard au pays montagneux. Le brouillard ☁ s'étend d'heure en heure.

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 :



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 à 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.

	1			3	2	1	
	1	2	3 	4	3	2	1
1	2	3		5			2
	3	4 		6	7 		3
3 	2	3	4		8		4
	1	2	3		7	6	5 
		1	2	3		7	6

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 $n \times m$ 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, elle 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

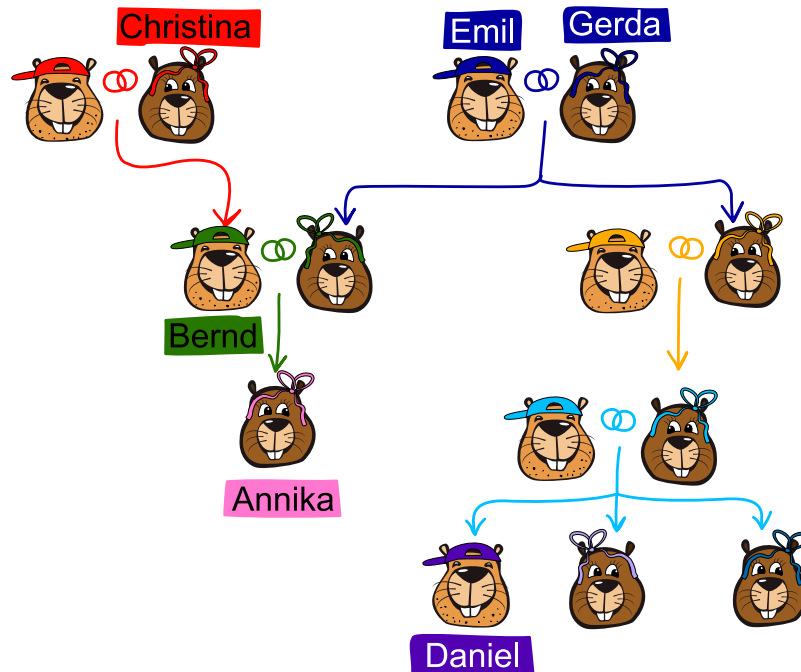
- Parcours en profondeur:
https://fr.wikipedia.org/wiki/Algorithme_de_parcours_en_profondeur
- Graphe connexe: https://fr.wikipedia.org/wiki/Graphe_connexe
- Algorithme de remplissage par diffusion:
https://fr.wikipedia.org/wiki/Algorithme_de_remplissage_par_diffusion





5. 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.

$$\text{père} \left(\text{mère} \left(\text{Annika} \right) \right) = \text{ } \left(\text{ } \left(\text{ } \left(\text{Daniel} \right) \right) \right)$$



Solution

Voici la bonne réponse :

$$\text{père} \left(\text{mère} \left(\text{Annika} \right) \right) = \text{père} \left(\text{mère} \left(\text{mère} \left(\text{Daniel} \right) \right) \right)$$

Nous commençons par le côté droit de l'équation et voyons qu'Emil est le père de la mère d'Annika, donc $\text{père}(\text{mère}(\text{Annika})) = \text{Emil}$. 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, $\text{mère}(\text{Daniel})$, est apparentée à Emil; pas le père de Daniel.
2. La mère de la mère de Daniel, donc la grand-mère de Daniel, $\text{mère}(\text{mère}(\text{Daniel}))$, est apparentée à Emil, car...
3. ... Emil est le père de cette grand-mère: $\text{Emil} = \text{père}(\text{mère}(\text{mère}(\text{Daniel})))$.

C'est de l'informatique !

Annika utilise ses abréviations $\text{père}(X)$ et $\text{mère}(X)$ 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 $\text{père}(\text{mère}(\text{mère}(\text{Daniel})))$, 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*.

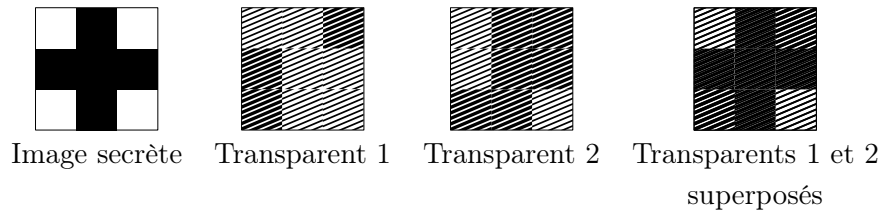
Mots clés et sites web



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









6. Message secret

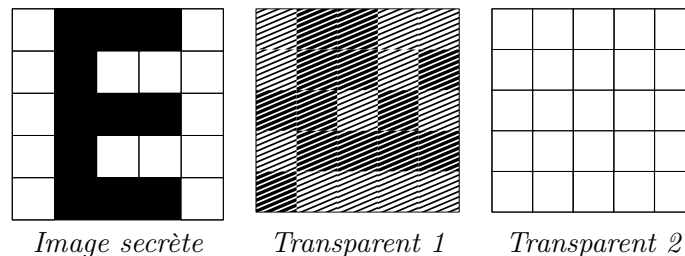
Une image secrète composée de pixels noirs  et blancs  doit être transmise de manière sûre. Pour cela, le service de messagerie crée deux images composées de pixels foncés  et clairs  sur des feuilles transparentes. L'image secrète n'est révélée que lorsque les deux feuilles transparentes sont superposées.



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 .

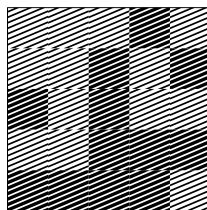
Le premier transparent a déjà été imprimé pour l'image ci-dessous. Détermine la couleur des pixels du deuxième transparent.





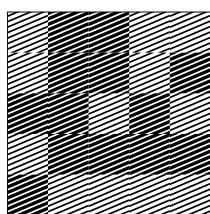
Solution

Voici la bonne réponse :

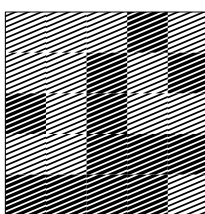


Sur cette image du transparent 2, chaque pixel a été imprimé en suivant la règle décrite plus haut – en fonction de l'image secrète et du transparent 1. L'image du transparent 2 n'est différente du transparent 1 qu'aux endroits où l'image secrète a un pixel noir.

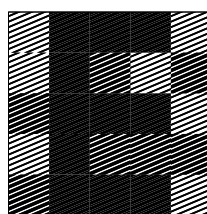
Tu peux voir ici comment la superposition du transparent 1 et de ce transparent 2 révèle l'image secrète :



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.



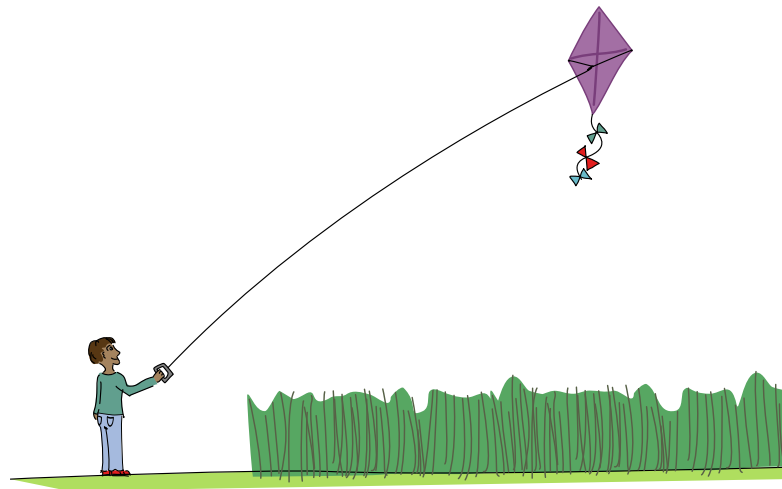
Mots clés et sites web

- 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





7. Cerf-volant perdu

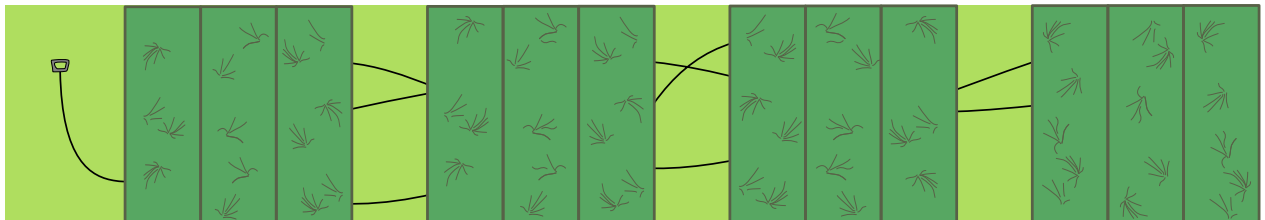


Pas de chance! Asterios a perdu son cerf-volant dans la prairie. Le fil s'est emmêlé dans les hautes herbes et c'est difficile de retrouver le cerf-volant.

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?

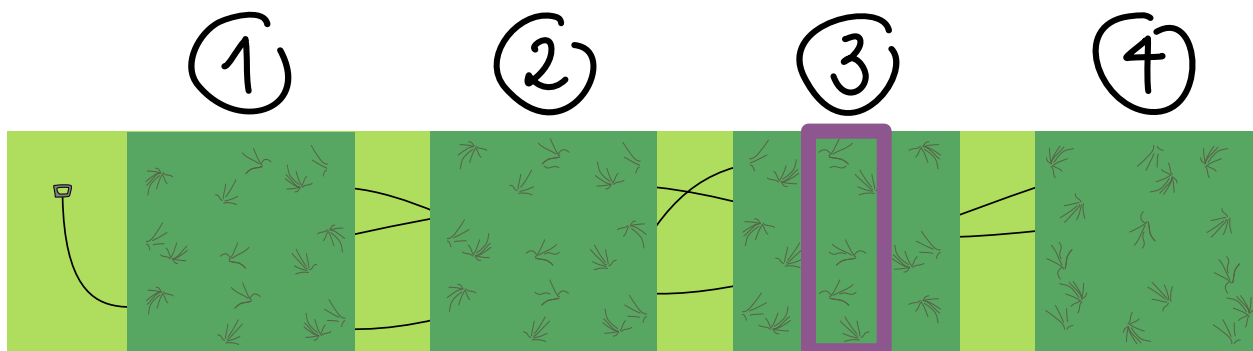




Solution

Voici la bonne réponse :

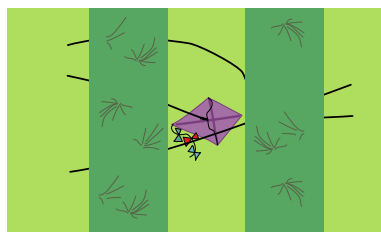
Asterios doit examiner la case encadrée en violet pour savoir exactement dans quelle case se trouve son cerf-volant.



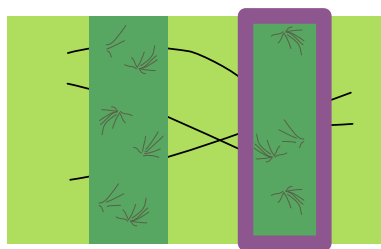
Tu peux trouver la solution en deux étapes. D'abord, tu détermines dans lequel des quatre blocs 1, 2, 3 ou 4 doit se trouver le cerf-volant. Pense au fait que le fil commence à gauche et que le cerf-volant se trouve à l'autre extrémité.

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.

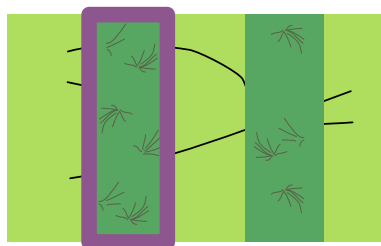
Lorsqu'Asterios examine la case au milieu du bloc 3, il peut tomber sur l'un des trois cas suivants :



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





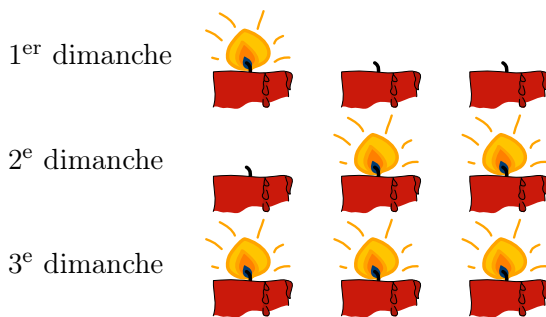
8. Couronne de l'Avent

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.

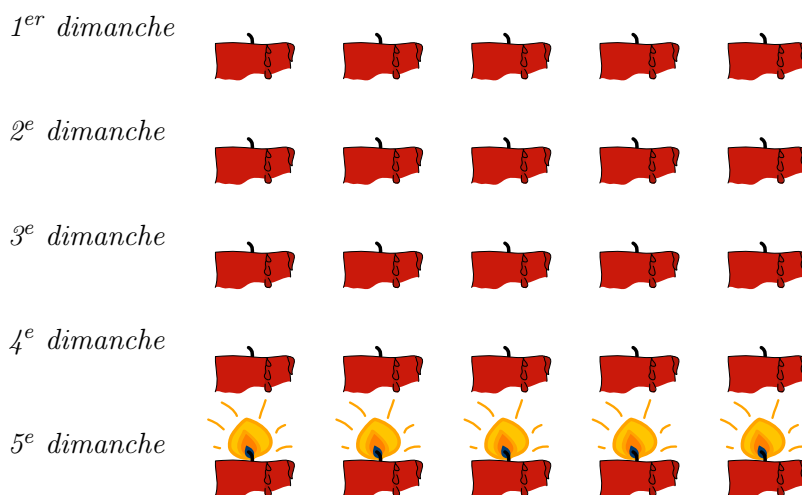
Si la tradition ne concernait que 3 dimanches (et 3 bougies), ce serait possible : Chris allumerait chaque bougie deux fois.



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.

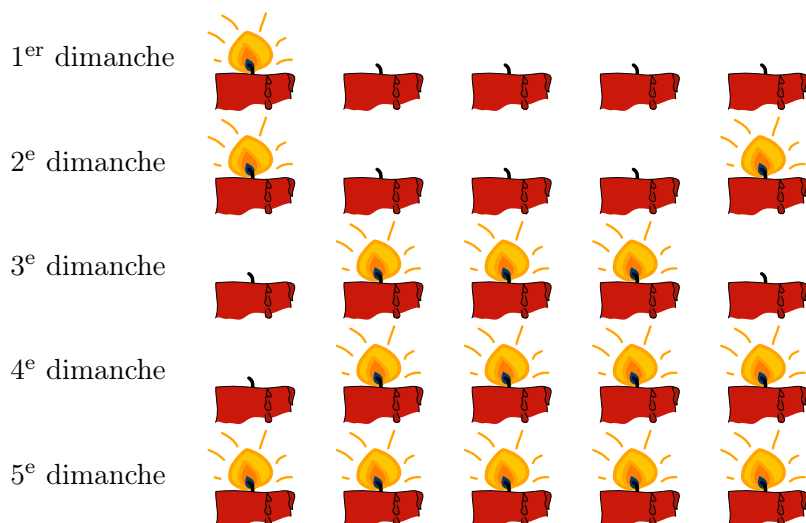




Solution

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_\(logique_et_mathématiques\)](https://fr.wikipedia.org/wiki/Démonstration_(logique_et_mathématiques))





9. 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 :

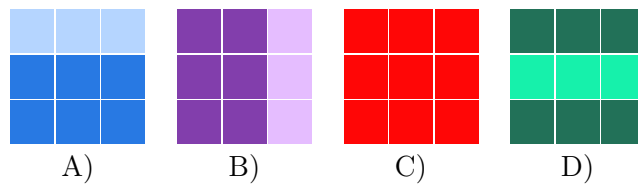
1		1 si le pixel est plus clair que son voisin de droite ;
0		0 si le pixel a la même teinte que son voisin de droite ;
-1		-1 si le pixel est plus foncé que son voisin de droite.

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.

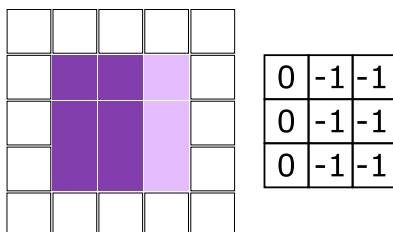
Quelle est l'image qui a une carte de teinte **différente** ?



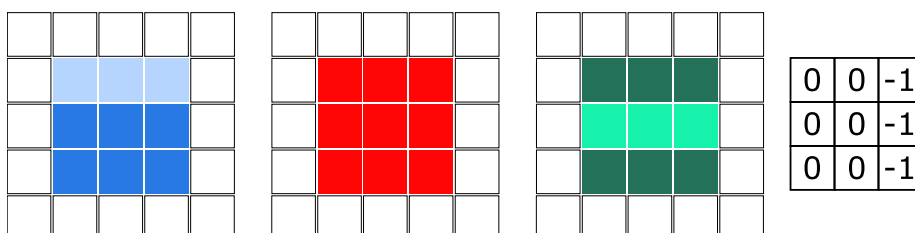


Solution

La bonne réponse est B:



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



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 *convolution* 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.



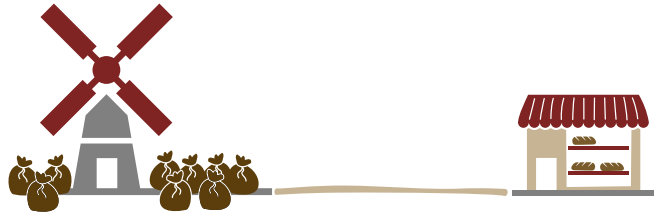
Mots clés et sites web

- 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_\(traitement_d'image\)](https://fr.wikipedia.org/wiki/Noyau_(traitement_d'image))
- Réseau neuronal convolutif:
https://fr.wikipedia.org/wiki/Réseau_neuronal_convolutif





10. Transport de farine



Albert et Marco travaillent dans une boulangerie. Ils doivent souvent aller chercher de la farine au moulin. Ils ne peuvent pas y aller en même temps, car l'un d'entre eux doit servir les clients dans la boulangerie.

Ils ne transportent pas la même quantité de farine à la même vitesse :



Albert transporte 13 kg de farine en 1 heure.



Marco transporte 5 kg de farine en 30 minutes.

Chacun doit se reposer pendant 30 minutes après 3 transports de farine. Ils peuvent servir des clients en se reposant.

Albert et Marco aimeraient transporter le plus de farine possible en 8 heures. C'est possible dans une seule des conditions suivantes. Laquelle ?

- A) Albert doit y aller en premier.
- B) Marco doit y aller en premier.
- C) Marco doit y aller en dernier.
- D) Albert ne doit pas y aller en dernier.
- E) Marco doit y aller une seule fois.



Solution

La bonne réponse est A.

Nous savons que :

- Albert transporte 13 kg par heure.
- Marco transporte 5 kg en 30 minutes, donc 10 kg par heure.
- L'un des deux doit toujours rester à la boulangerie.
- Chacun ne peut faire que 3 transports avant de devoir rester 30 minutes dans la boulangerie.
- L'autre peut transporter de la farine pendant que l'un est dans la boulangerie.

Albert transporte plus de farine par heure que Marco. Il devrait donc le faire le plus souvent possible.

Marco ne devrait transporter de farine que pendant qu'Albert se repose après 3 transports. De cette manière, ils peuvent transporter 44 kg de farine en 3,5 heures :

	1h	2h	3h	3,5h
Albert				
Mario				

44kg

Comme Albert est reposé après les 3,5 heures, ils peuvent répéter ce motif (A, A, A, M en bref) et transporter 88 kg de farine en 7 heures, puis Albert peut faire encore un transport. Ils transportent donc au maximum 101 kg de farine si Albert y va en premier.

	1h	2h	3h	4h	5h	6h	7h	8h	
Albert									
Mario									

101kg

Réponse B: Si Marco y va en premier, ils peuvent aussi transporter 44 kg durant les premières 3,5 heures, mais en suivant le motif M, A, A, A. Il reste encore 1 heure après une répétition de ce motif et 88 kg de farine. Comme Albert doit se reposer, c'est Marco qui doit y aller deux fois pendant cette heure et transporter 10 kg de farine. Ils ne peuvent donc transporter que $44 + 44 + 10 = 98$ kg de farine dans cette condition.



Réponses C et D: Si Marco y va en dernier, Albert ne peut pas y aller en dernier. Les réponses C et D sont donc équivalentes. Dans ces conditions, ils pourraient également transporter 88 kg de farine pendant les premières 7 heures, mais seulement 10 kg pendant la dernière heure, donc $44 + 44 + 10 = 98$ kg au maximum.

Réponse E: Si Marco n'y va qu'une seule fois, ils peuvent aussi transporter 44 kg de farine pendant les premières 3,5 heures. Par contre, pendant les 3,5 heures suivantes, Albert doit se reposer 30 minutes, et comme Marco n'y retourne pas, ils ne transportent que 39 kg de farine. Après cela, Albert peut y retourner, ce qui fait qu'ils transportent $44 + 39 + 13 = 96$ kg de farine dans cette condition.

C'est de l'informatique !

Si Albert et Marco veulent transporter le plus de farine possible pendant une certaine durée, ils doivent résoudre un problème d'ordonnancement et d'optimisation :

- Ils doivent planifier leurs trajets en respectant des contraintes (charge maximale et temps de repos).
- Ils veulent maximiser la quantité de farine transportée.

Les problèmes d'*ordonnancement* et d'*optimisation* sont fréquents dans la vie quotidienne. Leur aspect le plus important est l'*efficacité* : cela signifie simplement de faire le plus possible dans certaines conditions. Dans les usines, il faut produire le plus possible avec les personnes et machines disponibles ; dans les aéroports, le plus de passagers possibles doivent être servis aux guichets et portes, et ainsi de suite. Dès que les problèmes d'ordonnancement et d'optimisation deviennent plus complexes, des méthodes informatiques peuvent aider à les résoudre. De tels problèmes dans lesquels des conditions doivent être respectées sont aussi présents en informatique, car les processeurs doivent eux aussi être efficaces. Voici deux exemples : lors de l'exécution d'instructions, il faut prendre en considération que chaque unité logique ne peut effectuer qu'une instruction à la fois. Les instructions qui doivent lire des données de la mémoire doivent attendre l'entrée de ces données et ont ensuite un « temps de repos » – comme Albert et Marco dans cet exercice du castor.

Mots clés et sites web

- Optimisation : [https://fr.wikipedia.org/wiki/Optimisation_\(mathématiques\)](https://fr.wikipedia.org/wiki/Optimisation_(mathématiques))
- Ordonnancement : https://fr.wikipedia.org/wiki/Théorie_de_l'ordonnancement
- Ordonnancement en informatique :
[https://fr.wikipedia.org/wiki/Ordonnancement_\(informatique\)](https://fr.wikipedia.org/wiki/Ordonnancement_(informatique))





11. Noir et blanc

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

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

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

	B
	xBN
	xxNBN
	xNxBxNB

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

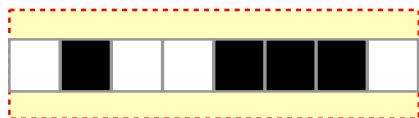




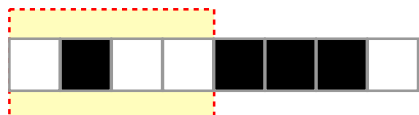
Solution

Voici la bonne réponse: **xxxBNBxNxNB**.

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



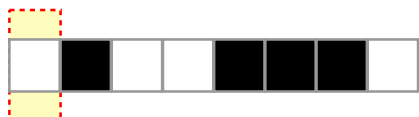
x - La suite contient des carrés noirs et des carrés blancs. L'algorithme écrit **x** et s'applique lui-même à la moitié de gauche de la suite.



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



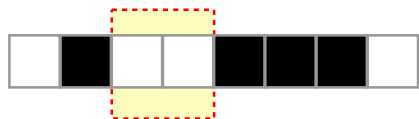
xxx - La suite contient des carrés noirs et des carrés blancs. L'algorithme écrit **x** et s'applique lui-même à la moitié de gauche de la suite.



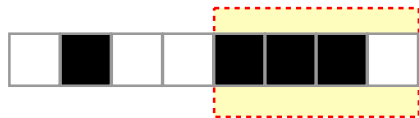
xxxB - Tous les carrés de la suite sont blancs. L'algorithme écrit **B** et a fini de traiter cette suite. Il traite maintenant la partie droite de la suite précédente.



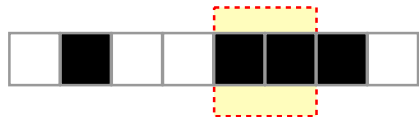
xxxBN - Tous les carrés de la suite sont noirs. L'algorithme écrit **N** et a fini de traiter cette suite. Il traite maintenant la partie droite de la suite précédente.



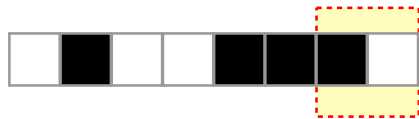
xxxBNB - Tous les carrés de la suite sont blancs. L'algorithme écrit **B** et a fini de traiter la moitié gauche de la suite complète. Il traite maintenant la moitié droite.



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



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



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



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



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



C'est de l'informatique !

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

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

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

Mots clés et sites web

- Diviser pour régner :
[https://fr.wikipedia.org/wiki/Diviser_pour_régner_\(informatique\)](https://fr.wikipedia.org/wiki/Diviser_pour_régner_(informatique))
- Récursivité : <https://fr.wikipedia.org/wiki/Récursivité>
- Compression de données : https://fr.wikipedia.org/wiki/Compression_de_données
- Quadtree : <https://fr.wikipedia.org/wiki/Quadtree>





12. Adresses e-mail

Les castors du service informatique ont besoin d'un système qui reconnaît si une suite de caractères est une adresse e-mail castor. Une adresse e-mail castor est faite de trois éléments :



	Nom	Explication	Valeurs possibles
①	Identifiant	N'importe quelle suite de caractères non vide avec des lettres minuscules et/ou des chiffres	0–9, a–z
②	Arobase		@
③	Nom du serveur		
③a	Nom de domaine	N'importe quelle suite de caractères non vide avec des lettres minuscules et/ou des chiffres	0–9, a–z
③b	Point		.
③c	Nom de domaine de premier niveau	N'importe quelle suite de caractères non vide avec des lettres minuscules et/ou des chiffres	0–9, a–z

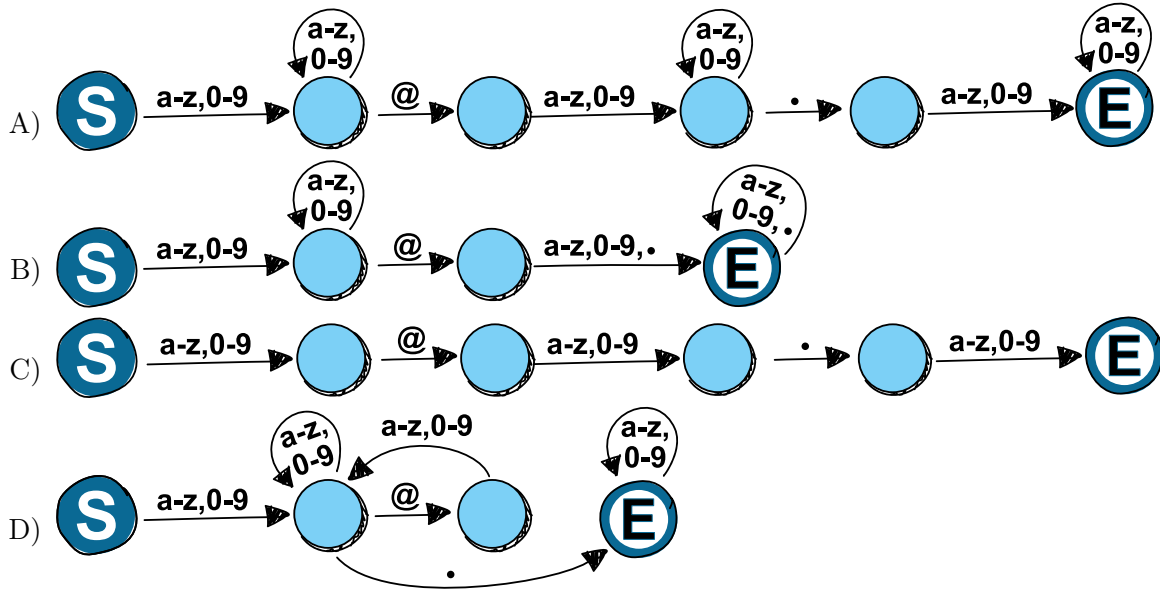
Une suite «non vide» est composée d'au moins un caractère. «bebra@sapin2.enaVal» est un exemple d'adresse e-mail castor.

Les castors du service informatique développent quatre systèmes et les représentent à l'aide de diagrammes composés de cercles et de flèches. Chaque cercle représente un état dans lequel le système peut être. Chaque flèche représente le changement à l'état suivant, état qui peut être le même que le précédent. L'annotation de la flèche nous dit à quel caractère ce changement d'état correspond.

Chaque système examine une suite de caractères un caractère après l'autre, de gauche à droite et se trouve dans l'état **S** au départ. L'état suivant dépend du caractère actuellement examiné : le système suit la flèche de changement qui correspond au caractère courant. Si le système se trouve dans l'état **E** à la fin de la suite de caractères, celle-ci a été reconnue comme adresse e-mail castor.



Seul un de ces systèmes identifie correctement toutes les adresses e-mail castor. Lequel ?

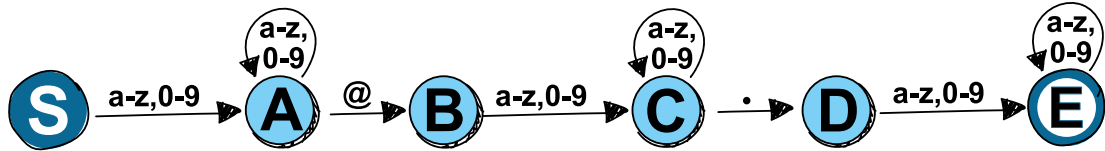




Solution

La bonne réponse est A.

Pour l'explication, nous attribuons une lettre à chaque état :



Le changement (appelé «transition» en informatique) de S à A reconnaît une lettre minuscule ou un chiffre, ce qui fait que l'identifiant ne peut pas être vide. La transition de A à A se fait avec n'importe quelle lettre minuscule ou chiffre. La transition de A à B reconnaît l'arobase (@). La transition de B à C reconnaît une lettre minuscule ou un chiffre, ce qui fait que le nom de domaine ne peut pas être vide. La transition de C à C se fait avec n'importe quelle lettre minuscule ou chiffre. La transition de D à D reconnaît un point. La transition de D à E reconnaît une lettre minuscule ou un chiffre, ce qui fait que le nom de domaine de premier niveau ne peut pas être vide. La transition de E à E se fait avec n'importe quelle lettre minuscule ou chiffre. Mis ensemble, cela correspond exactement à la définition de l'adresse e-mail castor de la donnée de l'exercice.

La réponse B est fausse. Ce système reconnaît aussi «castor@internet» ou «castor@.internet.com» par exemple, mais ce ne sont pas des adresses e-mail castor.

La réponse C est également fausse. Ce système ne reconnaît que les adresses e-mail dont l'identifiant, de domaine et de domaine de premier niveau ne comportent qu'un seul caractère chacun.

La réponse D est également fausse. Ce système reconnaît «castor@hutte@enamont.», par exemple, et ce n'est pas une adresse e-mail castor.

C'est de l'informatique !

Les systèmes utilisés dans cet exercice sont des *automates finis*, plus précisément des *reconnaisseurs*. Ils sont constitués d'un ensemble d'états, l'un d'eux étant l'*état initial* et d'autres les *états finaux*. Lorsque l'automate se trouve dans un état final à la fin de la suite de caractère, le mot examiné est reconnu. L'automate détermine l'état suivant à l'aide de transitions qui dépendent de l'état courant.

Pour chaque automate fini existe une *grammaire régulière* permettant de générer exactement tous les mots reconnus par l'automate (le *langage régulier*) correspondant. Des analogies entre grammaires et langages existent aussi pour d'autres reconnaisseurs; elles sont classifiées dans la *hiérarchie de Chomsky* pour les *langages formels*.

De plus, les langages reconnus par les automates finis peuvent être décrits à l'aide d'*expressions régulières*. L'expression régulière des adresses e-mail castor définies dans cet exercice est :

$[a-z0-9]^+@[a-z0-9]^+.[a-z0-9]^+$



Les automates finis et les expressions régulières ont beaucoup d'applications dans la vérification de suites de caractères: est-ce une adresse e-mail ou de site internet correctement formulée? Le mot de passe contient-il les caractères nécessaires? Grâce à leur structure simple, ils sont aussi souvent utilisés dans les systèmes intégrés devant utiliser peu d'électricité, par exemple ceux alimentés par une seule batterie et devant durer plusieurs années.

Mots clés et sites web

- Automate fini: https://fr.wikipedia.org/wiki/Automate_fini
- Grammaire régulière: https://fr.wikipedia.org/wiki/Grammaire_régulière
- Langage régulier: https://fr.wikipedia.org/wiki/Langage_rationnel
- Expression régulière: https://fr.wikipedia.org/wiki/Expression_régulière
- Hiérarchie de Chomsky: https://fr.wikipedia.org/wiki/Hiérarchie_de_Chomsky



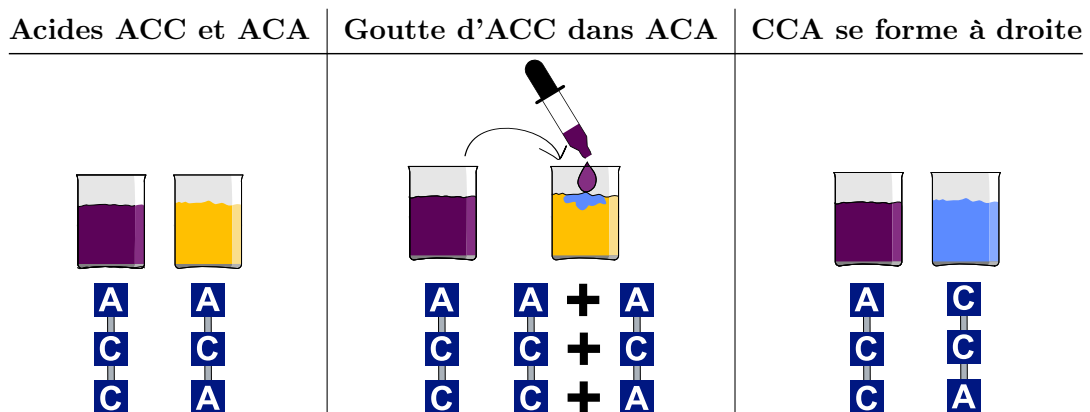
13. Acide castorique

Les acides castoriques sont des substances chimiques spéciales : leurs molécules sont faites de suites de trois éléments qui peuvent chacun être de type A ou C. Cette suite est utilisée pour nommer la molécule d'acide castorique, par exemple ACA.

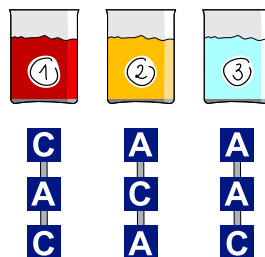
Lorsqu'un acide castorique est ajouté à un autre, un nouvel acide castorique se forme. Les éléments des deux acides de départ forment le nouvel acide d'après les règles suivantes :



Voici un exemple : si on ajoute une goutte d'ACC à de l'ACA, on obtient du CCA :

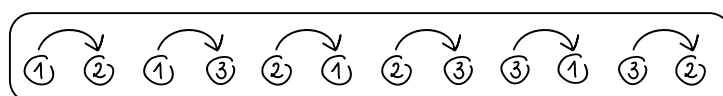


Au laboratoire, il y a trois récipients, 1, 2 et 3, contenant les acides castoriques CAC, ACA et AAC :



Une telle instruction te permet spécifier qu'une goutte de l'acide castorique d'un récipient (par exemple le récipient 2) doit être ajoutée à l'acide castorique d'un autre récipient (par exemple le récipient 3).

Il y a 6 instructions ci-dessous. Utilises-en le moins possible pour échanger les acides castoriques des récipients 1 et 3 : à la fin, AAC doit être dans le récipient 1, ACA rester dans le récipient 2 et CAC être dans le récipient 3.

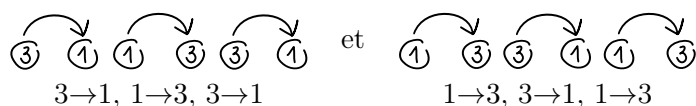




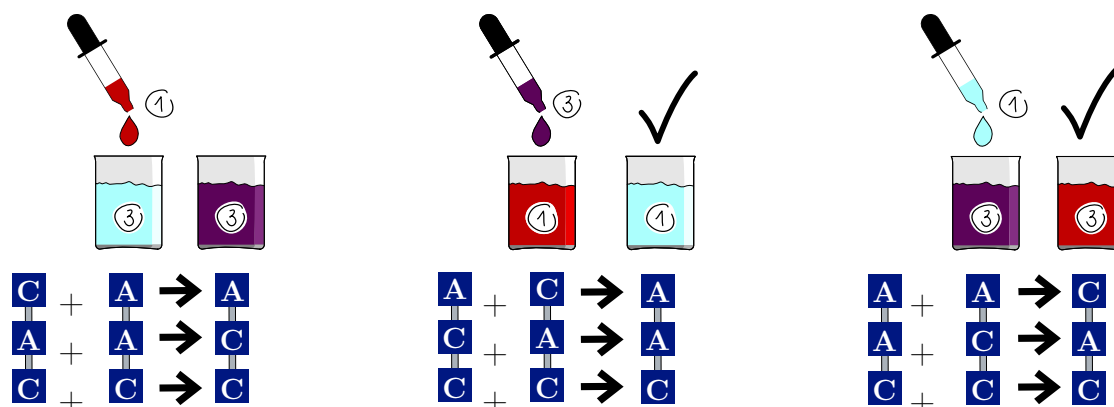
Solution

Voici la bonne réponse :

Les acides castoriques dans les récipients 1 et 3 peuvent être échangés avec trois instructions. Il a deux bonnes réponses :



Nous illustrons la première réponse, $1 \rightarrow 3, 3 \rightarrow 1, 1 \rightarrow 3$. Au départ, les acides dans les récipients sont : 1: CAC, 2: ACA et 3: AAC.



$1 \rightarrow 3$: $CAC + AAC \rightarrow ACC$ se forme dans le récipient 3. Les acides dans les récipients sont maintenant : 1: CAC, 2: ACA et 3: ACC.

$3 \rightarrow 1$: $ACC + CAC \rightarrow AAC$ se forme dans le récipient 1. Les acides dans les récipients sont maintenant : 1: AAC, 2: ACA et 3: ACC.

$1 \rightarrow 3$: $AAC + ACC \rightarrow CAC$ se forme dans le récipient 3. Les acides dans les récipients sont maintenant : 1: AAC, 2: ACA et 3: CAC. C'est l'ordre dans lequel nous voulions mettre les acides.

Nous allons maintenant démontrer qu'il n'est pas possible d'échanger les acides dans les récipients 1 et 3 avec moins de trois instructions. On voit tout de suite qu'une seule instruction ne peut pas suffire : on doit modifier deux acides, et chaque instruction n'en modifie qu'un.

Pour que deux instructions suffisent, il faudrait qu'une instruction forme l'acide AAC dans le récipient 1 et l'autre l'acide CAC dans le récipient 3.

Commençons par essayer de former AAC dans le récipient 1 en premier. Les deux instructions possibles ont les résultats suivants :

- $2 \rightarrow 1$: $ACA + CAC \rightarrow AAA$
- $3 \rightarrow 1$: $AAC + CAC \rightarrow ACC$

Ce n'est donc pas possible d'obtenir l'acide AAC dans le récipient 1 avec une seule instruction. Comme la deuxième instruction ne peut pas modifier deux acides, deux instructions ne suffisent pas dans ce cas.



Essayons maintenant de former CAC dans le récipient 3 en premier. Les deux instructions possibles ont les résultats suivants:

- $1 \rightarrow 3: CAC + AAC \rightarrow ACC$
- $2 \rightarrow 3: ACA + AAC \rightarrow CAA$

Ce n'est donc pas possible d'obtenir l'acide CAC dans le récipient 3 avec une seule instruction. Deux instructions ne suffisent pas dans ce cas non plus. Il faut donc au moins trois instructions.

C'est de l'informatique !

Les règles d'après lesquelles les éléments A et C forment l'élément de la nouvelle molécule correspondent à l'opération logique *OU exclusif* (*XOR*). On peut le constater en remplaçant A et C par les valeurs «vrai» et «faux», respectivement: XOR retourne la valeur «vrai» seulement lorsque ses deux entrées sont différentes.

XOR joue un rôle important en informatique: les propriétés permutatives de cette opération permettent de l'utiliser pour échanger les valeurs de deux variables sans devoir utiliser de troisième variable temporaire – comme avec les récipients de cet exercice.

XOR joue aussi un rôle important en *cryptographie*. Imagine que tu as un message représenté en code binaire. Tu peux le chiffrer en le couplant à une clé de la même longueur avec l'opérateur XOR. Par exemple, 01011 (message) XOR 11001 (clé) = 10010 (message chiffré). On peut retrouver le message original en couplant le message chiffré à la clé avec XOR: 10010 (message chiffré) XOR 11001 (clé) = 01011 (message). Sans clé, le message reste par contre complètement caché, comme chaque bit du message chiffré peut venir soit d'un 1, soit d'un 0 dans le message original. Le chiffrement de flux, utilisé en téléphonie mobile, utilise l'opérateur XOR.

Mots clés et sites web


- XOR: https://fr.wikipedia.org/wiki/Fonction_OU_exclusif
- Algorithme de permutation avec XOR: [https://fr.wikipedia.org/wiki/Permutation_\(informatique\)#En_utilisant_1.27op.C3.A9ration_XOR](https://fr.wikipedia.org/wiki/Permutation_(informatique)#En_utilisant_1.27op.C3.A9ration_XOR)
- Chiffrement de flux: https://fr.wikipedia.org/wiki/Chiffrement_de_flux







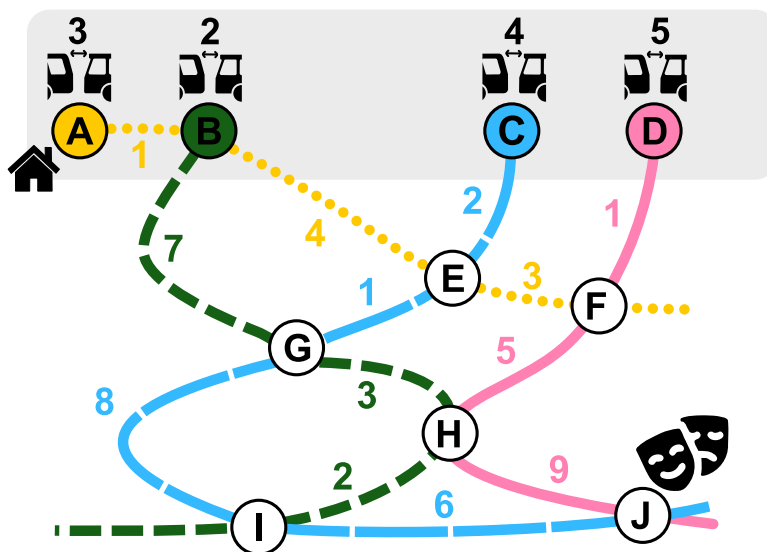
14. Transports publics

Il y a 4 lignes de bus dans la ville de Marcus. Elles partent des arrêts A, B, C et D, respectivement. Tu peux voir leur itinéraire sur le plan ci-dessous.

Les premiers bus de chaque ligne partent tous au même moment des arrêts de départ (A, B, C, D) : c'est la minute 0. Ensuite, les bus partent aux intervalles indiqués . Par exemple, un bus part toutes les 3 minutes de l'arrêt A, donc aux minutes 0, 3, 6, 9, etc.

Chaque tronçon d'itinéraire entre deux arrêts indique le nombre de minutes nécessaire à parcourir le tronçon. Par exemple, le premier bus partant de l'arrêt A atteint l'arrêt F à la minute $0 + 1 + 4 + 3 = 8$.

Marcus habite la maison  près de l'arrêt A. Depuis là, il aimerait prendre le premier bus pour aller jusqu'au théâtre . Il peut changer de bus en 0 minute aux arrêts auxquels deux lignes se croisent. Il peut donc continuer sa route avec n'importe quel bus qui arrive en même temps ou plus tard que lui au même arrêt. Marcus sait quelles lignes prendre et où changer pour arriver au théâtre le plus tôt possible.



Quels arrêts se trouvent sur la route de Marcus ?



Solution

Voici la bonne réponse :

La route de Marcus passe par les arrêts A, B, E, G, H et J. Il peut arriver au théâtre au plus tôt à la minute 20.

Comment trouver ce temps minimum et la route qui va avec ? Le temps le plus court dont Marcus a besoin pour arriver à l'arrêt J (et donc au théâtre) est le minimum entre :

- le temps le plus court pour atteindre l'arrêt H, plus le temps d'attente du bus venant de D, plus 9 minutes, et
- le temps le plus court pour atteindre l'arrêt I, plus le temps d'attente du bus venant de C, plus 6 minutes.

On pourrait maintenant déterminer ces deux temps minimums en revenant en arrière vers les stations précédentes. Alternativement, on peut calculer le temps minimum pour toutes les stations atteignables depuis la station A de Marcus en se souvenant du temps minimum pour l'arrêt précédent. On connaît à la fin ainsi le temps minimum pour atteindre le théâtre et les arrêts se trouvant sur ce chemin optimal.

- Arrêt **A** : Marcus atteint A à la minute **0**.
- Arrêt **B** : Marcus atteint B à la **minute 1, depuis A**.
- Les arrêts C und D ne se trouvent sur aucune route que Marcus pourrait raisonnablement prendre.
- Arrêt **E** : Marcus atteint E seulement **depuis B**, sans attente à B, à la **minute 5**.
- Arrêt **F** : Marcus atteint F (sans détour) seulement **depuis E**, sans attente à E, à la **minute 8**.
- Arrêt **G** : Marcus peut atteindre G par deux chemins, en venant de B ou de E. (1) Marcus est à B à la minute 1, peut en partir à la minute 2 et arrive à G à la minute 9. (2) Marcus est à E à la minute 5, en part à la minute 6 (le bus venant de C part de E aux minutes 2, 6, 10, etc.) et arrive à G à la minute 7. Le temps minimum est donc **7, depuis E**.
- Arrêt **H** : Marcus peut atteindre H en venant de F ou de G. (1) Marcus est à F à la minute 8, peut en partir à la minute 11 (le bus venant de D part de F aux minutes 6, 11, 16, etc.) et arrive à H à la minute 16. (2) Marcus est à G à la minute 7, en part directement (le bus venant de B part de G aux minutes 7, 9, 11, etc.) et arrive à H à la minute 10. Le temps minimum est donc **10, depuis G**.
- Arrêt **I** : Marcus peut atteindre I en venant de H ou de G. (1) Marcus est à H à la minute 10, peut en partir directement (le bus venant de B part de H aux minutes 10, 12, 14, etc.) et arrive à I à la minute 12. (2) Marcus est à G à la minute 7, en part directement (le bus venant de C part de G aux minutes 3, 7, 11, etc.) et arrive à I à la minute 15. Le temps minimum est donc **12, depuis H**.
- Arrêt **J** / **Théâtre** : Marcus peut atteindre J en venant de H ou de I. (1) Marcus est à H à la minute 10, peut en partir à la minute 11 (le bus venant de D part de H aux minutes 6, 11, 16, etc.) et arrive à J à la minute 20. (2) Marcus est à I à la minute 12, en part à la minute 15 (le



bus venant de C part de I aux minutes 11, 15, 19, etc.) et arrive à J à la minute 21. Le temps minimum est donc **20, depuis H**.

Comme nous nous sommes souvenus depuis où Marcus a atteint chaque arrêt le plus rapidement, nous pouvons parcourir sa route en arrière: $J \leftarrow H \leftarrow G \leftarrow E \leftarrow B \leftarrow A$.

C'est de l'informatique !

L'une des idées pour déterminer la bonne réponse de cet exercice du castor était de séparer le problème principal (arriver le plus tôt possible à J) en deux problèmes plus petits (arriver le plus tôt possible à I et à H). Nous aurions pu utiliser cette méthode pour les autres arrêts: pour arriver le plus tôt à I, il faut choisir le chemin le plus court en venant soit de G, soit de H, et ainsi de suite. Cette stratégie consistant à utiliser une fonction pour la calculer elle-même s'appelle *récurtivité* en informatique.

Dans l'explication ci-dessus, nous n'avons pas utilisé la récursivité, mais nous avons calculé les résultats de la fonction en commençant par le cas le plus simple et en recommençant pour chaque étape (*itérativement*). Ainsi, nous n'avons pas seulement reporté le calcul pour J sur les calculs pour I et H, mais utilisé les résultats connus pour I et H pour calculer le résultat de J.

Cette stratégie est appelée *programmation dynamique* en informatique. On peut l'utiliser pour résoudre des problèmes d'optimisation comme la recherche de l'heure d'arrivée au théâtre pour Marcus dans cet exercice. La programmation dynamique fonctionne lorsque le principe d'optimalité de Bellmann est satisfait, c'est-à-dire lorsque la solution optimale d'un problème est faite des solutions optimales des sous-problèmes qui le composent. Cette stratégie fonctionne alors généralement bien, parce qu'il suffit de calculer la solution de chaque sous-problème une seule fois en procédant itérativement.

Mots clés et sites web

- Programmation dynamique: https://fr.wikipedia.org/wiki/Programmation_dynamique
- Récursivité: <https://fr.wikipedia.org/wiki/Récursivité>
- Itération: <https://fr.wikipedia.org/wiki/Itération>





15. Lacs de montagne

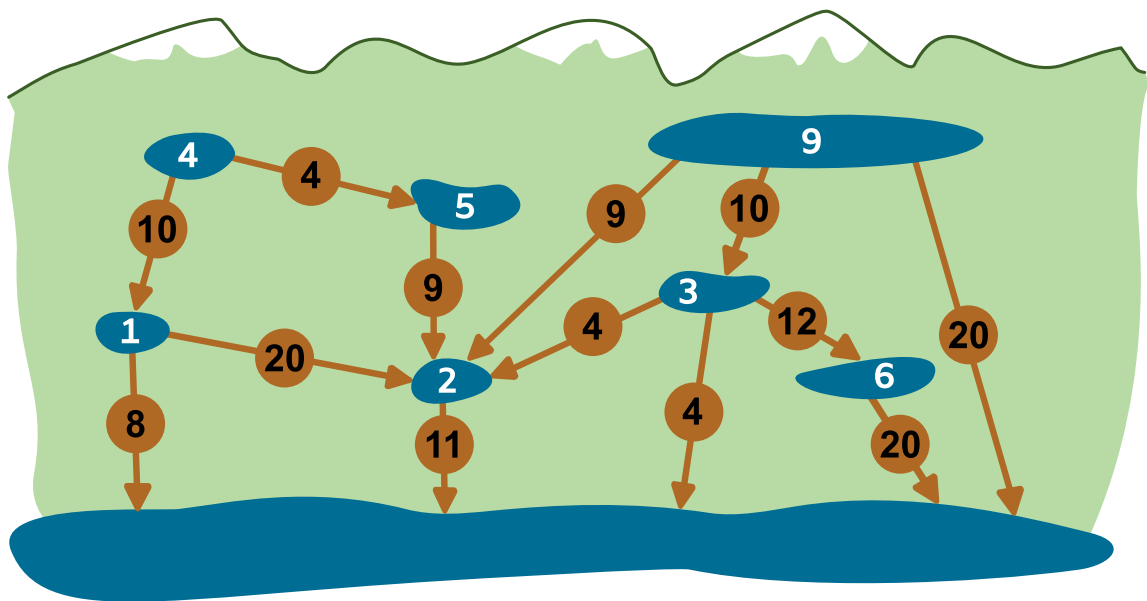
Dans la montagne au-dessus du réservoir du barrage, il y a plusieurs petits lacs naturels. Ils pourraient déborder s'il pleut beaucoup, ce qui est dangereux. C'est pour cela que des canaux doivent être construits entre certains lacs. Ces canaux doivent amener toute l'eau des lacs en trop dans le réservoir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

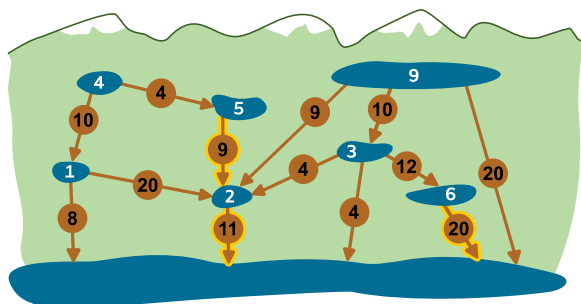
Où faut-il construire des canaux ?



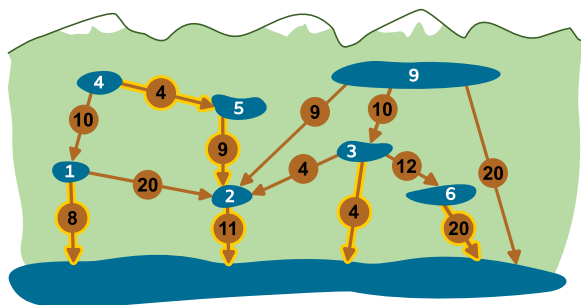


Solution

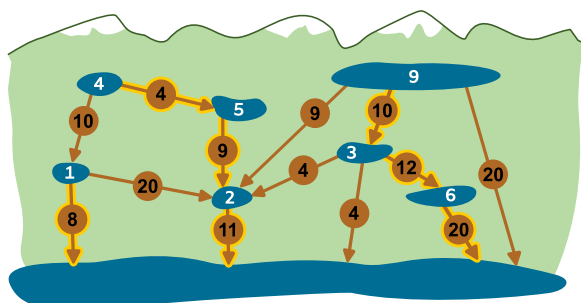
Voici la bonne réponse :



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





16. 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, Brigitte, Clara, David, Elia, Frank, Gabi, Hugo et enfin Julia.

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, David, Brigitte, Elia, Julia, Clara, Hugo, Anja et enfin Frank.

Les voitures d'Anja, Brigitte et Clara 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.



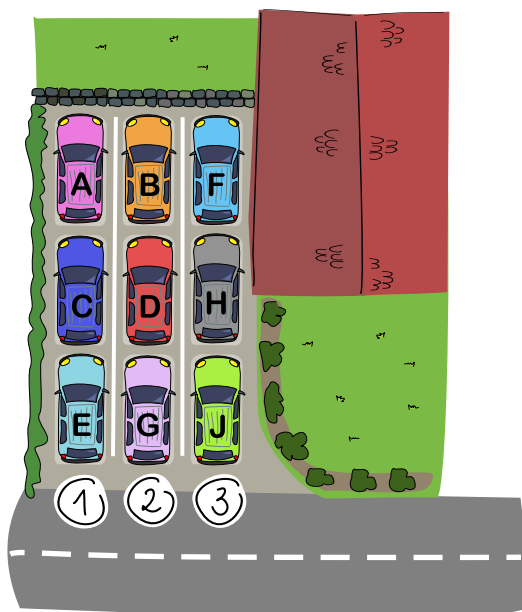
Montre aux invités comment ils peuvent se parquer !

*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 :

- Frank part en dernier et doit donc se parquer tout en avant.
- Gabi part en premier et doit donc être sur une des trois places de derrière.
- Brigitte part en troisième, donc David et Gabi doivent se parquer derrière elle.

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

- Le point a) signifie que Frank doit se parquer à côté de Brigitte, sur le devant de la colonne 3.
- David arrive après Anna, Brigitte et Clara. Le point c) signifie que David doit se parquer derrière Brigitte dans la colonne 2.
- Le point c) signifie que Gabi doit se parquer derrière David dans la colonne 2.
- Il ne reste que la place derrière Clara dans la colonne 1 pour Emil. Il part en quatrième (après Gabi, David et Brigitte) et doit donc être parqué tout à l'arrière d'une colonne. Quand il arrive, il n'y a que Frank dans la colonne 3 et Emil 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 Hugo (au milieu) et Julia (derrière). Heureusement que Julia veut partir avant Hugo, 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 parkées 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_\(informatique\)](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



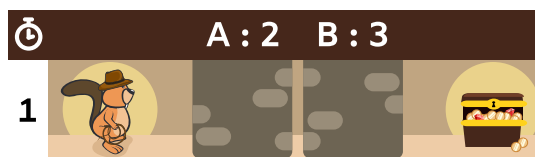


17. Castor Jones

Castor Jones  est dans une terrible pyramide remplie de couloirs dangereux. Au bout de chaque couloir se trouve un fabuleux trésor. Jones veut atteindre chaque trésor *le plus vite possible*.

Chaque couloir est bloqué par une série de blocs de pierre. Au départ, tous les blocs sont en bas. Dès que quelqu'un entre dans le couloir, ils commencent à se déplacer. Chaque bloc se déplace de haut en bas à un rythme régulier. Par exemple, un bloc avec le tempo 2 va monter d'un coup au bout de deux minutes, puis redescendre tout aussi vite deux minutes plus tard.

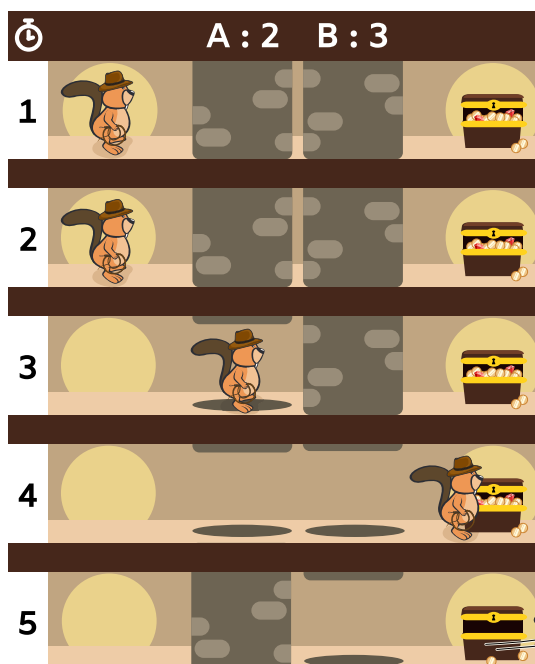
Jones se retrouve devant sa première épreuve :



Ce couloir a deux blocs : le bloc A a le tempo 2 et le bloc B le tempo 3. Heureusement, Jones a trouvé une note avec des instructions permettant d'atteindre le trésor le plus vite possible :

```
wait(2)
goto_block(A)
wait(1)
goto_treasure
```

Jones suit les instructions : il attend 2 minutes, puis va vers le bloc A, y attend 1 minute et va ensuite vers le trésor. Il atteint le trésor après 3 minutes :





Jones se rend compte qu'il aurait pu atteindre le trésor à la même vitesse avec moins d'instructions :

```
wait(3)
goto_treasure
```

Il arrive au couloir suivant. Celui-ci a quatre blocs qui ont les tempos 3, 5, 8 et 4.

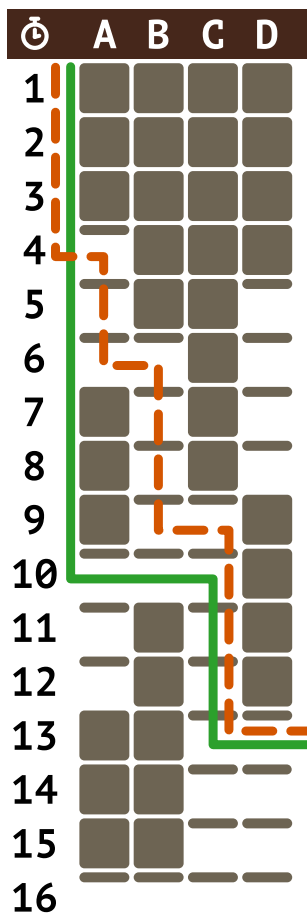
Quelle est la plus courte séquence d'instructions permettant à Jones d'atteindre le trésor le plus vite possible ?





Solution

L'image ci-dessous montre comment Jones peut se déplacer dans le couloir :



Pour se déplacer aussi vite que possible, il peut suivre les instructions suivantes et atteindre le trésor après la douzième minute (comme le montre la ligne orange) :

```
wait(3)
goto_block(A)
wait(2)
goto_block(B)
wait(3)
goto_block(C)
wait(4)
goto_treasure
```

Il y a cependant une séquence d'instructions plus courte lui permettant aussi d'atteindre le trésor après la douzième minute (comme le montre la ligne verte) :

```
wait(9)
goto_block(C)
wait(3)
goto_treasure
```



Ce n'est pas possible d'atteindre le trésor avec moins d'instructions sans perdre de temps. La seule possibilité serait que Jones traverse le couloir d'un seul coup, et il devrait attendre que tous les blocs soient en haut en même temps à la fin de la quinzième minute pour cela.

C'est de l'informatique !

Castor Jones, l'intrépide aventurier, veut bien sûr atteindre le fabuleux trésor le plus vite possible – pas que quelqu'un d'autre ne le précède ! Mais, en tant qu'optimiseur très bien organisé, Jones ne se contente pas de n'importe quelles instructions pour avancer vite. Non, il cherche la séquence d'instructions la plus courte lui permettant d'y arriver. Il optimise donc son chemin avec deux objectifs, ou ajoute une condition supplémentaire à l'optimisation de la vitesse. Facile pour Jones !

Ce serait plus compliqué si Jones avait choisi deux objectifs d'optimisation pas forcément compatibles : par exemple d'atteindre le trésor le plus vite possible tout en gardant ses habits en forme pour tourner dans un film hollywoodien. En informatique et mathématiques, on parle dans ce genre de cas d'*optimisation multiobjectif*. Ces problèmes n'ont souvent pas d'optimum unique, mais un *optimum de Pareto*. L'optimum de Pareto (nommée d'après Vilfredo Pareto, un économiste italien) est un état dans lequel il n'est pas possible d'améliorer un objectif sans réduire l'autre.

Comme pour toutes les bonnes informaticiennes et les bons informaticiens, c'est la qualité qui compte pour Jones. En informatique, le concept de qualité ou d'optimum ne concerne pas que les résultats de calculs, mais par exemple aussi la qualité du code d'un programme. Celle-ci peut être mesurée grâce à différents critères, comme la structure du code ou les commentaires qui y sont associés. S'il y a le choix entre plusieurs codes ayant la même fonction, celui ayant le moins d'instructions peut être sélectionné, comme dans cet exercice du castor.

Mots clés et sites web

- Optimisation: [https://fr.wikipedia.org/wiki/Optimisation_\(mathématiques\)](https://fr.wikipedia.org/wiki/Optimisation_(mathématiques))
- Optimisation multiobjectif:
https://fr.wikipedia.org/wiki/Optimisation_multiobjectif
- Optimum de Pareto: https://fr.wikipedia.org/wiki/Optimum_de_Pareto



18. Session d'examens

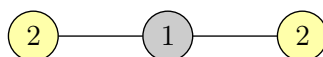
C'est la saison des examens de maturité au gymnase Euler. La session d'examens doit durer au maximum 5 jours. Comme on ne peut passer qu'un examen par jour, 2 examens qu'un même élève doit passer ne peuvent pas avoir lieu le même jour. Ces 2 examens sont alors en « conflit ».

Pour faciliter la planification, les conflits sont représentés dans un diagramme fait de cercles et de lignes :

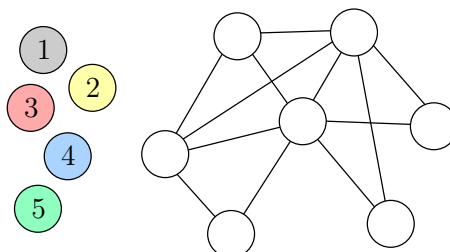
- Un cercle vide est dessiné pour chaque examen ;
- Une ligne est dessinée entre deux examens si (et seulement si) ces 2 examens sont en conflit et ne peuvent pas être prévus pour le même jour.

On écrit ensuite dans les cercles le numéro du jour (1 à 5) auquel l'examen se passera.

Voici un exemple avec 3 examens : l'examen du milieu se passe le premier jour et a 2 conflits, un avec chacun des deux autres examens, qui, eux, se passent le deuxième jour et n'ont pas de conflit entre eux.



Dans les 5 prochains jours, 7 examens doivent être prévus. Le diagramme suivant montre leurs conflits :



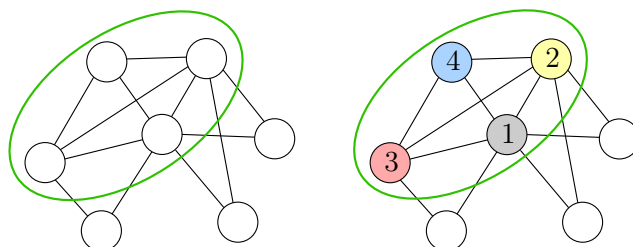
Répartis les examens sur le moins de jours possible (1 à 5) en tenant compte des conflits. Il y a plusieurs bonnes réponses.



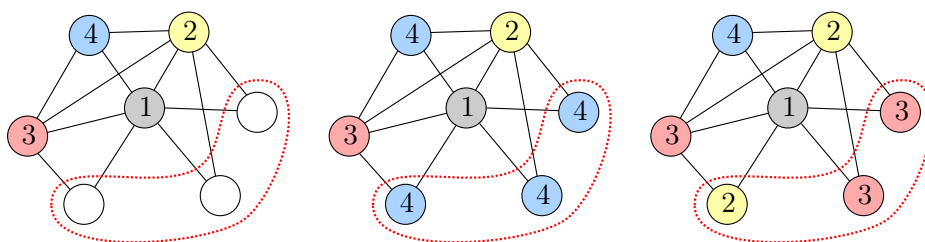
Solution

Voici la bonne réponse :

En observant le diagramme, on remarque un motif spécial entre les examens dessinés en haut à gauche (image de gauche). Chacun de ces examens est en conflit avec chacun des autres, ce qui fait qu'ils doivent être prévus pour 4 jours différents. Tu vois une possibilité de faire ça sur l'image de droite. Chacune des autres $1 \times 2 \times 3 \times 4 = 24$ possibilités de répartir les examens sur 4 jours est également juste.



Nous observons maintenant les 3 examens restants (image de gauche) pour cette répartition. Aucun d'entre eux ne peut être prévu pour le jour 1, mais tous pour le jour 4 (image du milieu). Si l'on veut faire les examens pendant les jours 2 ou 3, on ne peut le faire que comme sur l'image de droite à cause des conflits. Il reste encore 6 possibilités de déplacer un ou 2 des examens prévus pour les jours 2 et 3 au jour 4.



Il y a donc 8 possibilités de répartir les 3 examens en bas à droite pour chacune des 24 possibilités de répartir les examens en haut à gauche. Il y a donc 192 bonnes réponses avec les jours $\{1, 2, 3, 4\}$, et encore 192 solutions de plus pour chacune des quatre combinaisons de jours $\{2, 3, 4, 5\}$, $\{1, 3, 4, 5\}$, $\{1, 2, 4, 5\}$ et $\{1, 2, 3, 5\}$. Cet exercice a donc $5 \times 192 = 960$ solutions !

C'est de l'informatique !

Pour la répartition des examens, il est très utile que les conflits soient représentés dans un diagramme clair. C'est aussi important d'utiliser une représentation ou un modèle clair des données pour des problèmes de répartition plus complexes qui sont résolus à l'aide d'outils informatiques. Le diagramme des conflits de cet exercice du castor est une représentation visuelle d'un *graphe*, une structure spécialement importante en informatique pour modéliser les relations entre des objets. Un graphe est fait de *nœuds* (représentés ici par des cercles) et d'*arêtes* (les lignes entre les cercles) reliant deux nœuds chacune.



Le problème de répartition consistant à assigner aussi peu de valeurs possibles aux nœuds d'un graphe de façon à ce que deux nœuds reliés par une arête aient toujours des valeurs différentes est connu sous le nom de *coloration de graphe* en informatique – les valeurs peuvent être représentées par des couleurs. Les problèmes de coloration de graphe ont beaucoup d'applications, par exemple :

- le coloriage de cartes géographiques dans lesquelles les pays voisins doivent avoir des couleurs différentes
- La planification d'examens, comme dans cet exercice
- La répartition des fréquences dans des réseaux mobiles lorsque les interférences entre mâts doivent être évitées
- La répartition de trains sur les voies d'une gare afin que les trains présents au même moment soient sur des voies différentes.

De manière générale, les problèmes de coloration font partie des problèmes les plus difficiles en informatique – appelés problèmes *NP-difficiles*. Heureusement, suivant les applications, les problèmes de coloration peuvent être beaucoup plus faciles, et, dans les cas vraiment difficiles, des méthodes ne trouvant pas forcément la solution optimale, mais de bonnes solutions peuvent être utilisées.

Mots clés et sites web

- Coloration de graphe: https://fr.wikipedia.org/wiki/Coloration_de_graphe
- NP-difficile: <https://fr.wikipedia.org/wiki/NP-difficile>





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.





19. Autour des rochers

Benno le castor veut ramasser des bûches dans le Seeland. Écris un programme pour que le castor puisse ramasser la bûche dans tous les lacs. Clique sur les cercles sous le lac pour passer d'un lac à l'autre.

Tu peux utiliser ces instructions:

Instruction	Effet
<code>move()</code>	Benno avance d'une case dans la direction où il regarde
<code>turnRight()</code> / <code>turnLeft()</code>	Benno se tourne sur place de 90 degrés sur sa droite ou sur sa gauche
<code>removeLog()</code>	Benno ramasse la bûche de la case où il se trouve

```
while ...:
    instruction
    instruction
```

Benno répète les instructions décalées tant qu'une condition est remplie.

Dans l'exemple suivant, Benno avance d'une case tant qu'il y a un rocher à sa droite. Quand ce n'est plus le cas, il continue avec l'instruction suivante qui n'est pas décalée — ici, dans l'exemple, `turnRight()`:

```
while rockRight():
    move()
turnRight()
```



Lac 1



Lac 2



Lac 3

Écris les instructions pour ramasser la bûche dans tous les lacs. Les lignes 1 à 3 sont déjà données et ne peuvent pas être modifiées.





Solution

La solution correcte est la suivante:

```
while rockRight():
    move()
    turnRight()

move()
move()
move()
turnRight()
move()

while rockRight():
    move()
    move()

removeLog()
```

Une première exécution du programme avec les lignes 1 à 3 déjà données montre que Beno avance tant qu'il y a un rocher à sa droite, puis effectue une rotation vers la droite. Le reste du code peut maintenant être divisé en deux parties.

Nous devons d'abord guider Beno autour de la rangée horizontale supérieure des rochers. Comme cette rangée a exactement deux rochers de large dans tous les lacs, nous pouvons parcourir cette distance directement en exécutant trois fois la commande `move()`.

Pour la descente, en revanche, il ne suffit pas de compter le nombre de pas en avant, car la distance jusqu'à la bûche est différente dans les trois lacs. Nous avons donc de nouveau besoin de la structure de contrôle `while` pour exécuter une ou plusieurs instructions de manière répétée.

Contrairement à la rangée de rochers de gauche, il y a toujours exactement une case vide entre les rochers de la rangée de droite. Cela signifie que, pour que la condition `rockRight()` soit à nouveau remplie, Beno doit avancer de deux cases.

C'est de l'informatique !

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

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



un cas précis sont appelées *hardcode*. Nous essayons souvent d'éviter qu'une solution soit *hardcodée* et écrivons plutôt des programmes utilisables pour plusieurs problèmes similaires.

Grâce à la structure de contrôle `while` combinée à la condition `rockRight()`, il est possible de déclencher un mouvement vers l'avant qui se poursuit indépendamment du nombre de rochers, et qui ne s'arrête que lorsque la condition n'est plus remplie. Dans ce cas précis, le mouvement se termine lorsqu'il n'y a plus de rocher à la droite de Beno. On parle alors d'une boucle conditionnelle.

Mots clés et sites web

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



A. Auteur·e·s des exercices

 James Atlas	 Anna Morpurgo
 Masiar Babazadeh	 A-Yeong Park
 Wilfried Baumann	 Elsa Pellet
 Susanne Berchtold	 Jean-Philippe Pellet
 Maria Cepeda	 Emmanuel Plan
 Špela Cerar	 Zsuzsa Pluhár
 Gi Soong Chee	 Wolfgang Pohl
 Christian Datzko	 Cesar F. Bolanos Revelo
 Nora A. Escherle	 Pedro Ribeiro
 Gerald Futschek	 Rokas Rimkus
 Silvan Horvath	 Kirsten Schlüter
 Alisher Ikramov	 Margareta Schlüter
 Asterios Karagiannis	 Dirk Schmerenbeck
 Blaž Kelvišar	 Jacqueline Staub
 David Khachatryan	 Supawan Tasanaprasert
 Vaidotas Kinčius	  Susanne Thut
 Stefan Koch	 Christine Vender
 Jia-Ling Koh	 Florentina Voboril
 Sophie Koh	 Michael Weigend
 Víctor Koleszar	 Philip Whittington
 Lukas Lehner	 Kyra Willekes
 Hamed Mohebbi	 Hsu Sint Sint Yee
 Mattia Monga	



B. Partenaires académiques



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



AUSBILDUNGS- UND BERATUNGSZENTRUM
FÜR INFORMATIKUNTERRICHT

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

Scuola universitaria professionale
della Svizzera italiana



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

PÄDAGOGISCHE
HOCHSCHULE
ZÜRICH



Pädagogische Hochschule Zürich
<https://www.phzh.ch/>



Universität Trier
<https://www.uni-trier.de/>



C. Sponsoring

HASLERSTIFTUNG

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



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



Kanton Bern
Canton de Berne

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>



Kanton Zürich
Volkswirtschaftsdirektion
Amt für Wirtschaft

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

Informatik Stiftung Schweiz
Fondation d'Informatique Suisse
Fondazione Informatica Svizzera
Swiss Informatics Foundation



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



cyön
<https://www.cyon.ch>



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



UBS

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

