INFORMATIK-BIBER SCHWEIZ
CASTOR INFORMATIQUE SUISSE
CASTORO INFORMATICO SVIZZERA

Exercices et solutions 2020

Tous les âges

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

Éditeurs :

Jean-Philippe Pellet, Elsa Pellet, Gabriel Parriaux,
Susanne Datzko, Fabian Frei, Juraj Hromkovič, Regula Lacher

0101001101010110010010010000001001011010101001100101001101001010010100010100101010100110101001100100101001001010010000001

SSIE

www.svia-ssie-ssii.ch
schweizerischerverein für informatikinderausbildung//société suisse pour l'informatique dans l'enseignement//società svizzera per l'informatica nell'insegnamento

Ont collaboré au Castor Informatique 2020

Susanne Datzko, Fabian Frei, Martin Guggisberg, Lucio Negrini, Gabriel Parriaux, Jean-Philippe Pellet

Cheffe de projet : Nora A. Escherle

Nous adressons nos remerciements pour le travail de développement des exercices du concours à :
Juraj Hromkovič, Michael Barot, Christian Datzko, Jens Gallenbacher, Dennis Komm, Regula Lacher, Peter Rossmanith : ETH Zurich, Ausbildunges- und Beratungszentrum für Informatikunterricht

Le choix des exercices a été fait en collaboration avec les organisateur de Bebras en Allemagne, Autriche, Hongrie, Slovaquie et Lithuanie. Nosu remercions en particulier :
Valentina Dagienė : Bebras.org
Wolfgang Pohl, Hannes Endreß, Ulrich Kiesmüller, Kirsten Schlüter, Michael Weigend : Bundesweite Informatikwettbewerbe (BWINF), Allemagne
Wilfried Baumann, Anoki Eischer : Österreichische Computer Gesellschaft
Gerald Futschek, Florentina Voboril : Technische Universität Wien
Zsuzsa Pluhár : ELTE Informatikai Kar, Hongrie
Michal Winzcer : Université Comenius de Bratislava, Slovaquie

La version en ligne du concours a été réalisée sur l'infrastructure cuttle.org. Nous remercions pour la bonne collaboration :
Eljakim Schrijvers, Justina Dauksaite, Arne Heijenga, Dave Oostendorp, Andrea Schrijvers, Alieke Stijf, Kyra Willekes : cuttle.org, Pays-Bas
Chris Roffey : Université d'Oxford, Royaume-Uni

Pour le support pendant les semaines du concours, nous remercions en plus :
Hanspeter Erni : Direction, école secondaire de Rickenbach
Gabriel Thullen : Collège des Colombières
Beat Trachsler : Kantonsschule Kreuzlingen

Christoph Frei : Chragokyberneticks (Logo Castor Informatique Suisse)
Dr. Andrea Leu, Maggie Winter, Brigitte Manz-Brunner : Senarclens Leu + Partner AG

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

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

INFORMATIK-BIBER SCHWEIZ
CASTOR INFORMATIQUE SUISSE
CASTORO INFORMATICO SVIZZERA

Le Castor Informatique 2020 a été réalisé par la Société Suisse de l'Informatique dans l'Enseignement SSIE et soutenu par la Fondation Hasler.

HASLERSTIFTUNG

Cette brochure a été produite le 9 septembre 2021 avec le système de composition de documents LaTeX. Nous remercions Christian Datzko pour le développement et maintien de la structure de génération des 36 versions de cette brochure (selon les langues et les degrés). La structure actuelle a été mise en place de manière similaire à la structure précédente, qui a été développée conjointement avec Ivo Blöchliger dès 2014. Nous remercions aussi Jean-Philippe Pellet pour le développement de la série d'outils `bebras`, qui est utilisée depuis 2020 pour la conversion des documents source depuis les formats Markdown et YAML.

Tous les liens dans les tâches ci-après ont été vérifiés le 1er décembre 2020.

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 dans le cadre du programme d'encouragement « FIT in IT ».

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

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

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

Le concours Castor Informatique 2020 a été fait pour cinq tranches d'âge, basées sur les années scolaires :

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

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 la tranche d'âge « Petit Castor », et 36 pour les années HarmoS 7 et 8).

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

Les réponses de nombreux exercices étaient affichées dans un ordre établi au hasard. Certains exercices ont été traités par plusieurs tranches d'âge.

Pour de plus amples informations :

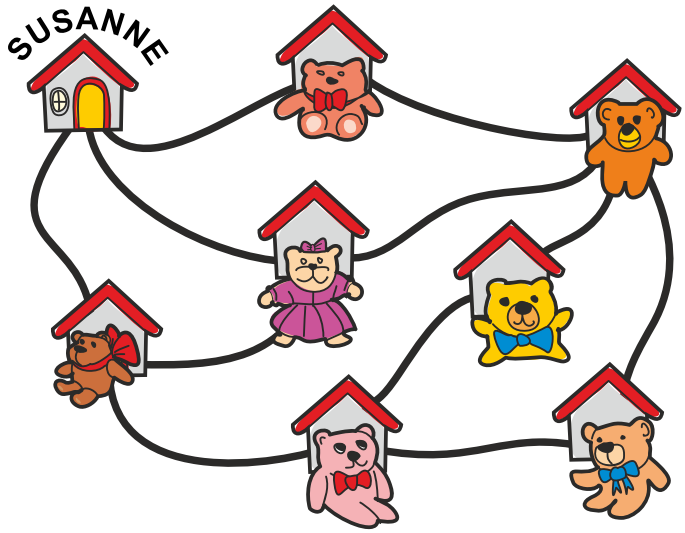
SVIA-SSIE-SSII Société Suisse de l'Informatique dans l'Enseignement
Castor Informatique
Gabriel Parriaux

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

Table des matières

1. Chasse à l'ours

Dans le quartier de Susanne, on peut voir les nounours suivants devant les maisons :



Susanne a fait un tour depuis chez elle en passant devant exactement quatre maisons. Elle n'a pas passé deux fois devant la même maison. Elle n'a pas vu le nounours devant l'une des maisons. Les trois autres nounours étaient :

Quel est le nounours que Susanne n'a pas vu ?

A) 

B) 

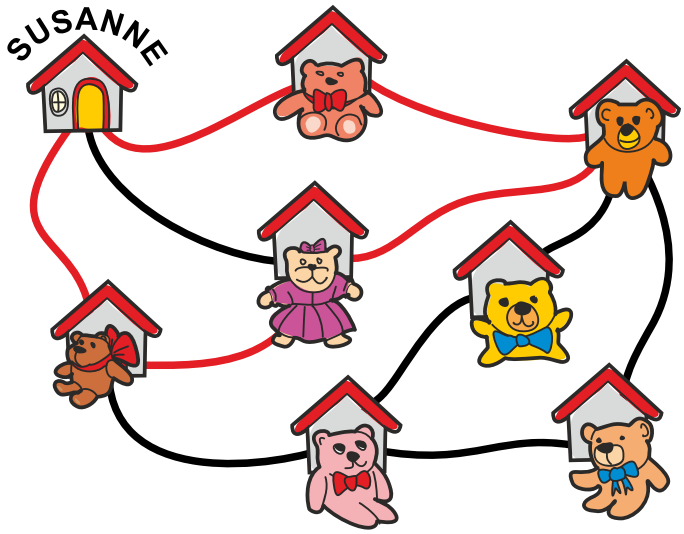
C) 

D) 

Solution

La bonne réponse est C) .

Susanne doit être passée devant les maisons avec les trois nounours , et en faisant son tour. Ces trois nounours sont directement reliés par un chemin. Le premier nounours est directement après sa maison. À la fin de ce chemin, elle se trouve près du troisième nounours . Depuis là, il n'y a qu'un chemin qui va chez elle en ne passant devant qu'une seule quatrième maison, et c'est le chemin qui passe devant le nounours . D'autres chemins possibles passent devant au moins deux autres nounours, et elle n'a passé que devant quatre maisons. La carte suivante montre le chemin :



Susanne peut faire son tour dans les deux directions, cela ne change rien.

C'est de l'informatique !

Des textes à trous en cours de français, des exercices de mathématiques avec des champs vides ou une chasse à l'ours à laquelle il manque une image : ce sont tous des exercices dans lesquels on recherche une information manquante. Les exercices sont construits (ou *structurés*) de façon à ce que l'information manquante puisse être trouvée par *raisonnement logique* ou *déduction*.

Ce genre de choses arrivent fréquemment en informatique. Il peut y avoir des erreurs lors de la transmission ou de la sauvegarde de données. C'est pour cela que l'on travaille avec des méthodes qui *détectent* ou même *corrigent les erreurs*. Si l'erreur n'est pas trop importante, on peut le faire en enregistrant volontairement plus d'informations que nécessaire. Dans cet exercice, il s'agit de la carte et du fait que Susanne passe devant exactement quatre nounours. Ainsi, elle peut trouver l'information manquante, c'est-à-dire quel nounours elle n'a pas vu.

En 2020, de telles chasses aux ours lors desquelles des peluches étaient cachées aux fenêtres de différentes maisons ont été organisées dans plusieurs pays du monde. Cela permettait aux enfants de jouer ensemble à cacher et découvrir malgré les règles de distanciation durant la pandémie de coronavirus. L'idée de jouer à faire une chasse à l'ours vient à l'origine du livre d'image « *We're Going*

on a Bear Hunt » de Michael Rosen (1989). En français, on connaît le livre et le jeu correspondant sous le nom de « La chasse à l'ours ».

Mots clés et sites web

- Détection et correction d'erreurs : `https://fr.wikipedia.org/wiki/Code_correcteur`
- Déduction logique : `https://fr.wikipedia.org/wiki/Déduction_logique`
- Chasse à l'ours : `https://www.insider.com/coronavirus-pandemic-sparked-worldwide-bear-hunt-to-entertain-kids-2020-4`, `https://www.actualitte.com/article/zone-51/pendant-toute-la-duree-du-confinement-la-chasse-a-l-ours-est-ouverte/100109`, `https://www.youtube.com/watch?v=0gyI6ykDwds`

2. La pièce de théâtre

Une pièce de théâtre raconte l'histoire d'une belle princesse, d'un noble chevalier, d'un roi sage et d'un méchant dragon. Au début de la pièce, la scène est vide. Pendant la représentation, les quatre personnages entrent en scène et quittent la scène dans l'ordre suivant :

Premier acte		**Deuxième acte**
Le roi entre en scène		Le dragon entre en scène
La princesse entre en scène		Le chevalier entre en scène
Le roi quitte la scène	ENTRACTE	Le dragon quitte la scène
Le dragon entre en scène		La princesse entre en scène
La princesse quitte la scène		Le chevalier quitte la scène
Le dragon quitte la scène		La princesse quitte la scène
Entracte		**Fin**

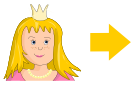
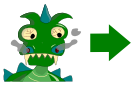
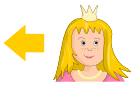
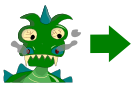
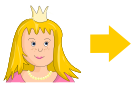
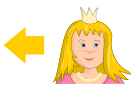
Quelle situation n'aura pas lieu ?

A) La princesse et le chevalier sont ensemble sur scène.
B) Le roi et le dragon sont ensemble sur scène.
C) Le chevalier n'entre en scène qu'après l'entracte.
D) Le chevalier et le dragon sont ensemble sur scène.

Solution

La bonne réponse est B) Le roi et le dragon sont ensemble sur scène, car cette affirmation n'est jamais vraie au cours de la pièce de théâtre.

On peut y réfléchir pas à pas :

Intrigue	 Roi sur scène ?	 Princesse sur scène ?	 Dragon sur scène ?	 Chevalier sur scène ?	Réponses correspondantes
Premier acte					
	Oui	Non	Non	Non	
	Oui	Oui	Non	Non	
	Non	Oui	Non	Non	
	Non	Oui	Oui	Non	
	Non	Non	Oui	Non	
	Non	Non	Non	Non	
Entracte					
Deuxième acte					
	Non	Non	Oui	Non	
	Non	Non	Oui	Oui	C), D)
	Non	Non	Non	Oui	
	Non	Oui	Non	Oui	A)
	Non	Oui	Non	Non	
	Non	Non	Non	Non	
Fin					

On peut vérifier pour chaque réponse si l'affirmation qui y est faite est vraie ou pas en parcourant la table ligne par ligne.

Pour la réponse A), on cherche une ligne à laquelle la princesse et le chevalier sont présents sur scène. C'est la cas à la deuxième ligne du deuxième acte, car la princesse entre en scène alors que le chevalier y est déjà depuis la deuxième ligne et y reste jusqu'à la cinquième ligne. L'affirmation de la réponse A) est donc vraie à au moins un moment de la pièce.

Pour la réponse D), on cherche une ligne à laquelle le chevalier et le dragon sont présents sur scène. C'est la cas à la deuxième ligne du deuxième acte, car le chevalier monte sur scène alors que le dragon y est déjà depuis la première ligne et y reste jusqu'à la troisième ligne. L'affirmation de la réponse D) est donc vraie à au moins un moment de la pièce.

L'affirmation de la réponse C) est d'un genre différent. Si cette affirmation est vraie, le chevalier ne doit pas avoir été sur scène de tout le premier acte. On doit donc regarder la colonne du chevalier pendant le premier acte. Dans celle-ci, c'est toujours écrit « non », donc le chevalier n'a en effet pas été sur scène pendant le premier acte. Par contre, il entre en scène à la deuxième ligne du deuxième acte, donc l'affirmation de la réponse C) est également vraie.

Si l'affirmation de la réponse B) était vraie, le roi et le dragon devraient être les deux sur scène à l'une des lignes. Cependant, il n'y a aucune ligne dans laquelle c'est écrit « oui » dans les deux colonnes. Le roi quitte déjà la scène à la troisième ligne du premier acte et n'y entre plus jusqu'à la fin. Le dragon, lui, entre en scène seulement à la quatrième ligne du premier acte. Peut-être qu'ils se rencontrent dans les coulisses, mais ils ne sont jamais ensemble sur scène. L'affirmation de la réponse B) n'est donc jamais vraie, et B) est la bonne réponse.

C'est de l'informatique !

On peut s'imaginer toute une histoire pendant le déroulement de la pièce de théâtre, mais seule une des propriétés de chaque personnage est importante pour cet exercice : se trouve-t-il sur scène à un moment précis ou pas ? Cette restriction de la perspective à certaines propriétés s'appelle l'*abstraction*.

De telles abstractions peuvent facilement être formulées en informatique. Pour chaque personnage, on définit ce que l'on appelle une *variable*, qui répond à la question de la présence du personnage sur scène à ce moment-là. Les quatre variables sont : « Roi sur scène ? », « Princesse sur scène ? », « Dragon sur scène ? » et « Chevalier sur scène ? ». La réponse à chacune de ces questions change plusieurs fois pendant la pièce de théâtre ; la réponse à chaque question est parfois « oui » et parfois « non ». En informatique, on appelle la réponse actuelle à une question la valeur actuelle de la variable correspondante. La valeur d'une variable peut donc changer autant de fois que nécessaire en informatique (c'est différent en mathématiques où les variables ne changent pas de valeur avec le temps). La table dans l'explication de la réponse montre les quatre variables et les valeurs correspondantes à chaque moment.

Il y a d'autres manières de considérer la pièce de théâtre. On peut regarder quels personnages sont sur scène en ce moment (on observe alors la valeur momentanée des quatre variables). On appelle chaque combinaison de personnages un état de la scène. Lorsqu'un personnage entre en scène ou la quitte, l'état de la scène change. On appelle aussi cela une transition de la scène d'un état à un

autre. Si l'on dessine un rond séparé pour chaque état (combinaison de personnages) sur une feuille de papier, on peut voir l'ensemble comme une abstraction de la scène.

De plus, on peut représenter les transitions possibles par des flèches reliant un état à un autre. En faisant cela, on obtient ce que s'appelle en informatique un *diagramme états-transitions* de la scène.

Au début de la pièce de théâtre, la scène est vide. On appelle l'état correspondant l'*état initial*. On peut dessiner le déroulement de la pièce de théâtre comme un chemin dans le diagramme états-transitions. Le chemin commence à l'état initial et suit ensuite les flèchent qui correspondent à l'intrigue de la pièce.

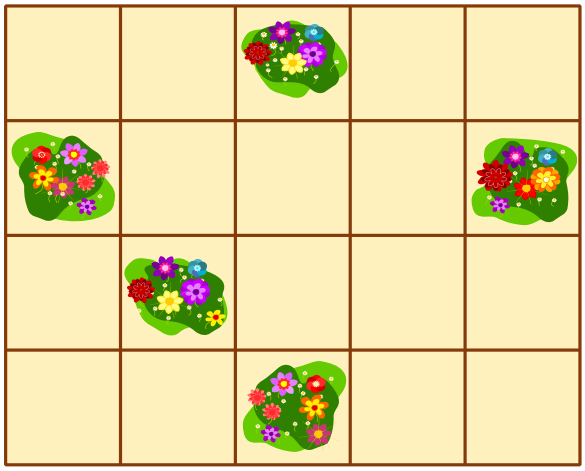
Les diagrammes états-transitions sont très importants en informatique. On doit réfléchir au diagramme états-transitions de chaque système complexe à un moment donné. Pour les êtres humains, c'est souvent laborieux de travailler avec de tels états et transitions abstraits, alors que les ordinateurs peuvent très bien le faire. Cela vaut donc la peine pour les êtres humains de représenter leurs problèmes dans des diagrammes états-transitions afin que les ordinateurs puissent les résoudre.

Mots clés et sites web

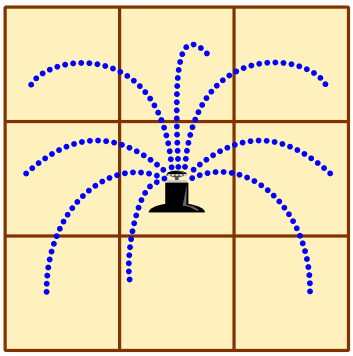
- Variable : `https://fr.wikipedia.org/wiki/Variable_(informatique)`
- État, transition, diagramme états-transitions : `https://fr.wikipedia.org/wiki/Diagramme_états-transitions`

3. Arrosage

Le jardin de Daniel est composé de cases carrées. Il a planté des fleurs dans certaines de ces cases :



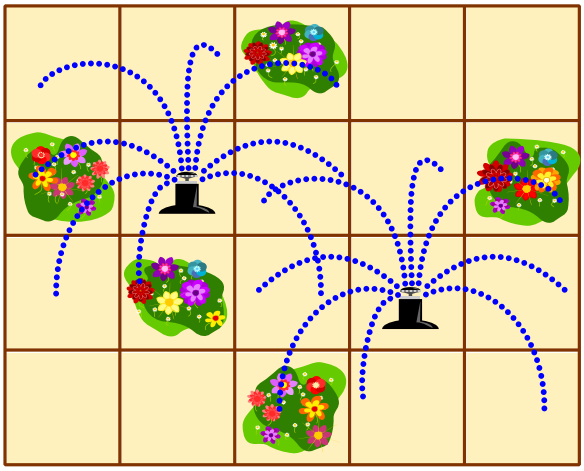
En été, il aimerait arroser ces fleurs avec des tourniquets arroseurs. Il ne peut pas mettre d'arroseur dans les cases avec des fleurs. Un arroseur arrose toutes les fleurs dans les 8 cases autour de lui :



Place aussi peu d'arroseurs que nécessaire pour arroser toutes les cases fleuries. Indique-le en cochant les cases correspondantes dans le jardin de Daniel.

Solution

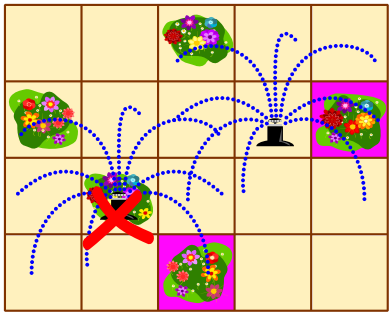
La solution suivante nécessite deux arroseurs pour arroser toutes les cases fleuries :



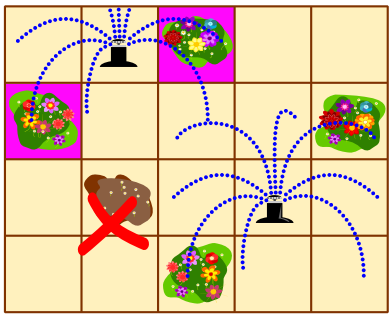
Entre la case fleurie tout à gauche et la cas fleurie tout à droite se trouvent trois cases. Un seul arroseur ne peut pas arroser deux cases si éloignées l'une de l'autre.

Il n'y a pas non plus d'autre solution que celle-ci en n'utilisant que deux arroseurs.

Pour arroser la case fleurie tout à droite et celle en bas au centre en même temps, il doit y avoir un arroseur là où il est placé dans la solution. S'il était placé plus haut pour arroser aussi la case fleurie en haut au centre, il ne pourrait plus arroser la case fleurie en bas au centre, et on ne pourrait pas arroser les trois cases fleuries restante avec un seul autre arroseur, car on ne peut pas en mettre dans une case fleurie.



Pour arroser la case fleurie tout à gauche et celle en haut au centre, un arroseur doit être mis soit comme dans la solution, soit une case plus haut. Si cet arroseur doit aussi arroser la case fleurie de la deuxième colonne depuis la gauche et troisième ligne depuis le haut, il ne peut pas être mis tout en haut.



C'est de l'informatique !

Cet exercice est un problème d'*optimisation* typique : alors qu'il est clair que toutes les cases fleuries doivent être arrosées, le nombre d'arroseurs est variable et doit être le plus petit possible. Des problèmes d'optimisation similaires existent par exemple si l'on veut placer des stations de pompiers pour protéger une ville ou couvrir des maisons avec le réseau natel.

En informatique, on parle également de *problèmes de couverture*. Ces problèmes font partie d'une classe de problèmes très difficiles en informatique. Le placement d'un nombre minimal d'arroseurs dans cet exercice était encore très simple, mais la difficulté augmente tellement fortement avec le nombre de cases qu l'on ne peut bientôt plus trouver de solution optimale en un temps raisonnable, même à l'aide d'ordinateurs.

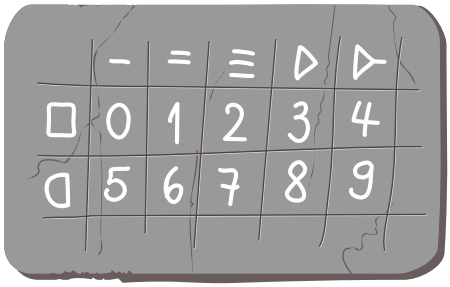
Une possibilité dans de tels cas est de se satisfaire de solutions qui ne sont peut-être pas optimales mais qui sont quand même bonnes. Ça ne fait pas grande différence si l'on place 101 au lieu 100 stations de pompiers, ou 1000 antennes natel au lieu de seulement 990, mais cela rend souvent le problème beaucoup plus facile à résoudre.

Mots clés et sites web

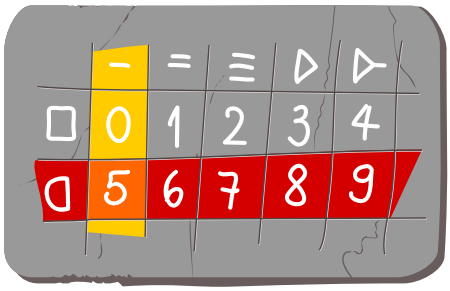
- Optimisation : `https://fr.wikipedia.org/wiki/Optimisation_(mathématiques)`
- Problème de couverture

4. Chiffres secrets

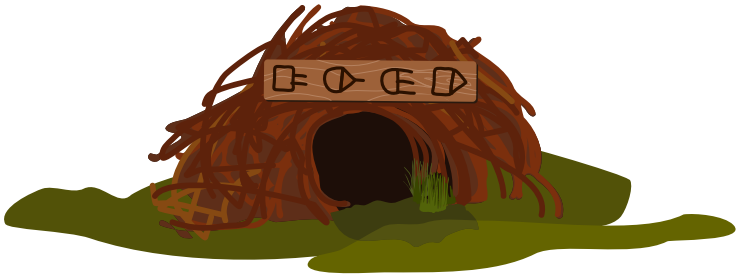
L'année de construction de chaque hutte de castor est écrite sur un panneau en dessus de l'entrée. Les castors utilisent leurs propres symboles pour représenter les chiffres. La table à droite montre comment les symboles des castors sont assemblés à partir des chiffres :



Par exemple, les castors représentent le chiffre « 5 » par le nouveau symbole  , qui est assemblé comme ça :



Voici la hutte de Cleverias :



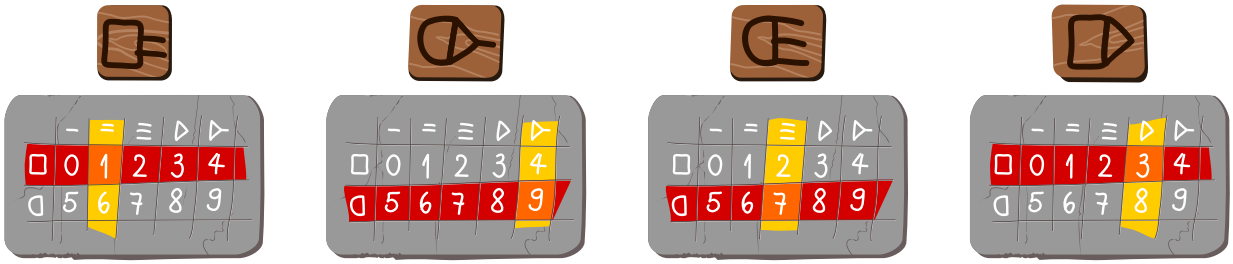
En quelle année la hutte de Cleverias a-t-elle été construite ?

A) 0978
B) 1574
C) 1923
D) 1973
E) 1993
F) 2973
G) 6378

Solution

Tu peux trouver l'année de construction de la hutte en déterminant la ligne et la colonne correspondant à chaque symbole. Le chiffre recherché se trouve à l'intersection de la ligne et de la colonne.

Comme il y a quatre symboles, tu fais cela quatre fois.



Les quatre chiffres dans le bon ordre donnent le nombre 1973.

C'est de l'informatique !

Garder des informations secrètes ou protéger des données est une tâche vielle de 4000 ans. D'innombrables écritures secrètes ont été développées et utilisées dans ce but. Aujourd'hui, la sécurité des données est l'un des thèmes majeurs de l'informatique. Une des méthodes pour empêcher la lecture non autorisée de données est de les *chiffrer*. Le chiffrement transforme un *texte clair* en *cryptogramme*. La reconstruction du texte clair à partir du cryptogramme s'appelle *déchiffrement*. L'étude des cryptogrammes s'appelle *cryptologie*.

Les cultures antiques utilisaient le plus souvent des écritures secrètes remplaçant des lettres par d'autres lettres ou de tout nouveaux symboles. L'écriture secrète utilisée ici a été développée spécialement pour le Castor Informatique, mais se base sur un concept venant de la Palestine antique. À l'époque, la règle de sécurité était que seules des écriture secrètes faciles à apprendre par cœur pouvaient être utilisées. C'était considéré comme un trop grand risque de garder une description écrite de l'écriture secrète. Une table comme celle utilisée ici est facile à apprendre par cœur. Le célèbre chiffre des francs-maçon se base sur ce principe.

Au lieu d'assembler de nouveaux symboles seulement pour les chiffres, on peut également inventer une écriture secrète pour les textes. Pour cela, on écrit toutes les lettres dans une table, puis on invente de nouveaux symboles pour les lignes et les colonnes. Ainsi, on obtient un nouveau symbole pour chaque lettre.

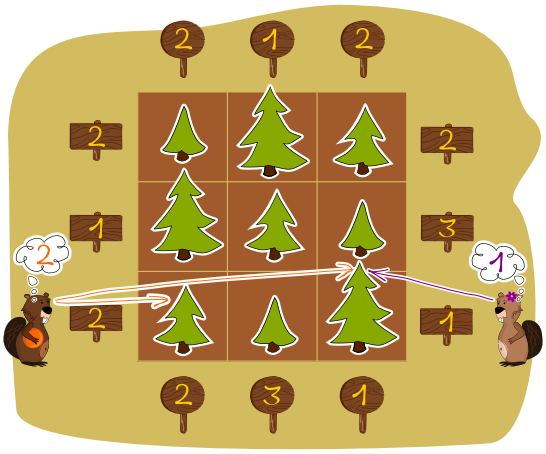
Mots clés et sites web

- Cryptographie : `https://fr.wikipedia.org/wiki/Cryptographie`
- Cryptogramme

5. Sudoku boisé 3×3

Les castors plantent des rangées de sapins. Les sapins ont trois hauteurs différentes (1, 2 et 3) et il y a exactement un sapin de chaque hauteur sur chaque rangée.

Lorsque les castors observent une rangée de sapin depuis l'une de ses extrémités, il ne peuvent **pas** voir les plus petits sapins qui sont cachés derrière de plus grands sapins.



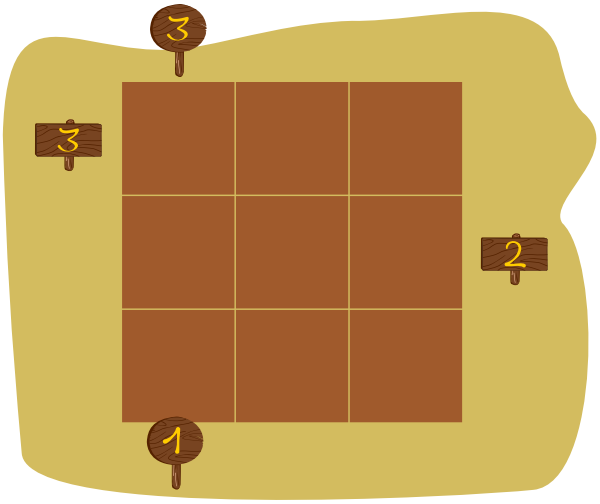
C'est écrit sur un panneau au bout de chaque rangée combien de sapins l'on peut voir depuis cet endroit-là.

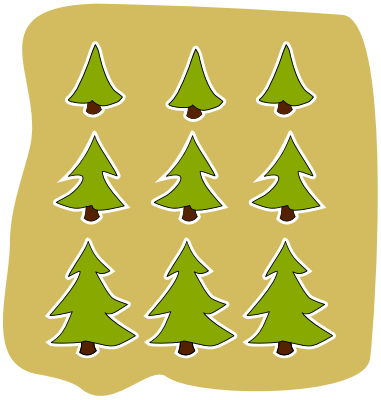
Les castors plantent à présent neuf sapins sur un champ de 3×3 cases, comme dans l'exemple à droite.

Pour cela, les règles suivantes s'appliquent :

- dans chaque ligne, il y a exactement un sapin de chaque hauteur ;
- dans chaque colonne, il y a exactement un sapin de chaque hauteur ;
- les panneaux indiquant le nombre de sapins visibles sont plantés tout autour du champ de 3×3 cases.

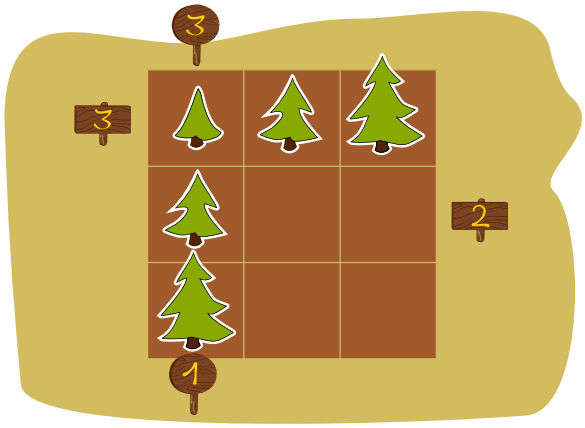
Écris dans chaque case la hauteur du sapin qui s'y trouve.





Solution

Il y a dans le champ deux panneaux indiquant que l'on peut voir trois sapins depuis leurs positions. On ne peut voir trois sapins dans une rangée que lorsque les sapins sont dans un ordre croissant, donc depuis cette position. La colonne de gauche et la ligne du haut sont ainsi déterminées :

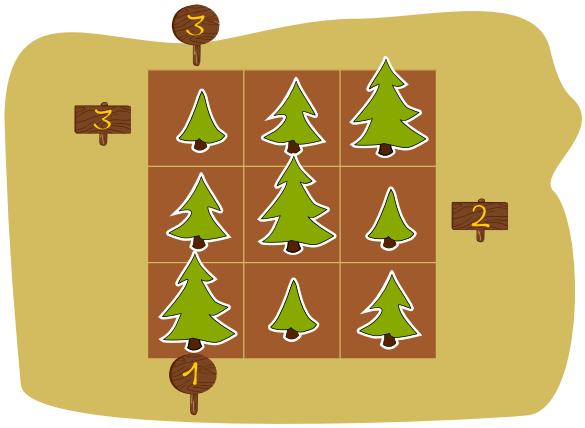


Le panneau avec le 2 à droite indique que l'on peut voir deux sapins depuis là, il doit donc y avoir un sapin de hauteur 3 au milieu et la ligne centrale est ainsi 2 (), 3 (), 1 ().

Les cases suivante sont remplies d'après la règle du « sudoku » qui oblige chaque rangée à avoir exactement un sapin de chaque hauteur.

Il doit y avoir un sapin de hauteur 1 () au milieu de la ligne du bas, car les deux autres hauteurs de sapin sont déjà présentes dans la colonne du milieu. Il manque un sapin de hauteur 2 () tout en bas à droite pour compléter la rangée.

Voici la solution complète :



C'est de l'informatique !

Cet exercice est centré sur trois compétences fondamentales pour les informaticiennes et informaticiens.

Premièrement, il s'agit de trouver une solution respectant certaines contrainte, ou si nécessaire de corriger une solution proposée.

Deuxièmement, il s'agit de la capacité de reconstruire des objets en se basant sur leur représentation à partir d'informations partielles. Ceci est lié à la génération d'objets (représentation d'objets) à partir d'informations disponibles limitées lorsque leur conformité aux lois est connue. On peut aussi utiliser de tels procédés dans la compression de données.

Troisièmement, on peut utiliser de tels champs d'arbres avec des panneaux pour créer des codes correcteurs. Des erreurs arrivant lors de l'entrée des données ou du transfert d'information peuvent ainsi être automatiquement reconnues ou même corrigées.

Mots clés et sites web

- Sudoku : `https://fr.wikipedia.org/wiki/Sudoku`
- Détection et correction d'erreurs : `https://fr.wikipedia.org/wiki/Code_correcteur`
- Reconstruction d'objets à partir d'informations partielles
- Vérification de l'exactitude de la représentation de données

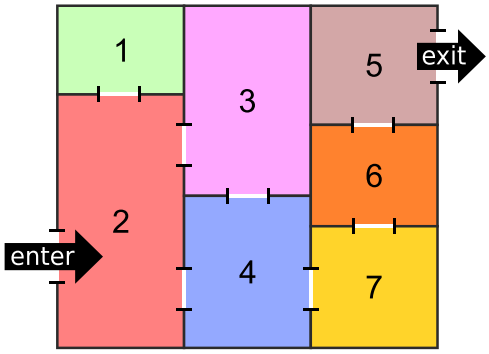
6. Visite de musée

Quatre plans au sol sont proposés pour la construction d'un nouveau musée. Chaque plan comporte les sept pièces 1 à 7. Les pièces doivent être arrangées de façon à ce que les visiteurs puissent visiter toutes les pièces sans passer deux fois par la même pièce.

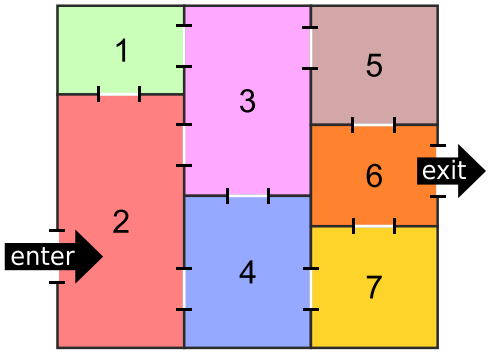
Les visiteurs commencent la visite à l'entrée « enter » et quittent le musée par la sortie « exit ».

Quel plan au sol permet aux visiteurs d'entrer et de sortir de chaque pièce exactement une fois ?

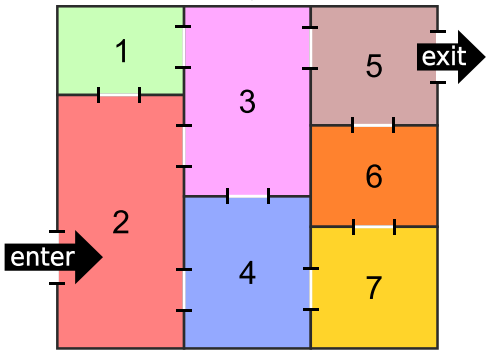
A)



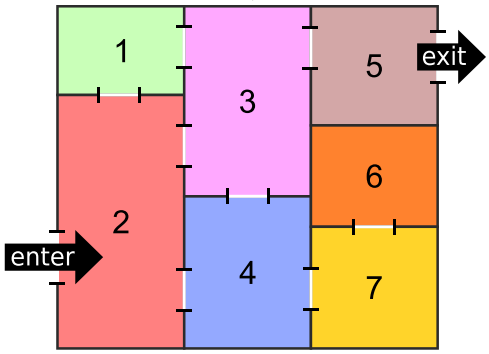
B)



C)

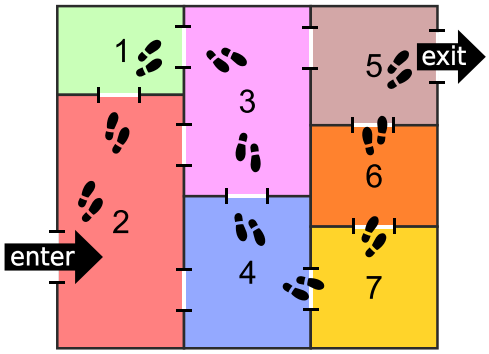


D)



Solution

Seul le plan C permet aux visiteurs de n'entrer et de ne sortir qu'une seule fois de chaque pièce. L'ordre des pièces visitées est 2, 1, 3, 4, 7, 6, 5.



De manière générale, une telle visite n'est pas possible si n'importe laquelle des pièces ne possède qu'une entrée. L'explication est la suivante : si un visiteur entre dans cette pièce, il est obligé de retourner dans la pièce de laquelle il vient en ressortant, et ne respecte donc pas la règle de n'entrer qu'une fois dans chaque pièce.

Le plan A n'a qu'une entrée à la pièce 1.

Le plan D n'a qu'une entrée à la pièce 6.

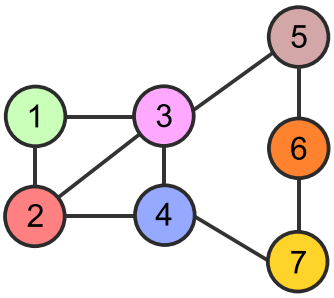
Le plan B ne permet l'entrée dans la dernière pièce, la 6, que par la pièce 5 ou la pièce 7. Si le visiteur vient de la pièce 5, il peut entrer dans la pièce 7, mais ne peut atteindre la sortie qu'en retournant dans la pièce 6 (ou l'inverse), ce qui va à l'encontre des règles.

C'est de l'informatique !

La plupart des enfants et des jeunes résolvent ce problème par tâtonnement sans utiliser de représentation abstraite supplémentaire. Pour cela, ils utilisent à un certain niveau la stratégie générale appelée *retour sur trace*. Ils reconnaissent tout au moins que l'on peut apprendre d'essais infructueux et que l'on peut, dans ce cas, revenir en arrière pour essayer une autre possibilité. Ils sont également confrontés au concept du *non-déterminisme*, car il y a souvent plusieurs possibilités à choix.

Cet exercice est un exemple d'un problème connu en informatique : la recherche d'un *chemin hamiltonien*. Dans une représentation discrète du plan au sol par un *graphe*, chaque pièce est représentée par un *nœud* et chaque porte entre deux pièces par une *arête* entre les deux nœuds correspondants.

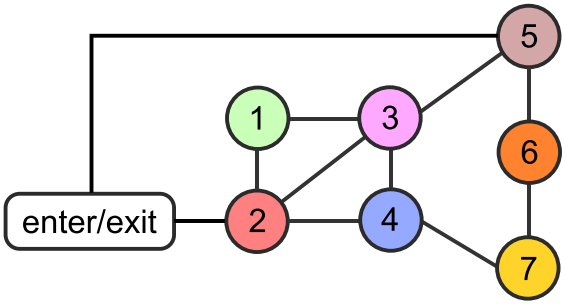
Voici la représentation abstraite du plan de l'exercice :



Il s'agit à présent de trouver dans ce graphe un chemin ayant les propriétés suivantes :

1. Le chemin commence au nœud 2 (« enter »).
2. Le chemin finit au nœud 5 (« exit »).
3. Le chemin passe exactement une fois par chaque nœud.

Si l'on représente l'espace extérieur par un nœud, ceci correspond à la recherche d'un cycle hamiltonien (un tour) dans lequel on passe également une seule fois par chaque nœud avant de terminer au nœud de départ.



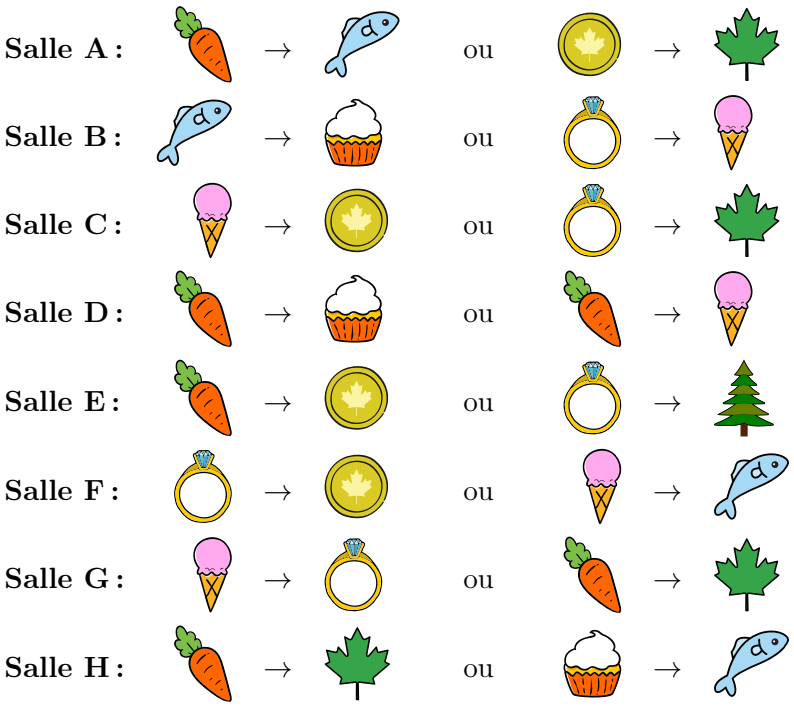
Mots clés et sites web

- Théorie de graphes, nœud, arête : `https://fr.wikipedia.org/wiki/Théorie_des_graphes`
- Chemin hamiltonien : `https://fr.wikipedia.org/wiki/Graphe_hamiltonien`

7. Troc au château

Un castor malin a besoin d'un sapin pour construire un barrage sur la rivière. Malheureusement, il n'a qu'une carotte. C'est un jour de marché au château aujourd'hui, et le castor veut y troquer sa carotte contre un sapin.

Dans chaque salle du château, deux types de troc sont proposés. Le table suivante liste ces propositions :



Dans la salle B, le castor peut par exemple troquer une bague contre une glace, mais pas l'inverse.

Dans quel ordre le castor doit-il passer dans les salles du château pour finalement avoir le sapin désiré ?

A) DGE : D'abord la salle D, puis la salle G et finalement la salle E.
B) GCE : D'abord la salle G, puis la salle C et finalement la salle E.
C) AGE : D'abord la salle A, puis la salle G et finalement la salle E.
D) DBC : D'abord la salle D, puis la salle B et finalement la salle C.

Solution

La bonne réponse est A) DGE. D'abord la salle D, puis la salle G et finalement la salle E.

Dans la salle D, le castor troque sa carotte contre une glace. Ensuite il va dans la salle G, où il troque la glace contre une bague. Finalement, le castor va dans la salle E pour troquer la bague contre un sapin.

Voici le déroulement complet :

Salle D : →

Salle G : →

Salle E : →

Il y a deux stratégies appropriées pour trouver le bon ordre dans lequel visiter les salles. La première essaie de prendre toutes les possibilités de troc en considération. Elle commence avec le premier troc, lors duquel on peut échanger la carotte dans cinq salles (A, D, E, G et H) contre six objects différents. Ensuite, toutes les possibilités de troc pour chacun de ces six objects sont considérées. C'est complexe et peut même tourner en rond, comme dans l'exemple suivant dans lequel le castor peut passer dans les salles B et H indéfiniment :

Salle A : →

Salle B : →

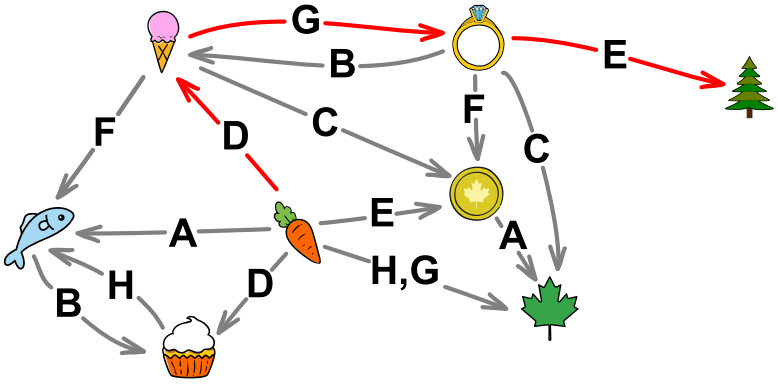
Salle H : →

Salle B : →

Cette première stratégie est donc très complexe et n'arrive rapidement à une solution qu'avec beaucoup de chance.

La deuxième stratégie atteint rapidement le but dans cet exemple concret. Le principe est de commencer la recherche par le résultat souhaité, donc le sapin. Le castor ne peut obtenir un sapin que dans la salle E. On ne reçoit un sapin qu'en échange d'une bague. Le prochain object désiré est donc une bague ! La bague ne peut aussi être obtenue que dans une salle, la salle G, en échange d'une glace. On peut obtenir une glace soit dans la salle B contre une bague, soit dans la salle D contre une carotte. Le castor malin doit donc commencer sa série de trocs dans la salle D.

On peut représenter les trocs possibles par un graphe avec des arêtes orientées (flèches) pour avoir une meilleure vue d'ensemble. Chaque nœud du graphe représente un des objects du troc et chaque arête qui en part une possibilité de troc. Sur l'arête est aussi noté le nom de la salle dans laquelle le troc est possible.



Cette représentation visuelle des objects, des possibilités de troc et des salles permet de trouver facilement comment aller de la carotte au sapin, soit en suivant un chemin sur le graphe orienté : d'abord la salle D, puis la salle G et finalement la salle E.

C'est de l'informatique !

De manière générale, on peut considérer les *processus de calcul* comme des *suites de transformations* (ici, ce sont des trocs) ou comme des *suites d'états* d'un système. L'état de départ du système est dans notre cas la carotte, et la transformation (la transition) de la carotte à la glace change cet état en glace.

Une *transition* mène donc d'un état à un autre. On appelle aussi un suite de transitions un *calcul*.

Cet exercice traite donc aussi de calcul à un niveau très général. Le système de l'exemple est *non déterministe* : cela veut dire qu'il y a parfois plusieurs étapes de calcul possibles, comme il y a plusieurs possibilité de trocs dans l'exercice. Le non-déterminisme est un concept important dans la modélisation en informatique. Les étapes de calcul possibles sont décrites par des règles de transformation (la table montrant les possibilités de troc). Il s'agit du célèbre *problème d'accessibilité* lorsque l'on veut déterminer si le castor peut troquer une carotte contre un sapin, donc si un certain *état final* peut être atteint depuis un certain *état initial* — un problème qui a de nombreuses applications.

L'exercice ci-dessus montre que c'est parfois une bonne idée de chercher l'état initial en partant de l'état final plutôt que le contraire. Cette stratégie s'appelle aussi *recherche en arrière*.

En comparant les différentes stratégies pour résoudre le problème, on voit que le graphe orienté offre une possibilité claire de représenter l'*espace d'états* d'un système avec toutes les transitions possibles entre les états. Dans ce modèle de base, on pourrait appliquer les algorithmes de parcours de graphes fondamentaux, comme le *parcours en largeur* et le *parcours en profondeur*.

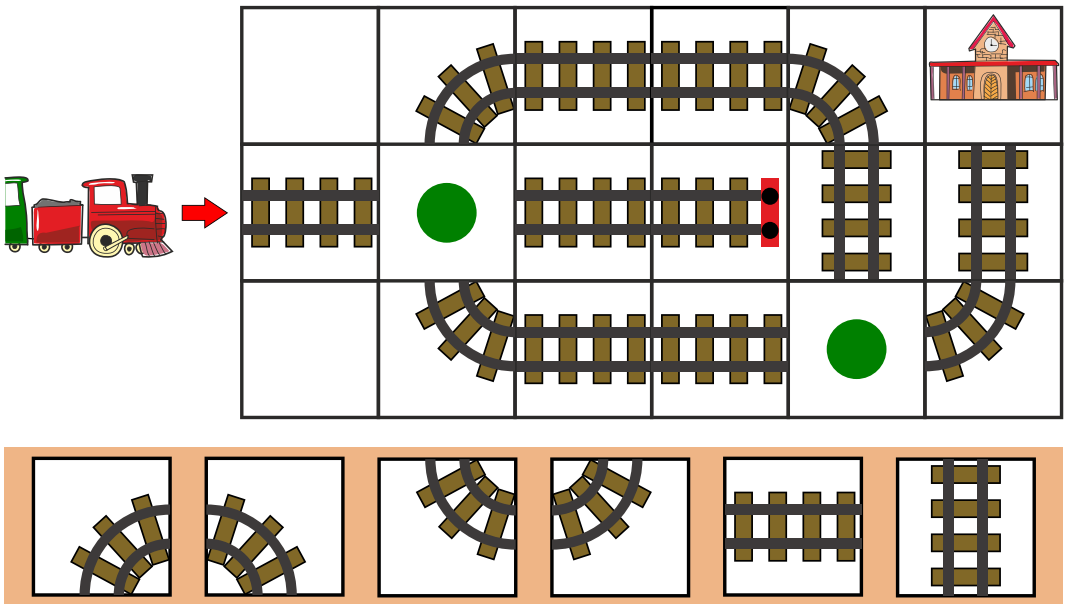
Mots clés et sites web

- Théorie des graphes : `https://fr.wikipedia.org/wiki/Théorie_des_graphes`
- Problème d'accessibilité : `https://fr.wikipedia.org/wiki/Problème_d'accessibilité`

- Parcours en profondeur :
  `https://fr.wikipedia.org/wiki/Algorithme_de_parcours_en_profondeur`
- Parcours en largeur :
  `https://fr.wikipedia.org/wiki/Algorithme_de_parcours_en_largeur`

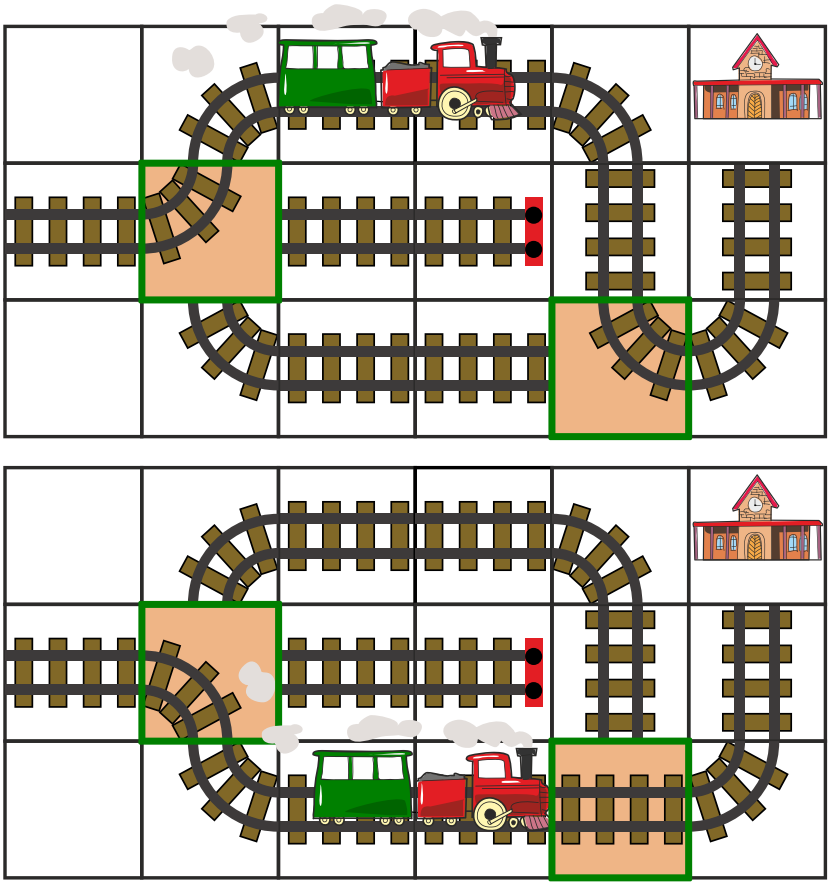
8. Prochain arrêt, gare !

Choisis les bons rails pour les deux cases avec les points verts de façon à ce que le train puisse aller à la gare .



Solution

Ce problème a les deux solutions suivantes :



Avec toutes les autres combinaisons, le train déraille ou fonce dans le buttoir.

C'est de l'informatique !

Comme un train qui suit simplement les rails en roulant, un ordinateur suit simplement les instructions d'un programme. Il ne peut pas savoir si le programme contient une erreur et peut alors « planter », comme un train peut dérailler si les rails ne sont pas assemblés correctement. On doit donc être beaucoup plus attentif en écrivant un programme que lorsque l'on indique la direction de la gare à quelqu'un, par exemple.

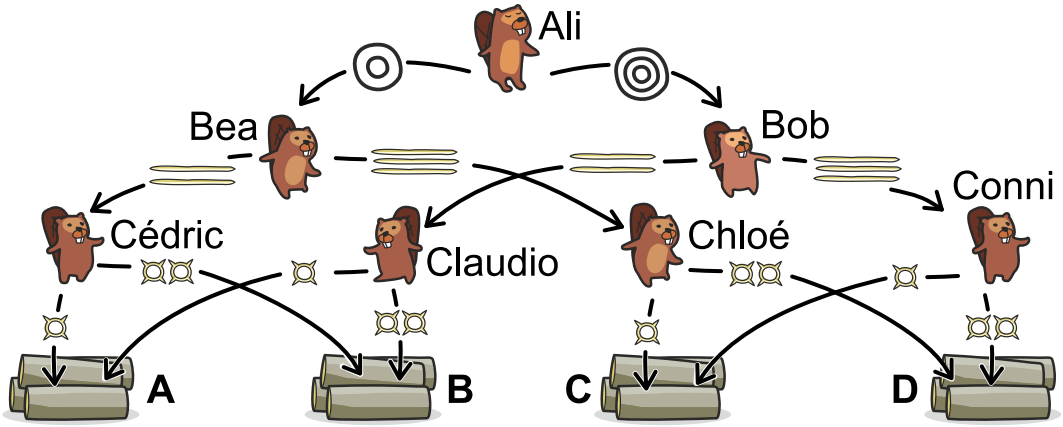
Dans cet exercice, il s'agit d'ajouter les instructions manquantes aux bons endroits d'un programme pour pouvoir atteindre l'objectif.

Mots clés et sites web

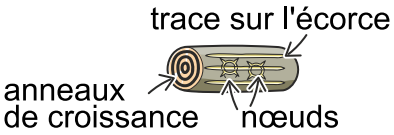
- Programme
- Instruction : `https://fr.wikipedia.org/wiki/Instruction_informatique`
- `https://fr.wikipedia.org/wiki/Algorithme`

9. Piles de troncs d'arbres

Dans le village des castors, les troncs d'arbres sont répartis dans quatre groupes (A, B, C, D) d'après trois propriétés (le nombre d'anneaux de croissance, le nombre de traces sur l'écorce et le nombre de nœuds dans le bois). Le diagramme de décision montre comment cela se passe.



Par exemple, ce tronc-ci est posé sur la pile D en raison des décisions suivantes :



- Ali voit trois anneaux de croissance et donne le tronc à Bob ;
- Bob voit trois traces sur l'écorce et donne le tronc à Conni ;
- Conni voit deux nœuds dans le bois et pose le tronc sur la pile D.

Sur quelle pile ce tronc va-t-il être posé ?

A) Pile A
B) Pile B
C) Pile C
D) Pile D

Solution

La bonne réponse est la pile C, car Ali voit deux anneaux de croissance et donne le tronc à Bea. Bea voit trois traces sur l'écorce et transmets le tronc à Chloé. Chloé voit un nœud dans le bois et pose le tronc sur la pile C.

Si l'on veut, on peut déterminer quels troncs correspondent à chaque pile. Il y a deux types de troncs sur chaque pile.

Sur la pile **A** :

- Les troncs avec 2 anneaux de croissance, 2 traces sur l'écorce et 1 nœud dans le bois.
- Les troncs avec 3 anneaux de croissance, 2 traces sur l'écorce et 1 nœud dans le bois.

Sur la pile **B** :

- Les troncs avec 2 anneaux de croissance, 2 traces sur l'écorce et 2 nœuds dans le bois.
- Les troncs avec 3 anneaux de croissance, 2 traces sur l'écorce et 2 nœuds dans le bois.

Sur la pile **C** :

- Les troncs avec 2 anneaux de croissance, 3 traces sur l'écorce et 1 nœud dans le bois.
- Les troncs avec 3 anneaux de croissance, 3 traces sur l'écorce et 1 nœud dans le bois.

Sur la pile **D** :

- Les troncs avec 2 anneaux de croissance, 3 traces sur l'écorce et 2 nœuds dans le bois.
- Les troncs avec 3 anneaux de croissance, 3 traces sur l'écorce et 2 nœuds dans le bois.

C'est de l'informatique !

Cet exercice touche à plusieurs concepts informatiques.

En premier lieu, le concept des *diagrammes de décision* est traité, diagrammes qui ont des applications très variées en informatique. Ici, on les utilise pour la *classification* d'objects dans certaines catégories (très souvent, on utilise des arbres de décision, une forme spéciale de diagrammes de décision. Le diagramme de décision de l'exercice n'est pas un arbre de décision, car deux groupes sont posés sur la même pile au dernier niveau du diagramme).

On peut aussi voir le diagramme de décision de cet exercice comme la représentation abstraite des valeurs d'une fonction à plusieurs variables. La terminologie exacte en informatique est *diagramme de décision binaire*.

De plus, on aborde ici le concept des *attributs* (caractéristiques ou propriétés) d'objets. Dans cet exemple, les objects ont trois attributs (anneaux de croissance, trace sur l'écorce, nœuds dans le bois), et chaque attribut a deux valeurs possible (deux ou trois anneaux ou traces et un ou deux nœud[s]).

Il y a beaucoup d'applications possibles pour de tel diagramme de décision. L'une d'entre elles est la classification de paquets envoyés sur un réseau (par des routeurs ou des commutateurs réseau).

Mots clés et sites web

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

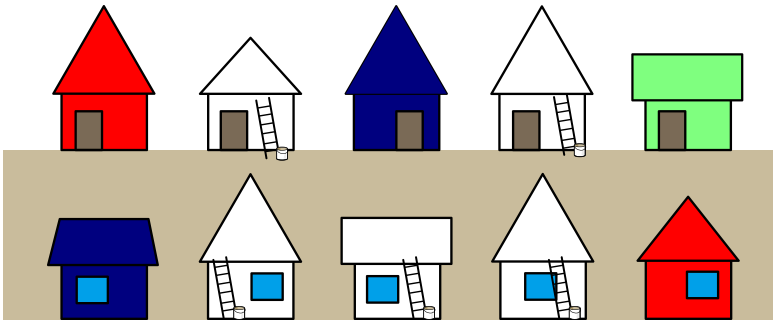
10. Quartier coloré

Les habitants d'une rue veulent peindre leurs maisons blanches en couleur. Chaque maison doit être peinte en l'une de ces trois couleurs : vert clair, rouge ou bleu foncé. Pour que ça n'ait pas l'air ennuyeux, ils suivent les règles suivantes :

- Deux maisons directement l'une à côté de l'autre ne doivent pas être de la même couleur.
- Deux maisons directement face à face dans la rue ne doivent pas avoir la même couleur.

Quelques habitants ont déjà peint leur maison en couleur. Les autres habitants doivent maintenant peindre leur maison de manière à respecter les règles.

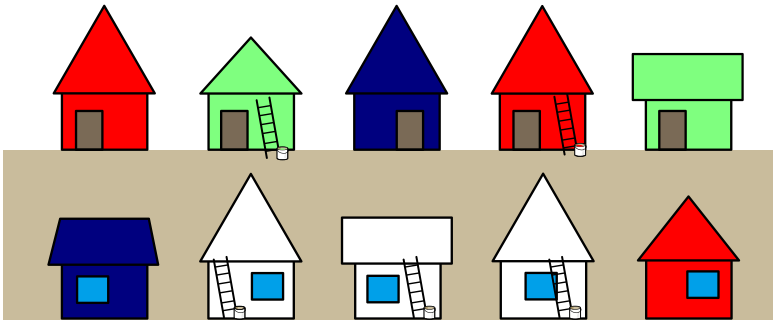
Trouve les couleurs appropriées pour les habitants.



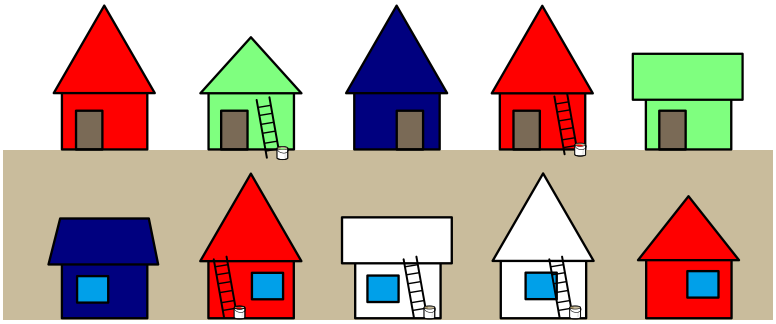
Solution

Le plus facile est de trouver la couleur de chaque maison l'une après l'autre.

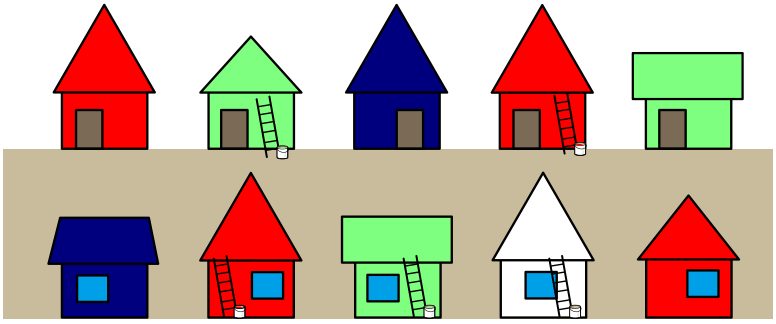
Les deux maisons blanches du côté supérieur de la route sont chacune entourées de deux maisons de couleurs différentes à gauche et à droite. On ne peut donc les peindre qu'en une seule couleur si l'on suit les règles : La maison blanche en haut à gauche en vert clair et la maison blanche en haut à droite en rouge.



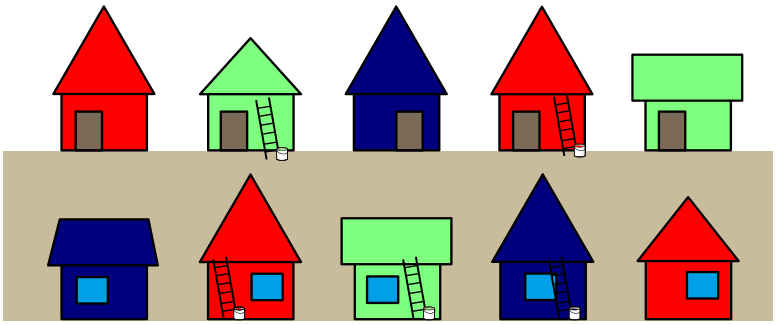
Ensuite, on peut voir que la maison blanche en bas à gauche doit être peinte en rouge, car la maison directement à sa gauche est bleu foncé et la maison directement en face est vert clair :



On peut faire presque le même raisonnement pour la maison au milieu du côté inférieur de la rue : Elle doit être peinte en vert clair, car directement à sa droite est la maison que l'on vient de peindre en rouge et directement en face se trouve une maison bleu foncé.



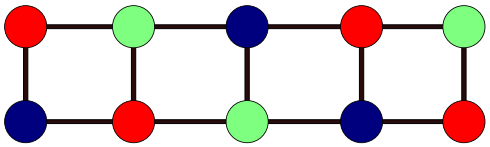
Finalement, on peut aussi trouver la couleur appropriée pour la maison blanche en bas à droite : la maison directement à sa droite et celle directement en face sont les deux rouges, mais comme la maison directement à sa gauche est maintenant vert clair, il ne reste plus que la possibilité de peindre la maison en bleu foncé :



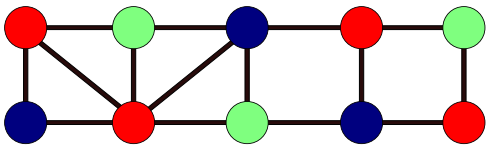
C'est de l'informatique !

Vu de manière abstraite, il s'agit dans cet exercice de trouver une solution qui satisfasse certaines contraintes (règles). C'est un problème rencontré très souvent en informatique.

Les maisons et leur voisinage direct (aussi bien à gauche qu'à droite et que de l'autre côté de la rue en face) peuvent très bien être représentées à l'aide d'un *graphe*, une structure de données très répandue en informatique. Dans ce graphe, chaque maison est un *nœud* est chaque lien de voisinage direct est une *arête* :



Sur l'image, les nœuds sont déjà colorés comme les maisons correspondantes. Les maisons devaient suivre la règle de ne pas avoir la même couleur que leurs voisines. C'est pour cela que les nœuds reliés directement par une arête sur l'image ne sont jamais de la même couleur. Le fait qu'il existe une *coloration valable* du graphe avec trois couleurs ne va pas de soi. Si l'on ajoute deux arêtes au graphe comme sur l'image suivante, il n'y a plus de coloration valable : qu'importe comment on répartit les couleurs dans ce graphe, il y a toujours deux nœuds directement reliés qui sont de la même couleur.



C'est à nouveau possible en utilisant quatre couleurs. Peut-être que c'est toujours possible avec quatre couleurs ? La réponse est à nouveau non. Mais il existe au moins une certaine sorte de graphe que l'on peut toujours colorer de manière valable avec quatre couleurs : on les appelle les graphes planaires. Ce sont des graphes que l'on peut dessiner sans que leurs arêtes ne se croisent (le graphe sur la dernière image n'est pas planaire à cause des liens entre les quatre nœuds à gauche). Le fait

que les graphes planaires aient toujours une coloration à quatre couleurs valable est décrit par le *théorème des quatre couleurs*.

Le théorème des quatre couleurs est spécialement intéressant pour la réalisation de cartes géographiques. Si l'on se représente chaque pays comme un nœud et que l'on relie les pays voisins par une arête, on obtient toujours un graphe planaire (pour être exact, nous devons pour cela exclure l'existence d'enclaves et d'exclaves, c'est-à-dire de parties de pays se trouvant au milieu d'un autre pays). On peut donc colorer ces graphes de manière valable avec quatre couleurs, et on peut donc aussi colorier ces pays sur la carte de manière à ce que les pays voisins ne soient jamais de la même couleur.

La preuve que quatre couleurs suffisent n'est pas facile à faire. On savait déjà il y a 200 ans que cinq couleurs suffisent. La preuve que quatre couleurs suffisent a été faite en 1976 par les mathématiciens Kenneth Appel and Wolfgang Haken. Ils ont pour cela utilisé un ordinateur pour tester un grand nombre d'exceptions et de contre-exemples. L'ordinateur a eu besoin de plus de mille heures pour faire cela. Cela aurait été totalement impossible à faire à la main. Beaucoup de mathématiciens se sont demandé si une telle preuve était valable, car il faut pour cela faire confiance à l'ordinateur.

Mots clés et sites web

- Théorème des quatre couleurs :
  `https://fr.wikipedia.org/wiki/Théorème_des_quatre_couleurs`
- Coloration de graphe : `https://fr.wikipedia.org/wiki/Coloration_de_graphe`

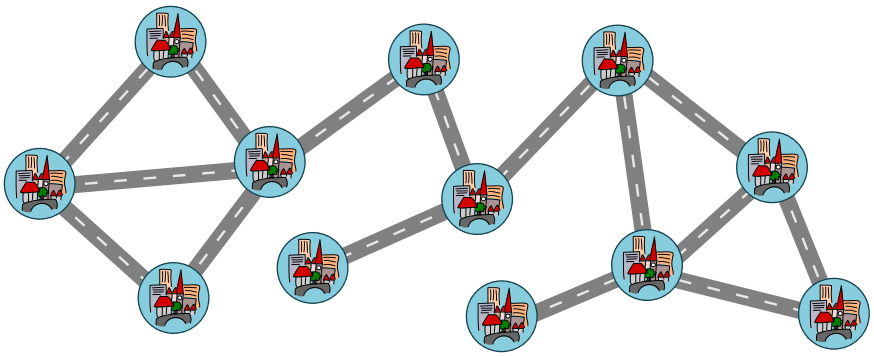
11. Épidémiologie

Castorland comporte 12 villes qui sont reliées par des routes. Les villes qui sont reliées de manière directe ou indirecte forment une communauté commerciale. La carte dans sa forme actuelle montre donc une seule communauté commerciale de 12 villes.

Pour endiguer une épidémie, la circulation doit être réduite. Le parlement des castors décide de fermer exactement deux routes pour diviser les villes en trois communautés commerciales.

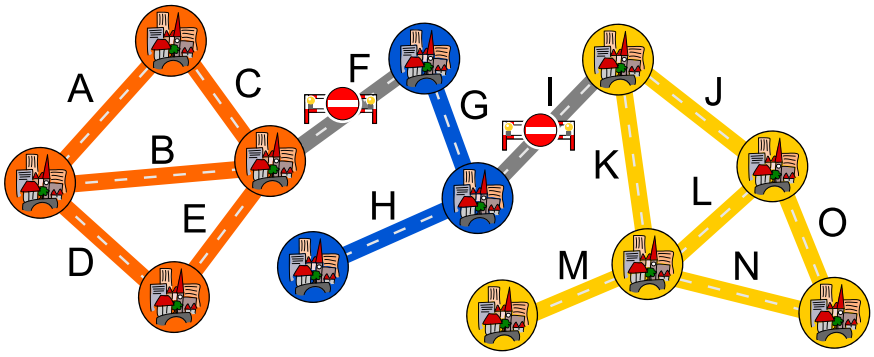
Pour n'isoler personne plus que nécessaire, la plus petite communauté commerciale devrait compter autant de villes que possible

Quelles sont les deux routes qui doivent être fermées ? Biffe-les.



Solution

La bonne réponse est : les routes F et I sur l'image ci-dessous doivent être fermées. De cette manière, trois communautés commerciales de 3, 4 et 5 villes, respectivement, sont formées.



C'est évident que nous ne devons considérer que les routes dont la fermeture engendre une division de la communauté commerciale car elles représentent une connexion unique. Nous avons en effet besoin de deux vraies divisions pour créer trois unités. Ça n'apporte donc rien de fermer la route B, par exemple, car on peut encore atteindre toutes les villes en passant par les routes A ou C. Les seules candidates pour la fermeture sont donc les routes F, G, H, I et M.

On arrive à la réponse du dessus en essayant toutes les dix possibilités de fermer deux de ces cinq routes. En tant qu'être humain, on remarque de plus tout de suite que la fermeture des routes H ou M isolerait une seule ville et n'entre donc pas en question. Cela limite encore le nombre de possibilités à considérer.

C'est de l'informatique !

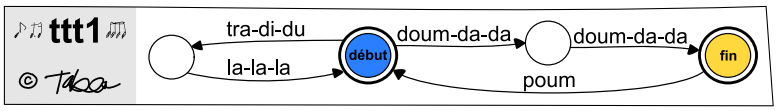
En informatique, on cherche souvent à diviser un certain réseau en *composantes connexes*. Toutes les parts d'une composante connexe sont reliées de manière directe ou indirecte, alors qu'il n'y a aucun lien entre différentes composantes connexes. L'utilisation dans les réseaux informatiques dans lesquels il est important de déterminer quels ordinateurs peuvent être atteints depuis quels autres est évidente. C'est aussi important de déterminer quels points sont reliés dans la reconnaissance optique de caractères (OCR).

Mots clés et sites web

- Composante connexe : `https://fr.wikipedia.org/wiki/Graphe_connexe`
- Parcours d'arbre : `https://fr.wikipedia.org/wiki/Parcours_d'arbre`

12. Les textes tendres de Tabea

Tabea a beaucoup de succès avec ses textes de chanson de la marque ttt : les textes tendres de Tabea. Ceux-ci peuvent être produits avec le diagramme ttt1 suivant :



Pour produire une chanson, Tabea commence par le « début » (début) et suit l'une des flèches partant de ce point. Lorsqu'il y a plusieurs possibilités, elle choisit. Elle chante les syllabes correspondantes le long de chemin dans l'ordre donné. Lorsqu'elle atteint la « fin » (fin), sa chanson peut se terminer ou continuer.

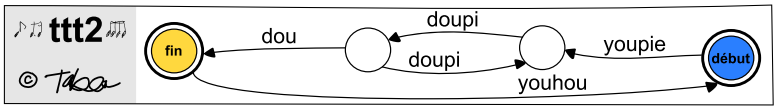
Voici des exemples de chansons possibles :

« Tra-di-du La-La-La Tra-di-du La-La-La
Doum-da-da Doum-da-da Poum Doum-da-da Doum-da-da »

Ou

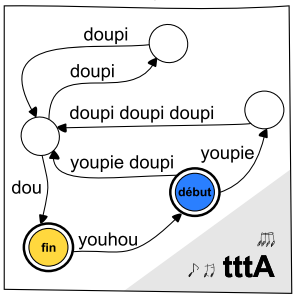
« Doum-da-da Doum-da-da Poum Tra-di-du La-La-La
Doum-da-da Doum-da-da Poum Tra-di-du La-La-La
Doum-da-da Doum-da-da Poum Doum-da-da Doum-da-da »

En novembre 2020, Tabea commence la production avec les nouveaux textes ttt2 :

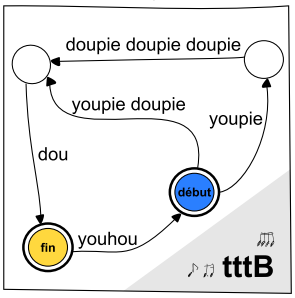


Lequel des diagrammes suivants permet de produire exactement les mêmes textes que ttt2 ?

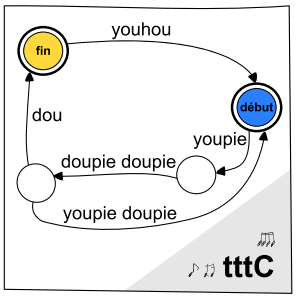
A)



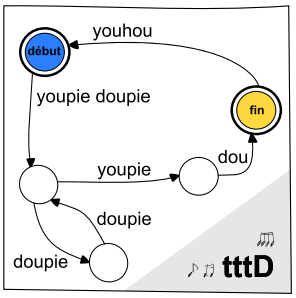
B)



C)

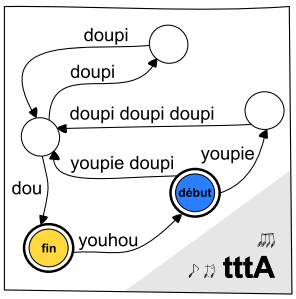


D)

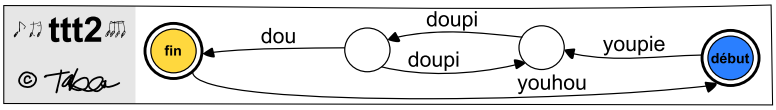


Solution

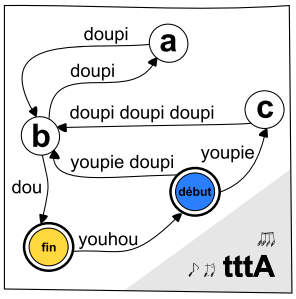
La bonne réponse est A), le diagramme tttA :



Lorsque l'on produit une chanson avec ttt2, elle commence dans tous les cas avec « youpie » qui est suivi d'au moins un « doupie ». On peut ensuite continuer soit avec « dou », soit avec un nombre pair de « doupie » suivi de « dou ». La chanson peut ensuite se terminer ou continuer avec un « youhou » avant de reprendre depuis le début.



Le diagramme tttA permet de produire exactement la même chose : depuis le « début », la chanson peut aller directement à **b** et ainsi commencer avec « youpie doupie » ou y aller en passant par **c** avec « youpie doupie doupie doupie ». Ensuite viennent, avec un détour par **a**, un nombre pair de « doupie » avant d'arriver à la fin de la chanson avec « dou ». Comme avec ttt2, on peut alors continuer la chanson avec un « youhou ».



Aussi bien ttt2 que tttA peuvent générer un nombre impair de « doupie » l'un après l'autre, après le « youpie » du début. Par contre, tttB ne peut générer qu'un ou trois « doupie » qui se suivent, et tttC seulement un ou deux. Quant à tttD, il peut certes générer un nombre impair de « doupie » l'un après l'autre, mais va toujours insérer, avant le « dou » final, un « youpie » de plus, ce que ttt2 ne fait pas.

La bonne réponse est donc tttA.

C'est de l'informatique !

Une tâche importante de l'informatique est de trouver des structures dans des données, par exemple dans le langage comme celui du texte d'une chanson. Les diagrammes représentent ce que l'on

appelle des automates finis, qui utilisent des règles très strictes pour générer et reconnaître certains langages. C'est primordial dans le développement des langages de programmation. Dans notre exemple, l'automate fini décrit l'ensemble des chansons que celui-ci peut générer.

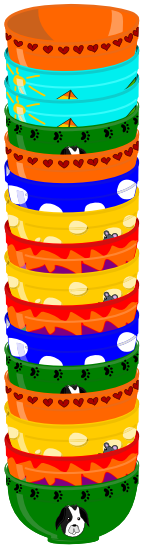
La reconnaissance de motifs est également importante dans beaucoup d'autres domaines, comme par exemple le traitement du langage naturel.

Mots clés et sites web

- Automate fini : `https://fr.wikipedia.org/wiki/Automate_fini_déterministe`
- Langage formel : `https://fr.wikipedia.org/wiki/Langage_formel`
- `https://sites.google.com/isabc.ca/computationalthinking/pattern-recognition`

13. Bols

Trois frères et sœurs veulent manger leur petit-déjeuner dans trois bols pareils. Ils ont une grande pile de bols. Par précaution, ils n'enlèvent toujours qu'un bol à la fois du haut de la pile.

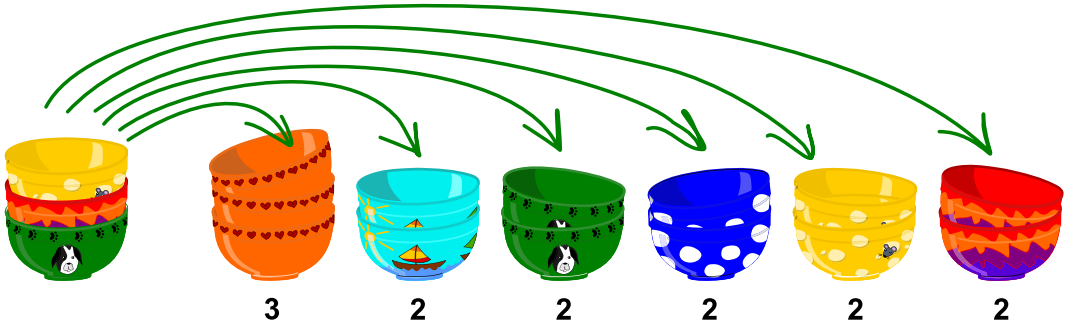


Quel est le plus petit nombre de bols qu'ils doivent enlever de la pile dessinée pour en avoir trois pareils ?

A) 3 bols
B) 4 bols
C) 5 bols
D) 6 bols
E) 7 bols
F) 8 bols
G) 9 bols
H) 10 bols
I) 11 bols
J) 12 bols
K) 13 bols
L) 14 bols
M) 15 bols
N) 16 bols

Solution

Réponse K) : Au moins 13 bols doivent être enlevés de la pile pour avoir trois bols pareils.



C'est de l'informatique !

Une *pile*, aussi appelée *stack* en informatique, est une façon très répandue d'enregistrer des choses. Une pile est une structure très simple, mais très puissante, que l'on utilise souvent en programmation. Il y a des règles qui décrivent comment on peut ajouter ou enlever des choses de la pile, le plus souvent seulement depuis le haut. Dans cet exercice, nous n'avons travaillé qu'avec des choses à enlever de la pile. La règle indique que seul l'objet le plus haut de la pile peut être enlevé. Si l'on veut le dixième bol de la pile, on doit donc enlever dix bols les uns après les autres. Pour cela, c'est important d'avoir un autre endroit à disposition où poser les neuf autres bols ; c'est pareil en programmation. Si l'on a une deuxième pile et que les piles peuvent être aussi hautes que l'on veut, on peut en théorie déjà tout calculer ce qui est calculable avec un ordinateur (en informatique, on dit d'une telle chose qu'elle est *complète au sens de Turing*) ! De simples piles comme ça sont vraiment puissantes !

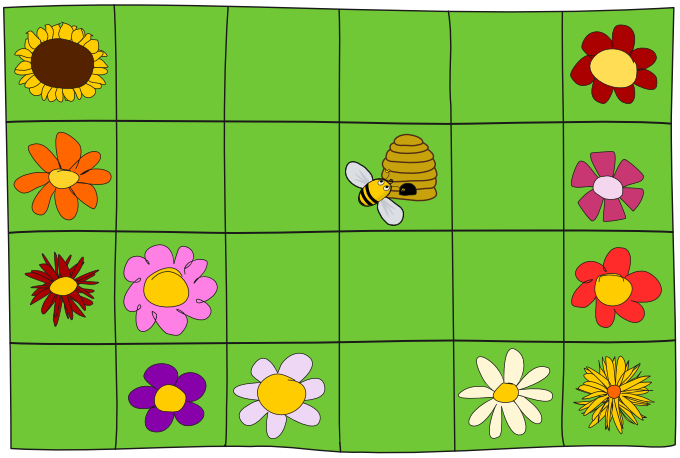
Mots clés et sites web

- Pile : `https://fr.wikipedia.org/wiki/Pile_(informatique)`
- Machine de Turing : `https://fr.wikipedia.org/wiki/Machine_de_Turing`

14. Abeille assidue

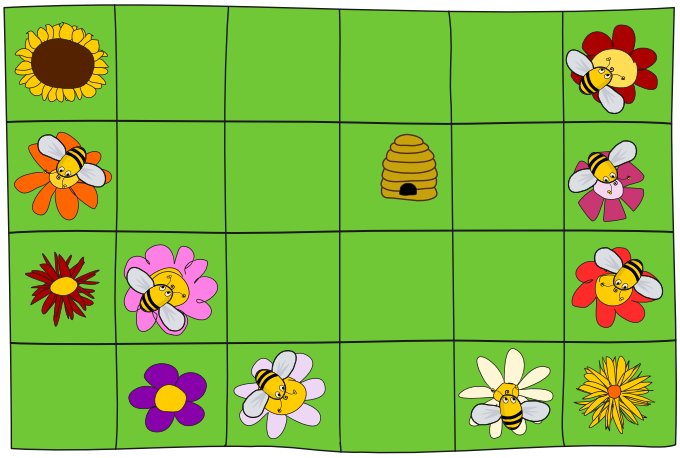
Une abeille met 10 minutes pour voler d'une case vers le haut, le bas, la gauche ou la droite. Après être partie de la ruche, elle vole pendant 30 minutes au maximum avant de revenir en arrière.

Entoure les fleurs qui peuvent être atteintes en 30 minutes au maximum depuis la ruche.



Solution

Les fleurs sur lesquelles une abeille est posée peuvent être atteinte en 30 minutes au maximum depuis la ruche :



L'image ci-dessous montre pour chaque fleur le nombre de minutes dont l'abeille a besoin pour y arriver en partant de la ruche. En une demi-heure, l'abeille pour donc atteindre toutes les cases dans lesquelles 10, 20 ou 30 est écrit.

Pour remplir les cases avec les nombres, on procède ainsi : dans les cases à côté de la ruche, on écrit 10, car l'abeille met 10 minutes à y arriver depuis la ruche. Ensuite, on écrit 20 dans toutes les cases vides à côté des cases « 10 », car l'abeille met 10 minutes pour passer d'une case à une autre. On continue à faire comme cela. On écrit donc 30 dans toutes le cases vides à côté des cases « 20 », puis 40 dans toutes les cases vides à côté des cases « 30 », et pour finir 50 dans toutes les cases vides à côté des cases « 40 ».

C'est de l'informatique !

Pour résoudre l'exercice, nous avons calculé pour chaque case le temps dont l'abeille a besoin pour y arriver depuis la ruche. D'abord, on détermine quelles cases sont atteignables en 10 minutes. On

les utilise ensuite pour déterminer quelles cases sont atteignables en 20 minutes. À l'aide des cases éloignées de 20 minutes, on trouve les cases atteignables en 30 minutes, et ainsi de suite.

Nous utilisons donc des résultats déjà calculés et enregistrés (les nombres dans les cases remplies) pour calculer les résultats suivants (les nombres dans les cases voisines encore vides). Ce principe très général s'appelle *programmation dynamique*. L'ordre dans lequel les résultats sont calculés est pour cela en général très important. Il faut aussi y faire attention pour le vol de l'abeille.

Dans l'exercice, l'abeille ne vole que vers le haut, le bas, la gauche ou la droite en 10 minutes. C'est un peu irréaliste, car en réalité, une abeille vole sûrement aussi en diagonale par dessus les cases. Avec cette hypothèse plus réaliste, les cases atteignables en 30 minutes seraient délimitées par un cercle et non par un losange comme dans l'exercice.

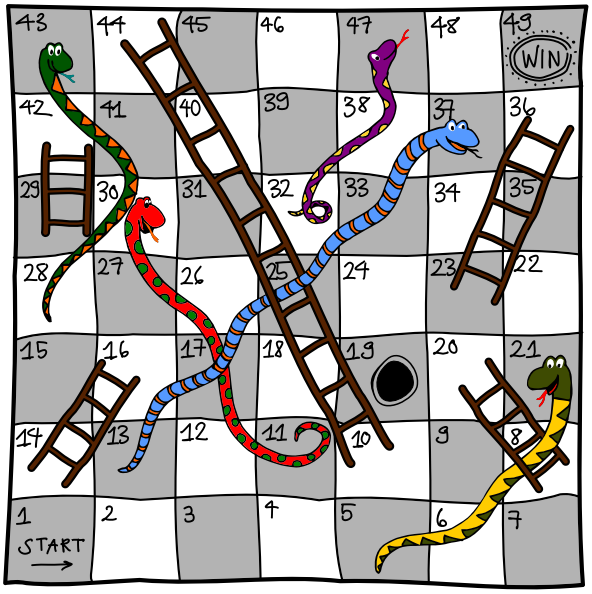
La mesure de distance habituelle qui génère un cercle s'appelle *distance euclidienne*. La mesure de distance utilisée dans l'exercice avec laquelle on ne peut se déplacer que verticalement ou horizontalement sur des carrés s'appelle la *distance de Manhattan* (le nom vient de l'arrangement quadrillé des villes modernes comme Manhattan).

Mots clés et sites web

- Programmation dynamique : `https://fr.wikipedia.org/wiki/Programmation_dynamique`
- Distance euclidienne : `https://fr.wikipedia.org/wiki/Distance_(mathématiques)`
- Distance de Manhattan : `https://fr.wikipedia.org/wiki/Distance_de_Manhattan`
- `https://fr.wikipedia.org/wiki/Espace_euclidien`

15. Serpents et échelles

Dans le jeu Serpents et échelles, tous les joueurs commencent sur la case 1. Le gagnant est le joueur qui arrive en premier à la case 49. À chaque tour, on jette le dé et avance son pion du nombre de cases correspondant (entre 1 et 6).



Si l'on arrive sur une case avec la tête d'une serpent, on glisse vers le bas jusqu'à la case contenant le bout de la queue du serpent. Par contre, si l'on arrive au pied d'une échelle, on peut monter jusqu'à la case contenant le dernier échelon dans le même tour.

Par exemple : tu es sur la case 26, jettes le dé et obtiens un 3, tu avances jusqu'à la case 29 et peux directement monter jusqu'à la case 42. Au tour suivant, tu obtiens un 5 et arrives sur la tête du serpent de la case 47, tu dois redescendre à la case 32.

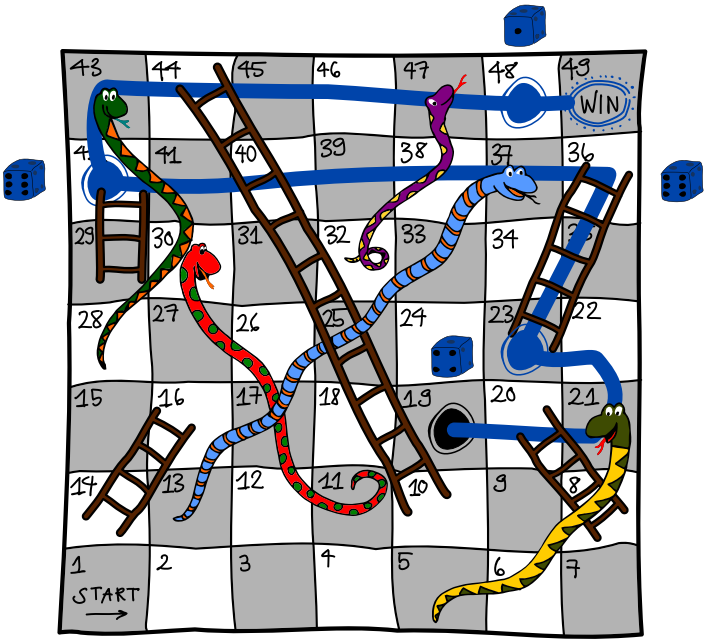
Ton pion est sur la case 19. De combien de tours au minimum as-tu besoin pour atteindre la case 49 ?

A) 2 tours
B) 3 tours
C) 4 tours
D) 5 tours

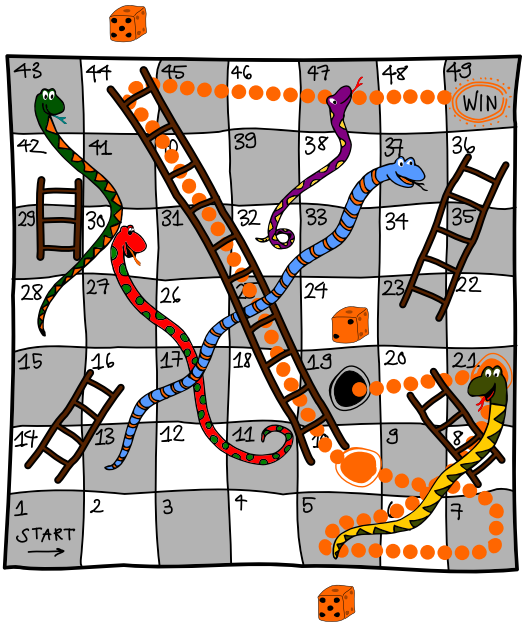
Solution

La bonne réponse est B) 3 tours.

Si tu es impatient et ne prends en compte que les jets de dés qui te rapprochent directement du but, Il te faut au minimum quatre tours : avec un 4, on passe de la case 19 à la case 23, puis par échelle à la case 36. Depuis là, il n'y a plus d'autres échelles vers le haut et il faut trois jets de dé supplémentaires, par exemple 6 – 6 – 1, pour arriver au but.



Par contre, si tu acceptes un apparent détour, tu peux y arriver en trois tours avec les jets de dé 2 – 5 – 5. Tu passes de la case 19 à la case 21, puis glisse en bas du serpent jusqu'à la case 5. Depuis là, tu vas à la case 10, puis jusqu'en haut l'échelle à la case 44 avant d'arriver au but.



Le but ne peut pas être atteint en deux tours. Seules les cases 48, 46, 45 et 44 sont à un tour du but, et aucune de ces cases ne peut être atteinte en un tour depuis la case 19.

C'est de l'informatique !

On peut résoudre beaucoup de problèmes en cherchant le chemin le plus court entre deux points. Le mot « court » n'a ici pas le sens qu'on lui donne intuitivement. Dans cet exercice, nous avons par exemple cherché le chemin durant le moins de tours et non pas le chemin passant par le moins de cases. D'après le même principe, les systèmes de navigations proposent de chercher le chemin le plus court au niveau de la distance ou au niveau du temps nécessaire. Les mêmes appareils calculent les chemins avec le moins de frais de péages pour les entreprises de logistique.

En informatique, les mêmes procédés (algorithmes) peuvent souvent être utilisés pour des tâches complètement différentes si celles-ci sont modélisées de manière adaptée.

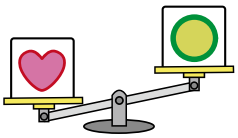
Mots clés et sites web

- Plus court chemin : `https://fr.wikipedia.org/wiki/Problème_de_plus_court_chemin`, `https://fr.wikipedia.org/wiki/Algorithme_de_Dijkstra`
- Serpents et échelles : `https://fr.wikipedia.org/wiki/Serpents_et_échelles`

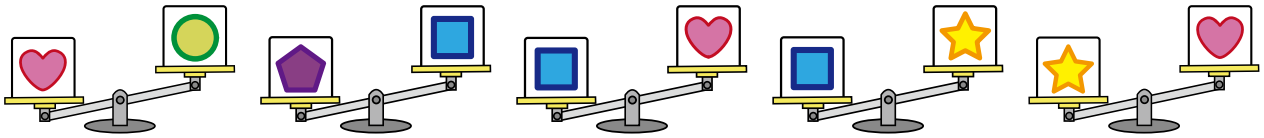
16. Lourdes comparaisons

Cinq boîtes sont marquées de cinq symboles différents :  .

Une balance est utilisée pour comparer deux boîtes. La comparaison suivante montre par exemple que  est plus lourde que  :



En tout, cinq comparaisons ont lieu :



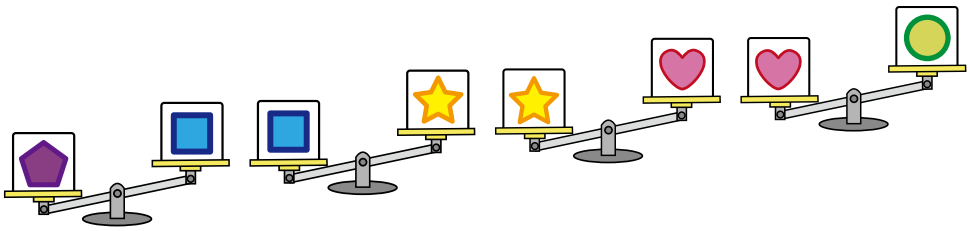
Quelle est la boîte la plus lourde ?

A) 
B) 
C) 
D) 
E) 

Solution

La boîte C) avec le pentagone est la plus lourde.

L'image suivante montre quatre des cinq comparaisons faites et toutes les boîtes :



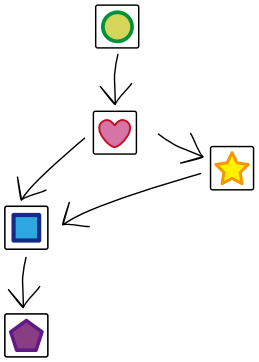
Comme cela, on voit tout de suite que : la boîte avec le pentagone est plus lourde que la boîte avec le carré. La boîte avec le carré est plus lourde que la boîte avec l'étoile. La boîte avec l'étoile est plus lourde que la boîte avec le cœur. Et finalement, la boîte avec le cœur est plus lourde que la boîte avec le cercle.

On peut déduire de cela que la boîte avec le pentagone est plus lourde que chacune des autres. Cela vient d'une propriété spéciale de la comparaison de poids : Si A est plus lourd que B et que B est plus lourd que C, alors A est aussi plus lourd que C. Cette propriété évidente s'appelle la *transitivité*.

Il existe une méthode maline pour raccourcir cet exercice. Comme on cherche *la* boîte la plus lourde, il suffit de cherche la boîte qui n'est jamais plus légère qu'une autre, et cela n'est vrai que de la boîte avec le pentagone.

C'est de l'informatique !

En fin de compte, il s'agit dans cet exercice de trier des objects quelconques. En informatique, on utilise souvent des *graphes* spéciaux pour trier, qui sont composé de *nœuds* (les objects à trier) et d'*arêtes* (les comparaisons entre les objects). Dans cet exercice, les objects sont les boîtes et les comparaisons sont les pesées. En dessinant les arêtes comme des flèches montrant l'objet qui est plus lourd, on obtient le graphe suivant pour cet exercice :



Les objects doivent à présent être arrangés sur une ligne de manière à ce que les flèches aillent toujours de gauche à droite. Un tel arrangement s'appelle un *tri topologique*. On obtient un tri topologique

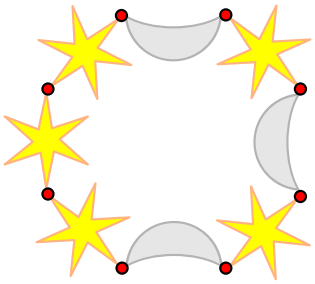
facilement en enlevant étape par étape un object sur lequel n'arrive aucune flèche, puis en mettant les objects ainsi enlevés les uns après les autres dans le même ordre.

Mais attention : ce ne sont pas tous les graphes qui ont un tri topologique. Il n'en existe par exemple pas s'il y a un endroit dans le graphe où trois arêtes fléchées sont dirigées de manière à former un cercle lorsqu'on les suit.

Mots clés et sites web

- Transitivité : `https://fr.wikipedia.org/wiki/Relation_transitive`
- Graphe : `https://fr.wikipedia.org/wiki/Théorie_des_graphes`
- Tri topologique : `https://fr.wikipedia.org/wiki/Tri_topologique`

17. Bracelet céleste



Marie aimerait le bracelet à droite.

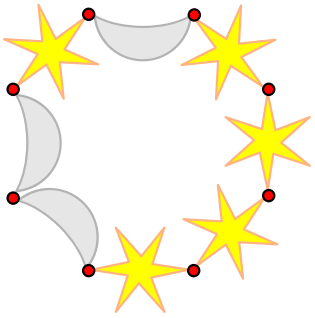
Elle donne donc les instructions suivantes à Jonas :

- Prends une étoile (★) et une lune (☾) et relie-les pour former une paire. Répète ceci trois fois en tout afin d'avoir trois paires.
- Prends ces trois paires, tourne-les comme tu veux, et relie-les pour former une longue chaîne.
- Ajoute deux étoiles supplémentaires à l'un des bouts de la chaîne. Relie maintenant les deux bouts de la chaîne pour obtenir un bracelet.

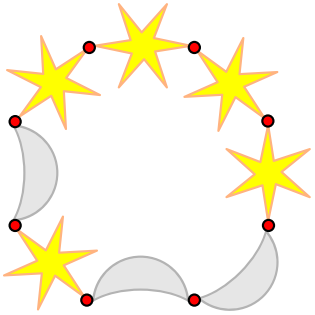
Jonas n'a pas d'image du bracelet désiré. C'est possible que Jonas obtienne un bracelet complètement différent, même s'il suit exactement les instructions de Marie.

L'un de ces quatre bracelets ne peut ***pas*** *être obtenu par Jonas s'il suit exactement les instructions de Marie. Lequel ?*

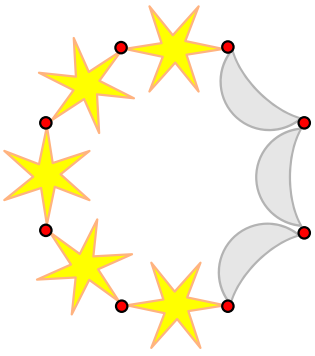
A)



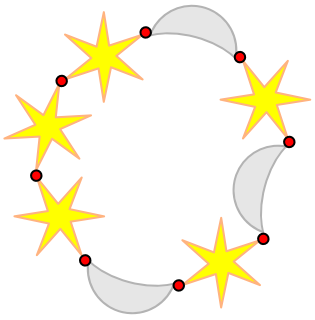
B)



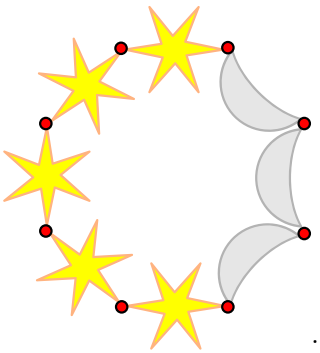
C)



D)

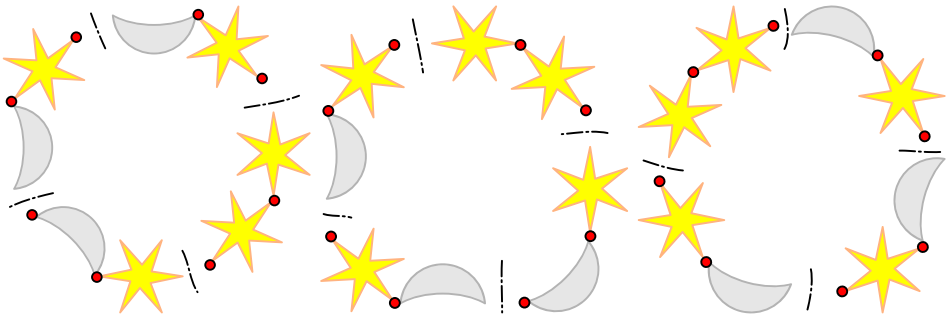


Solution

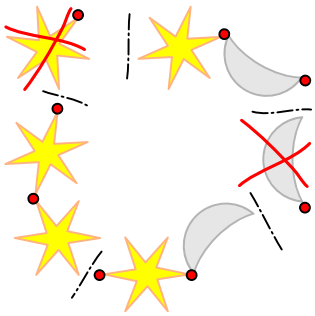
La bonne réponse est C)  .

Seul ce bracelet ne peut pas être obtenu en suivant les instructions de Marie.

Les bracelets des trois autres réponses sont par contre corrects d'après les instructions de Marie. On peut le voir, par exemple, en observant que chacun de ces bracelets peut être séparé en trois paires étoile-lune et une paire étoile-étoile, comme montré sur l'image.



Une lune ne peut être présente que comme la moitié d'une paire étoile-lune dans le bracelet. Chaque lune est donc à côté d'au moins une étoile. Ce n'est donc pas possible d'obtenir trois lunes côte à côte comme dans le bracelet C. C'est aussi impossible d'avoir cinq étoiles ou plus côte à côte.



C'est de l'informatique !

Lorsque des programmeurs et programmeuses donnent des instructions à un ordinateur, c'est important qu'ils spécifient exactement ce que l'ordinateur doit faire. Sinon, on pourrait obtenir un résultat non désiré. Par exemple, Marie a oublié, dans ses instructions, de dire exactement comment les trois paires étoile-lune devaient être reliées. Dans le bracelet qu'elle désire, une lune est toujours entourée d'étoiles. Il manquait donc quelque chose, même si les instructions avaient l'air très précises. S'il

existait un ordinateur qui contrôle une machine fabriquant des bracelets, les instructions de Marie ne seraient pas assez précises. Heureusement, en général, les vrais ordinateurs s'arrêteraient simplement en annonçant : « Je ne sais pas ce que tu veux dire parce que les instructions ne sont pas assez claires. »

En informatique, il y a beaucoup de mécanismes permettant de décrire les choses très précisément. L'un de ces mécanismes est appelé *grammaire*. Une grammaire contient des *règles* qui décrivent précisément comme certains *mots* (une suite de lettres) peuvent être générés. On pourrait par exemple exprimer les instructions de Marie à l'aide d'une grammaire comme ceci :

$$\begin{aligned} &\mathrm{B} \to \mathrm{CEE} && (1)\\ &\mathrm{C} \to \mathrm{PPP} && (2)\\ &\mathrm{P} \to \mathrm{EL} && (3) \end{aligned}$$

Ici, B représente le bracelet, C la chaîne, P la paire, E l'étoile et L la lune. On commence avec B et peut générer de nouveau mots en appliquant les trois règles de substitutions aussi souvent que nécessaire. On fait cela jusqu'à ce que le mot ne contiennent plus que les symboles E et L. Par exemple :

$$\begin{aligned} &\mathrm{B} \Rightarrow \mathrm{CEE} && \text{à l'aide de la règle (1)}\\ &\mathrm{CEE} \Rightarrow \mathrm{PPPEE} && \text{à l'aide de la règle (2)}\\ &\mathrm{PPPEE} \Rightarrow \mathrm{ELPPEE} \Rightarrow \mathrm{ELELPEE} \Rightarrow \mathrm{ELELELEE} && \text{à l'aide de la règle (3)} \end{aligned}$$

On peut se demander si la grammaire ci-dessus correspond exactement aux instructions de Marie.

En informatique, il ne s'agit pas que de programmer. Souvent, il s'agit de décrire des objets. Beaucoup de règles de production (comme une grammaire ou les instructions de Marie) peuvent décrire une classe d'objets (certains mots ou les bracelets possibles). Dans cette classe se trouvent exactement les objects que l'on peut générer à l'aide des règles.

Mots clés et sites web

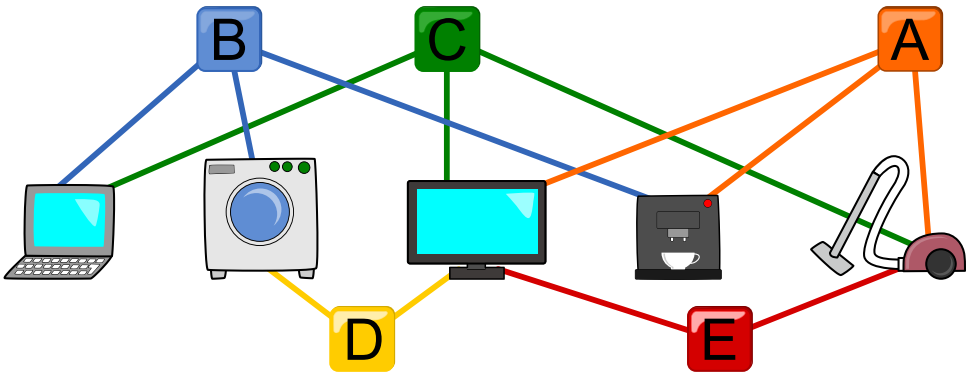
- Grammaire formelle : `https://fr.wikipedia.org/wiki/Grammaire_formelle`

18. Appareils ménagers

Dans la maison de Bruno le castor, il y a cinq appareils électriques (un ordinateur, un lave-linge, une télévision, une machine à café et un aspirateur) et cinq interrupteurs (A, B, C, D et E) pour allumer et éteindre des appareils. Le raccordement électrique est très inhabituel. Chaque interrupteur est connecté à plusieurs appareils, comme montré sur l'image en dessous. Chaque fois que l'on appuie sur un interrupteur, il change l'état de tous les appareils connectés : les appareils éteints s'allument et les appareils allumés s'éteignent.

Au départ, tous les appareils sont éteints. Si l'on appuie par exemple sur les interrupteurs A, C et E, l'aspirateur est allumé, car le premier bouton l'allume, le deuxième l'éteint et le troisième le rallume.

Sur quels interrupteurs Bruno doit-il appuyer pour que seules la télévision et la machine à café soit allumées ?



Solution

Lorsque l'on appuie, dans n'importe quel ordre, sur les interrupteurs B, C, D et E, seules la télévision et la machine à café sont allumées.

Nous pouvons aussi déterminer de manière systématique comment allumer et éteindre chaque appareil indépendamment des autres. Commençons avec deux combinaisons simples :

- A + E (appuyer sur A et B) contrôle la machine à café seulement.
- C + E (appuyer sur C et E) contrôle l'ordinateur seulement.

On observe ensuite que le lave-linge peut être contrôlé indépendamment des autres en appuyant d'abord sur B avant de remettre l'ordinateur et la machine à café dans le même état qu'avant, soit en appuyant sur A + E et sur C + E. Le lave-linge peut donc être contrôlé indépendamment du reste en appuyant sur B + A + E + C + E. L'interrupteur E est ici utilisé deux fois. Appuyer deux fois sur un interrupteur revient à ne pas l'utiliser du tout, on peut donc aussi contrôler le lave-linge indépendamment avec B + A + C. En utilisant cette méthode, on obtient la liste de combinaisons suivante pour contrôler les appareils séparément :

- Ordinateur : C + E
- Machine à café : A + E
- Lave-linge : A + B + C
- Télévision : A + B + C + D
- Aspirateur : A + B + C + D + E

Pour allumer seulement la télévision et la machine à café, nous devons donc appuyer sur A + B + C + D + A + E ce que l'on peut simplifier en B + C + D + E étant donné que les deux A s'annulent.

C'est de l'informatique !

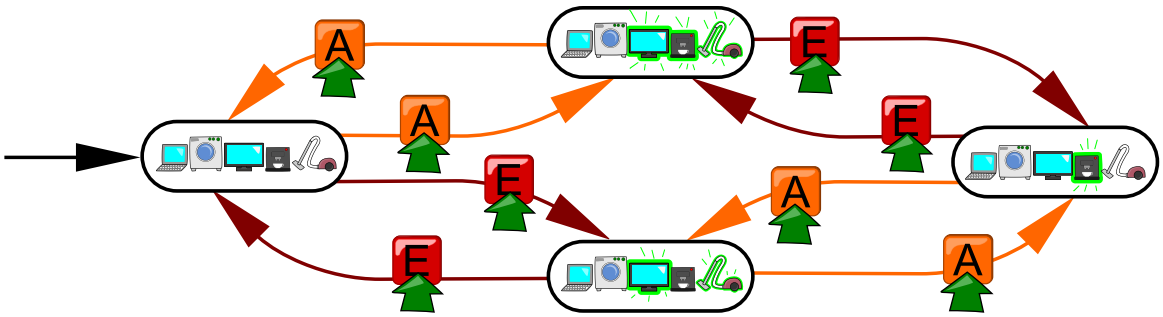
Le sytème composé des appareils et des interrupteurs peut être modélisé à l'aide d'un *automate fini* de la façon suivante.

Le système des cinq appareils a beaucoup d'*états* différents. Un de ces états est par exemple le cas où seule la télévision est allumée. Un autre état est le cas où tous les appareils sont éteints (comme tous les appareils sont éteints au départ, on appelle cet état l'*état initial*). Un autre état est le cas où seules la télévision et la machine à café sont allumées (c'est l'*état final* dans notre exercice, car c'est ce que nous voulons obtenir).

En appuyant sur un interrupteur, on fait passer le système d'un état à un autre. Par exemple, lorsque le système est à l'état initial et que l'on appuie sur l'interrupteur E, il passe à l'état auquel seuls la télévision et l'aspirateur sont allumés. Un tel passage d'un état à un autre s'appelle une *transition*.

Si l'on représente les états du système avec des cercles et les transitions avec des flèches, on obtient une image comme celle ci-dessous (pour des raisons de place, seuls quatre états et les transitions

entre ceux-ci sont représentés). L'état initial est marqué par une flèche spéciale. En informatique, ceci s'appelle un automate fini (un automate fini est simplement un graphe spécifique dans lequel les nœuds sont les états et les arêtes sont les transitions). On peut facilement voir sur l'image dans quel état l'on arrive lorsque l'on appuie sur différents interrupteurs.



Dans cet exercice, il s'agit de déterminer comment passer de l'état initial (tous les appareils ont éteints) à l'état final (seules la télévision et la machine à café sont allumées). On veut donc trouver un chemin allant de l'état initial à l'état final. Le recherche de chemin dans des graphes est une tâche fondamentale de l'informatique.

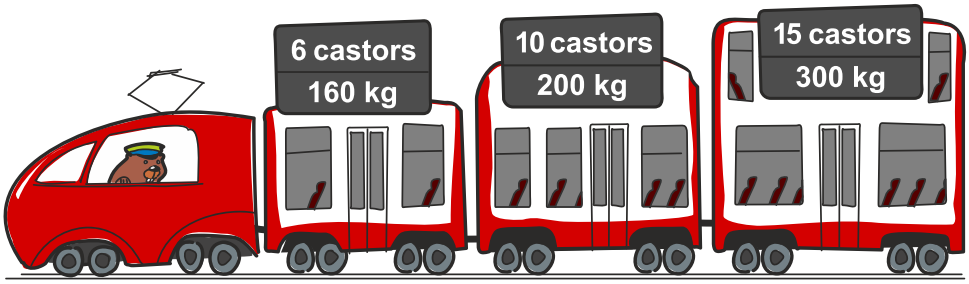
Mots clés et sites web

- Automate fini : `https://fr.wikipedia.org/wiki/Automate_fini`

19. Excursion de groupe

Huit familles de castors veulent prendre le « Glacier Express ». La table suivante liste les familles, leur nombre de membres et le poids de leurs bagages :

Nom de famille	Nombre de membres	Poids des bagages en kg
Ammann	3	50
Bernasconi	4	80
Camenzind	5	110
Donetta	4	80
Emery	2	40
Favre	3	70
Gerber	6	130
Huber	5	100



L'image montre combien de castors et quelle quantité de bagages peuvent être transportés au maximum dans chaque wagon. De plus, les familles et leurs bagages doivent rester ensemble dans le même wagon et ne peuvent pas se séparer.

Combien de familles de castors le « Glacier Express » peut-il transporter au maximum ?

A) 1 famille de castors
B) 2 familles de castors
C) 3 familles de castors
D) 4 familles de castors
E) 5 familles de castors
F) 6 familles de castors
G) 7 familles de castors
H) 8 familles de castors

Solution

Le « Glacier Express » peut transporter au maximum 7 familles de castors. Une des répartitions possibles est :

	Nom de famille	Nombre de membres	Bagages en kg
	Gerber	6	130
	Total :	**6**	**130**
	Ammann	3	50
	Camenzind	5	110
	Emery	2	40
	Total :	**10**	**200**
	Bernasconi	4	80
	Donetta	4	80
	Huber	5	100
	Total :	**13**	**260**

Les 8 familles de castors comptent en tout 32 personnes, alors que le train n'a que 31 places à disposition. C'est donc exclu que toutes les familles prennent le « Glacier Express ».

C'est de l'informatique !

L'informatique s'occupe souvent de problèmes d'optimisation, dans lesquels des ressources limitées – comme ici les places disponibles et le poids maximal – doivent être utilisées le mieux possible. En réalité, aucun passager ne devrait être laissé à la traîne, mais la compagnie de transport pourrait par exemple décider de transporter les voyageurs seuls en taxi plutôt que d'utiliser un train complet qui roulerait presque vide.

Ce genre de problème est appelé *problème de découpe et de conditionnement*. Le célèbre problème du sac à dos appartient aussi à cette catégorie.

Parfois, de tels problèmes peuvent être simplifiés de manière à pouvoir être résolus à l'aide de la *programmation dynamique*, c'est-à-dire en commençant par chercher des solutions partielles que l'on peut ensuite combiner en une solution complète. Dans beaucoup de cas, ces problèmes sont cependant ce que l'on appelle des problèmes *NP-complets*, ce qui veut dire que l'on ne connaît actuellement pas de meilleure méthode de résolution que le tâtonnement. C'est aussi ainsi que la plupart d'entre vous avez résolu cet exercice.

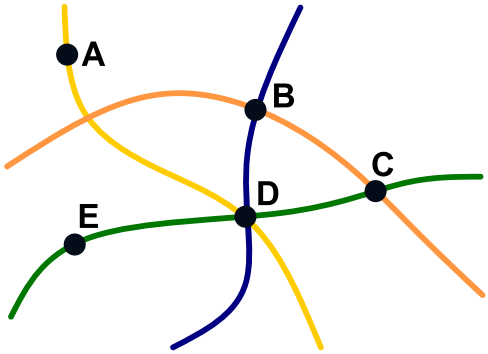
Mots clés et sites web

- Problème du sac à dos : `https://fr.wikipedia.org/wiki/Problème_du_sac_à_dos`

- Programmation dynamique : `https://fr.wikipedia.org/wiki/Programmation_dynamique`
- Problème de découpe et conditionnement
- NP-complet : `https://fr.wikipedia.org/wiki/Problème_NP-complet`

20. Réseau ferroviaire

Voici une carte de cinq villes et quatre lignes de train. Les points noirs représentent les villes, les lignes colorées les lignes de train.

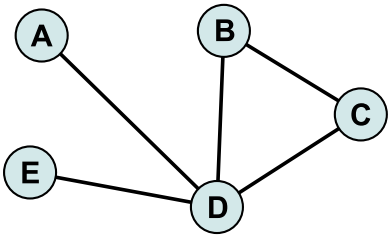


Un diagramme doit représenter cette carte de manière à ce que :

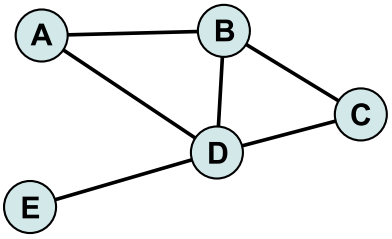
- les villes soient représentées par des cercles ;
- deux villes soient reliées d'un trait si elles sont situées sur la même ligne de train.

Quel diagramme représente la carte correctement ?

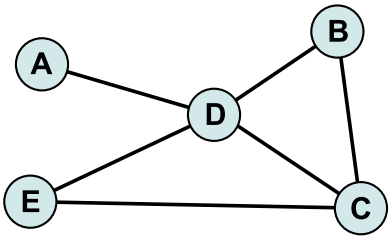
A)



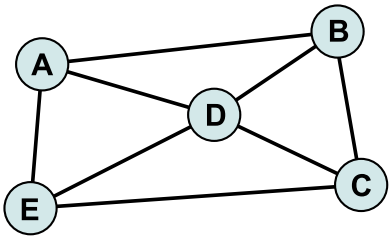
B)



C)

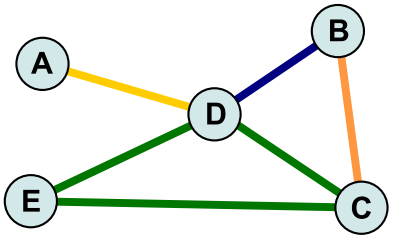


D)



Solution

La bonne réponse est C).



En observant bien la carte, on voit que :

- les villes A et D sont les deux sur la ligne de train jaune ;
- les villes B et C sont les deux sur la ligne de train orange ;
- les villes B et D sont les deux sur la ligne de train bleue ; et
- les villes C, D et E sont les trois sur la ligne de train verte.

Toutes les autres réponses sont fausses :

- Dans la réponse A, il manque le trait entre les villes C et E, qui doit être présent en raison de la ligne de train verte.
- Dans la réponse B, il y a le même problème que dans la réponse A, et il y a en plus un trait qui relie les villes A et B alors qu'elles ne sont pas sur la même ligne.
- Dans la réponse D, il y a deux traits entre les villes A et B ainsi qu'A et E, alors que la ville A n'est ni sur la même ligne que la ville B, ni que la ville E.

Il faut porter attention aux deux points suivants :

- Même s'il l'on peut arriver de la ville A à la ville B en prenant plusieurs lignes de train, les deux villes ne sont pas sur la même ligne.
- Même si une autre ville se trouve entre la ville C et la ville E sur la ligne verte, les deux villes sont sur la même ligne de train.

C'est de l'informatique !

Il y a plusieurs manières possibles de représenter la réalité. La carte plus haut avec les lignes colorées, par exemple, est déjà une représentation relativement abstraite de la situation réelle. Une forme de représentation très importante est le *graphe* – un diagramme qui comporte de nœuds (les cercles) et des arêtes (les traits entre les cercles). Cette forme de représentation est utilisée dans la solution.

Beaucoup de choses sont simplifiées par l'utilisation d'une forme de représentation adaptée. C'est pour cela qu'il est important de connaître beaucoup de formes de représentation pour programmer. En général, on ne peut pas dire qu'une forme de représentation soit mieux qu'une autre. Suivant l'usage prévu, une forme de représentation sera plus adaptée qu'une autre. Le graphe de la solution, par exemple, est pratique car on peut directement y voir que l'on peut aller de C à E sans changer

de train. On perd par contre l'information présente sur la carte que l'on passe par la ville D en allant de C à E avec cette ligne de train.

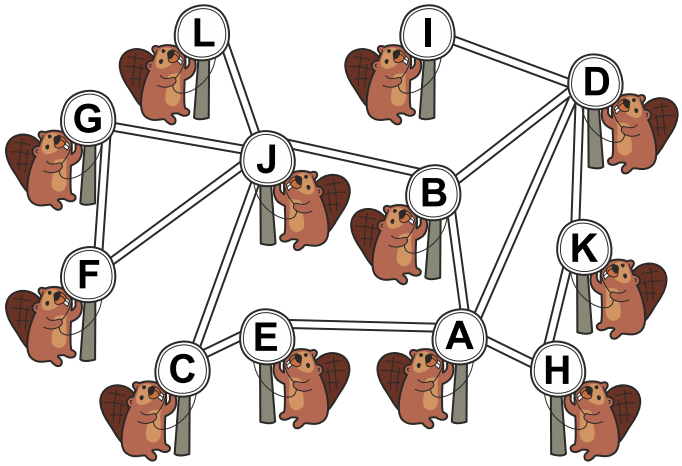
Mots clés et sites web

- Graphe : `https://fr.wikipedia.org/wiki/Graphe_(mathématiques_discrètes)`
- Théorie des graphes : `https://fr.wikipedia.org/wiki/Théorie_des_graphes`

21. Réseau de communication

Les castors aiment bien diffuser des informations entre eux. Pour cela, ils utilisent le réseau de communication ci-dessous. Lorsqu'un castor reçoit une nouvelle information, il l'envoie à tous les castors avec qui il partage un canal de communication direct (une ligne blanche). L'envoi d'information se passe en étapes. Une étape se passe entre l'envoi et la réception d'une information.

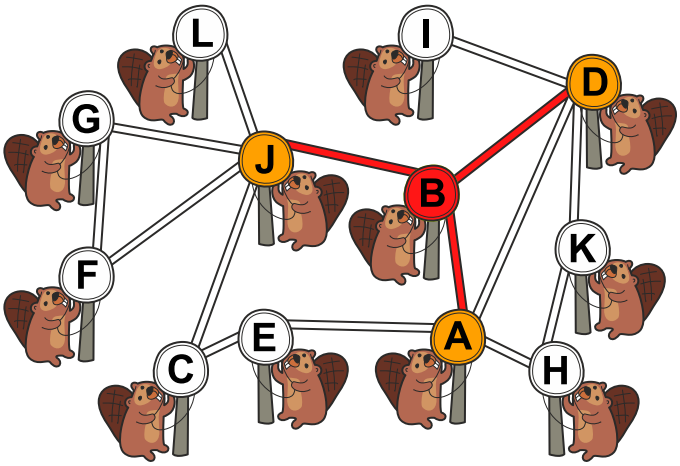
À partir de quel castor une nouvelle atteint-elle tous les autres castors le plus vite possible, c'est-à-dire en le moins d'étapes possible ?



Solution

La bonne réponse est le castor B. Il peut diffuser une information à tous les autres castors en deux tours.

Lors du premier tour, le castor B envoie l'information à ses voisins, donc aux castors A, D et J, avec qui il partage un canal de communication direct. L'image ci-dessous montre qui possède l'information après un tour.

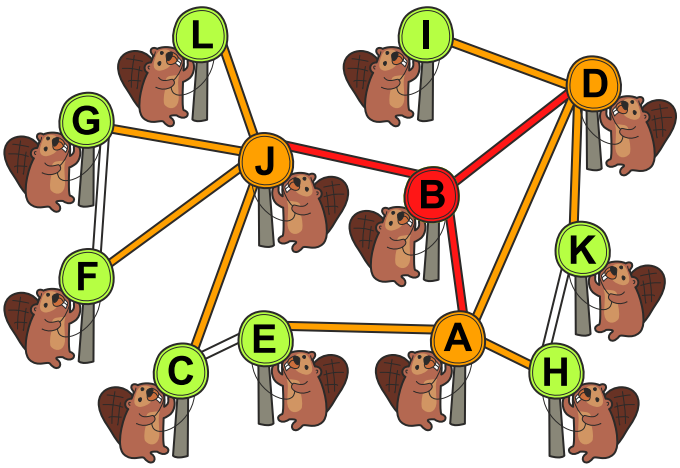


Lors du deuxième tour, les castors A, D et J envoient l'information à leurs voisins respectifs :

- le castor A envoie l'information aux castors E et H ;
- le castor D envoie l'information aux castors I et K ;
- le castor J envoie l'information aux castors C, F, G et L.

De plus, le castor B reçoit l'information trois fois en retour, car c'est aussi un voisin des castors A, D et J, mais comme ce n'est pas une nouvelle information, le castor B ne ne va pas la diffuser lors des tours suivants. Les castors A et D vont également s'envoyer l'information mutuellement par leur canal de communication direct, mais ne vont pas continuer à la diffuser non plus, étant donné qu'elle n'est pas nouvelle.

L'image ci-dessous montre la situation à la fin du deuxième tour.



L'information a donc atteint tous les castors en seulement deux tours.

Une diffusion plus rapide n'est pas possible, car pour cela, l'un des castors devrait être relié à tous les autres castors par une ligne blanche afin d'envoyer l'information à tout le monde en un tour.

Le castor B est le seul castor pouvant diffuser une information à tous les castors en deux tours : les castors C, E, F, G, H et J ne pourraient pas atteindre le castor I en deux tours, et les castors A, D, E, H, I et K ne pourraient pas atteindre le castor L en deux tours.

C'est de l'informatique !

On peut utiliser un *graphe* pour décrire le réseau de communication des castors. Chaque castor ce trouve à ce que l'on appelle un *nœud*, qui est dans ce cas désigné par une lettre. Les lignes blanches s'appellent des *arêtes* et relient deux nœuds. Les informations dont diffusées dans le réseau de communications en tours *synchronisés*, tous les castors envoient donc l'information en même temps. Lors d'un tour, chaque castor envoie les nouvelles informations à chacun de ses ses voisins. Ce que les castors font ici est appelé un *broadcast* sur un *réseau informatique* par les informaticiens et informaticiennes. Dans l'exercice ci-dessus, nous avons analysé à quelle vitesse un tel broadcast peut avoir lieu, c'est-à-dire à quelle vitesse une information peut atteindre tous les participants.

Un exercice encore plus exigeant consisterait à former un réseau de manière à ce qu'un broadcast rapide soit possible à partir de tous les nœuds sans que le nombre de connections ne soit trop grand.

Le nœud du castor B s'appelle le centre du graphe. D'une manière abstraite, le centre est un nœud qui minimise la distance entre lui-même et nœud le plus éloigné de lui. Il n'y a donc pas d'autre nœud ayant une plus petite distance à tous les autres nœuds. Dans l'exercice précédent, il n'y a qu'un centre. Dans certains graphes, il peut aussi y avoir plusieurs nœuds qui minimisent chacun la distance au nœud le plus éloigné ; un graphe peut donc avoir plusieurs centres.

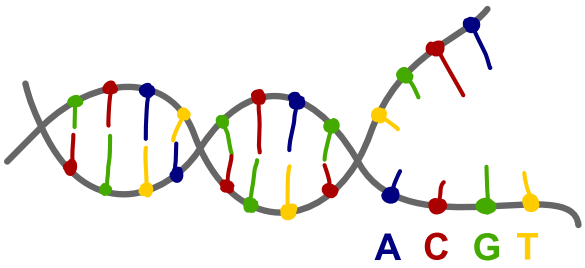
Ce n'est pas toujours aussi simple de trouver le centre d'un graphe que dans cet exercice. C'est possible par exemple que la transmission entre certains nœuds reliés directement dure plusieurs tours. Les graphes peuvent aussi être beaucoup plus grands et complexes. Pour de tels graphes, on peut par exemple utiliser l'*algorithme de Floyd-Warshall* pour trouver un centre de manière efficiente.

Mots clés et sites web

- Graphe : `https://fr.wikipedia.org/wiki/Graphe_(mathématiques_discrètes)`, `https://fr.wikipedia.org/wiki/Chaîne_(théorie_des_graphes)`
- Centre d'un graphe : `https://fr.wikipedia.org/wiki/Centralité#Centralité_de_proximité`
- Algorithme de Floyd-Warshall : `https://fr.wikipedia.org/wiki/Algorithme_de_Floyd-Warshall`

22. Séquence ADN

Notre patrimoine génétique est enregistré sous forme de séquences d'ADN. Une séquence d'ADN est essentiellement une suite de bases dont quatre formes existent : A, C, D et T.



Nous considérons les trois sortes de mutations suivantes :

Mutation	**Description**	**Exemple**
Substitution	Une base est remplacée par une autre.	AT**G**GT → AT**A**GT
Délétion	Une base est perdue sans être remplacée.	ATG**G**T → ATGT
Insertion	Une base est ajoutée dans une séquence.	ATGGT → A**C**TGGT

*Il y a exactement une des séquences suivantes qui ne peut **pas** être générée par trois mutations de la séquence GTATCG. Laquelle est-ce ?*

A) GCAATG
B) ATTATCCG
C) GAATGC
D) GGTAAAC

Solution

La bonne réponse est D) GGTAAAC.

La meilleure méthode pour trouver la réponse est de procéder par élimination, étant donné que les trois autres séquences peuvent résulter de trois mutations.

Réponse A : GTATCG ⇒ G**C**ATCG ⇒ GCA**A**CG ⇒ GCAATG
Réponse B : GTATCG ⇒ **A**TATCG ⇒ AT**T**ATCG ⇒ ATTAT**C**CG
Réponse C : GTATCG ⇒ G**A**ATCG ⇒ GAAT**G**G ⇒ GAATG**C**

En comparaison, il faut quatre mutations pour obtenir la séquence de la réponse D), par exemple celles-ci :

GTATCG ⇒ G**G**TATCG ⇒ GGT**A**ATCG ⇒ GGTAA**A**CG ⇒ GGTAAAC

Ce n'est pas facile de prouver que moins de quatre mutations ne sont pas suffisantes.

C'est de l'informatique !

La représentation d'information à l'aide de *chaînes de caractères* (des séquences de lettres) et leur utilisation est une tâche centrale de l'informatique.

Une question importante est de déterminer quel est le degré de différence entre deux chaînes de caractères. Il existe plusieurs méthodes pour mesurer le degré de différence entre deux chaînes de caractères. Une méthode fréquemment utilisée est la *distance de Levenshtein*, qui est définie à base des trois sortes de mutations décrites plus haut : la distance de Levenshtein entre deux chaînes de caractères est le nombre minimal de mutations permettant de transformer une chaîne en l'autre.

L'algorithme courant utilisé pour calculer la distance de Levenshtein se base sur la *programmation dynamique* : la distance de Levenshtein entre des préfixes toujours plus longs des deux chaînes de caractères sont inscrites dans un tableau jusqu'à ce que les préfixes correspondent aux mots entiers et que l'on puisse lire les résultats dans la table.

Lorsque l'exactitude de l'algorithme est prouvée, on peut calculer que la distance entre la séquence d'origine et celle de la réponse D) est exactement 4. On a ainsi prouvé que moins de quatre mutations ne suffisent pas.

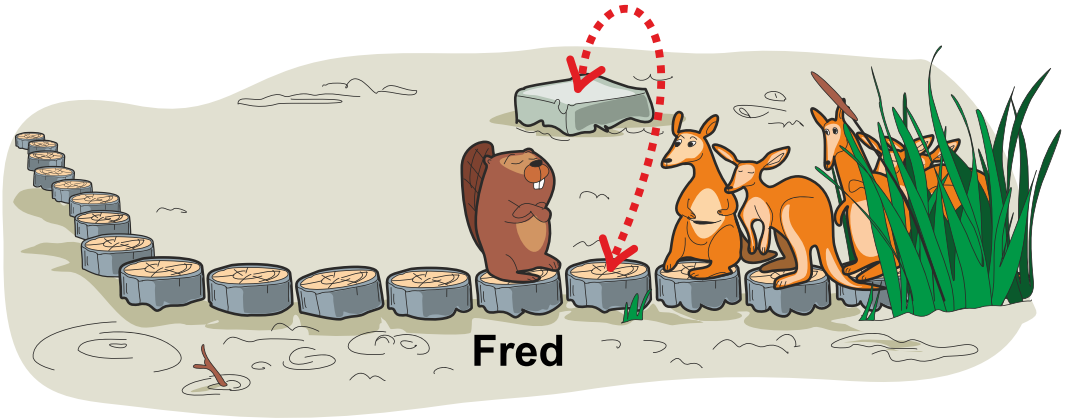
Mots clés et sites web

- Distance de Levenshtein : `https://fr.wikipedia.org/wiki/Distance_de_Levenshtein`
- `https://fr.wikipedia.org/wiki/Chaîne_de_caractères`

23. Fred le têtu

Des kangourous se déplacent en direction du castor Fred sur un chemin de rondins. Le chemin est assez étroit, ce qui fait que Fred et les kangourous ne peuvent pas s'y croiser. Il y a un certain rondin depuis lequel les kangourous peuvent sauter sur une pierre pour libérer le chemin avant de retourner le même rondin comme montré sur l'image. Un seul animal peut se tenir sur chaque rondin et sur la pierre.

Fred aimerait avancer. Il est assez têtu et n'est prêt à reculer d'un rondin que 10 fois. Par contre, il avance d'un rondin aussi souvent que nécessaire.



Quel est le nombre maximal de kangourous que Fred peut laisser passer ?

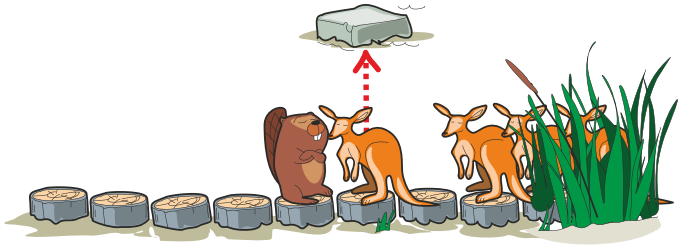
A) Plus de 10 kangourous
B) Exactement 10 kangourous
C) Exactement 6 kangourous
D) Exactement 4 kangourous
E) Moins de 4 kangourous
F) On ne peut pas savoir exactement

Solution

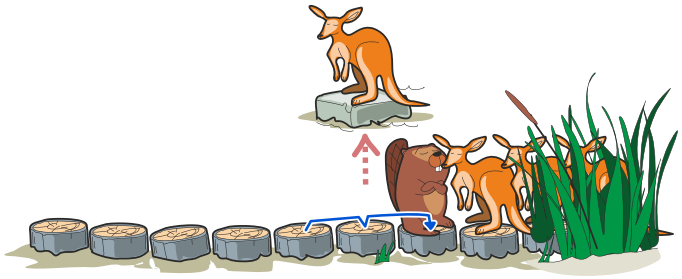
Fred peut laisser passer au maximum 6 kangourous.

Un kangourou dépasse Fred de la manière suivante :

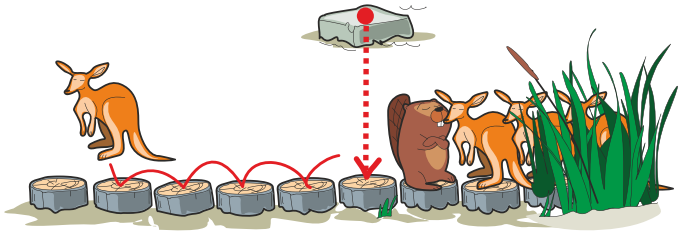
Le kangourou saute sur la pierre.



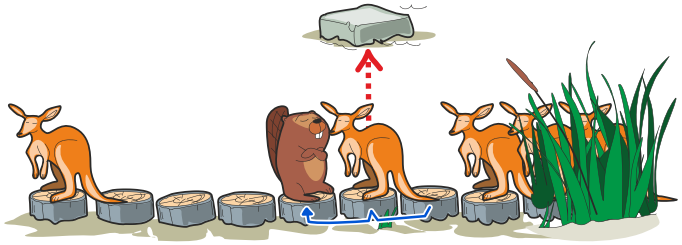
Fred avance de deux rondins.



Le kangourou revient sur le chemin en sautant et continue.



Si Fred recule à présent de deux rondins, il est à nouveau à sa position de départ et peut répéter le procédé pour laisser passer un autre kangourou.



Comme il ne recule que de dix rondins au maximum, il peut faire cela cinq fois, et donc laisser passer six kangourous en comptant le premier.

C'est de l'informatique !

En informatique, les tâches sont résolues entre autres en utilisant des algorithmes : des suites d'*instructions* qui sont effectuées pas à pas – exactement comme « Fred avance d'un rondin » ou « un kangourou saute sur la pierre ».

Dans ce qu'on appelle une *boucle*, une suite d'instructions peut être répétée. De cette manière, des tâches répétitives peuvent êtres exécutées plusieurs fois de manière fiable. Pour cela, c'est souvent un avantage de commencer chaque passage de la boucle par la même situation – ce qu'on appelle un

invariant de boucle. Dans notre cas, Fred doit toujours retourner à sa position de départ afin que le même procédé fonctionne à nouveau pour le kangourou suivant.

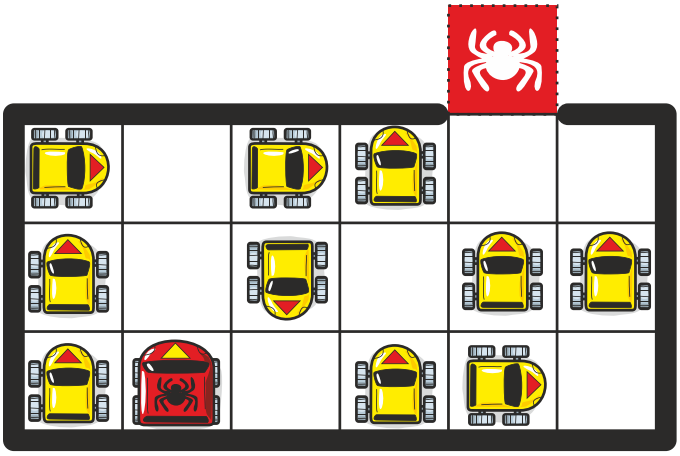
Mots clés et sites web

- Algorithme : `https://fr.wikipedia.org/wiki/Algorithme`
- `https://fr.wikipedia.org/wiki/Programmation_structurée`
- Boucle : `https://fr.wikipedia.org/wiki/Structure_de_contrôle#Boucles`
- Invariant : `https://fr.wikipedia.org/wiki/Invariant_de_boucle`

24. Heure de pointe

Onze voitures se parquent sur une place entourée d'un mur avec une sortie. Chaque voiture a les possibilités suivantes pour chacun de ses déplacements :

- Une case vers l'avant
- Une case vers l'arrière
- Un quart de tour vers la gauche ou la droite sur la case actuelle



Une voiture peut effectuer plusieurs déplacements. Une seule voiture peut se trouver sur chaque case.

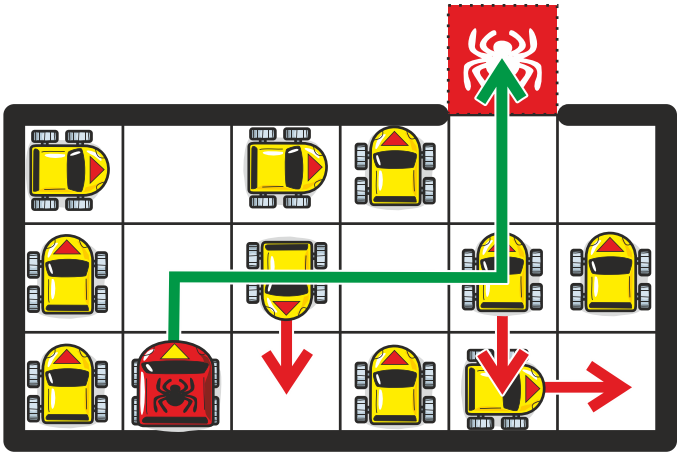
Combien de déplacements de voitures sont nécessaires pour amener la voiture rouge marquée d'une araignée sur la case rouge araignée ?

A) 9 déplacements
B) 11 déplacements
C) 13 déplacements
D) 15 déplacements

Solution

La bonne réponse est B) 11 déplacements.

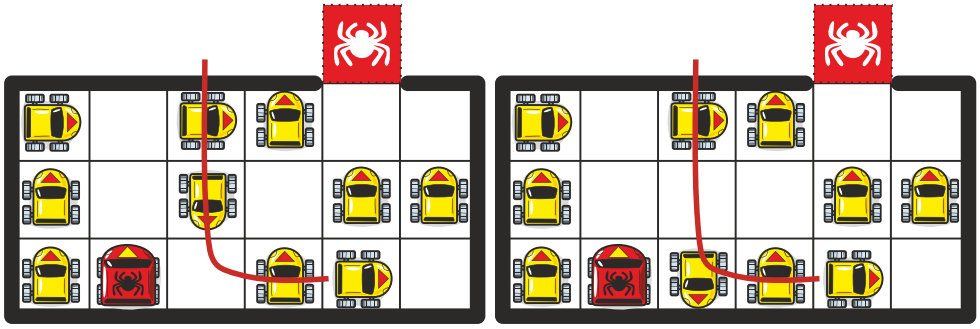
L'image montre les 11 déplacements nécessaires pour amener la voiture rouge à la case araignée :



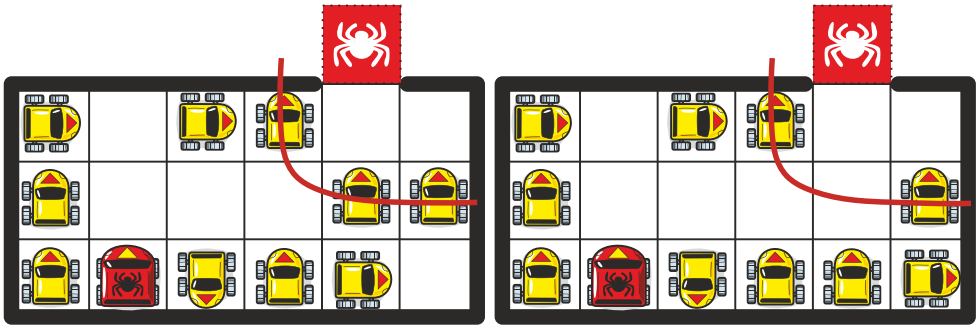
Il faut encore montrer que 11 est le nombre de déplacements minimal.

Pour cela, on commence par supposer que la voiture rouge est la seule voiture sur la place. Pour arriver à la case araignée à l'extérieur de la place, la voiture rouge doit se déplacer trois fois vers le haut et trois fois vers la droite, et se tourner deux fois d'un quart de tour. Cela peut se faire de différentes manières, mais nécessite toujours au minimum $3 + 3 + 2 = 8$ déplacements. Cependant, la voiture rouge n'est pas la seule voiture présente sur la place, et d'autres déplacements sont nécessaires pour libérer le chemin.

D'abord, on doit trouver un chemin à travers la barricade en forme de L montrée sur l'image suivante. Ceci peut être fait en un déplacement comme cela :



Ensuite, on doit trouver un chemin à travers une deuxième barricade en forme de L. Cette barricade ne peut pas être ouverte en un déplacement, mais deux déplacements suffisent, comme montré ci-dessous.



Le nombre minimal est donc $8 + 1 + 2 = 11$ déplacements.

C'est de l'informatique !

C'est souvent difficile de prouver qu'un certaine solution est optimale. Souvent, on ne détermine qu'il n'y a pas de meilleure solution seulement en comparant toutes les solutions possibles. On appelle cette méthode *recherche par force brute* ou *recherche exhaustive*, car on épuise toutes les possibilités. Cette méthode n'est cependant pas praticable à la main, mais est souvent une stratégie facile à réaliser à l'ordinateur.

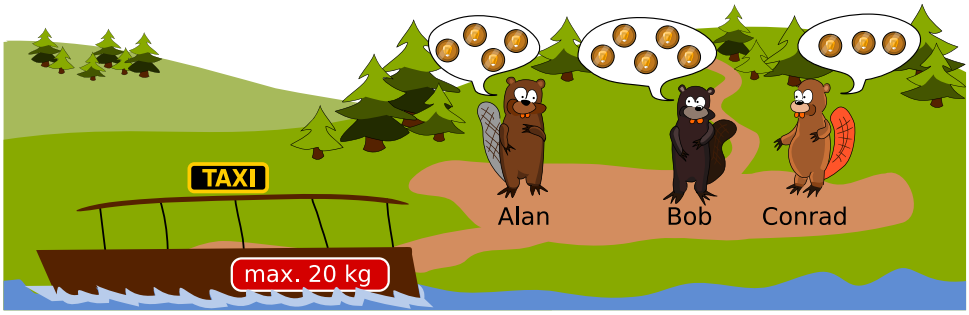
Parfois, il y a tellement de solutions possibles que même un ordinateur serait surchargé s'il devait toutes les passer en revue. Dans ce cas, il faut chercher une stratégie adaptée. Souvent, on peut utiliser des *algorithmes gloutons* ou le principe de *séparation et évaluation* (« *branch and bound* » en anglais).

Cet exercice est une variante du jeu *Rush Hour* (« heure de pointe » en anglais). Le jeu vidéo classique *Sokoban* est aussi similaire à cet exercice.

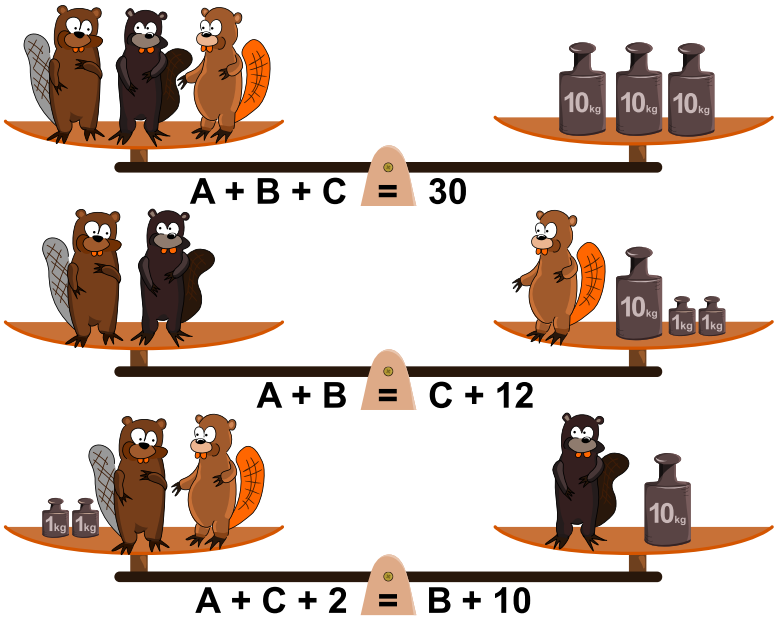
Mots clés et sites web

- Force brute : `https://fr.wikipedia.org/wiki/Recherche_exhaustive`
- Séparation et évaluation : `https://fr.wikipedia.org/wiki/Séparation_et_évaluation`
- Algorithme glouton : `https://fr.wikipedia.org/wiki/Algorithme_glouton`
- Rush Hour : `https://fr.wikipedia.org/wiki/Rush_hour_(casse-tête)`

25. Bateau-taxi



Les trois castors Alan, Bob et Conrad veulent prendre un bateau-taxi. Il n'y a qu'un bateau-taxi. Alan est prêt à payer 4 francs castor (4×), Bob 5 francs castor (5×) et Conrad seulement 3 francs castor (3×). Le taxi peut transporter au maximum 20 kg. Le chauffeur de taxi fait donc les pesées suivantes :



Quel(s) castor(s) le chauffeur prend-il avec s'il veut gagner le plus d'argent possible ?

A) Seulement Bob
B) Alan et Bob
C) Bob et Conrad
D) Alan et Conrad
E) Tous les trois : Alan, Bob et Conrad

Solution

La bonne réponse est C) Bob et Conrad.

Pour pouvoir faire une liste de toutes les solutions possibles et les évaluer, nous devons d'abord savoir combien pèse chaque castor.

Nous savons que les trois castors ensemble pèsent 30 kg et que le chauffeur ne peut donc pas tous les prendre avec. Si nous ajoutons un copie de C(onrad) du côté gauche et du côté droit de la deuxième balance, cela donne à gauche $A + B + C = 30$ kg et à droite $C + C + 12$ kg. Donc, nous avons $2C = 18$ kg et $C = 9$ kg.

Si nous ajoutons un copie de B(ob) du côté gauche et du côté droit de la troisième balance, nous obtenons à gauche $A + B + C + 2$ kg $= 32$ kg et à droite $2B + 10$ kg. Cela donne $2B = 22$ kg et donc $B = 11$ kg.

Comme $A + B + C = 30$ kg, $A = 10$ kg.

Le chauffeur de taxi peut donc :

- Prendre Alan et Conrad avec et gagner $4 + 3 = 7$ francs castor.
- Prendre Bob et Conrad avec et gagner $5 + 3 = 8$ francs castor.
- Prendre Alan et Bob avec et gagner le plus avec 9 francs castors, mais comme les deux castors pèsent ensemble plus de 21 kg, le bateau-taxi est surchargé.

La bonne réponse est donc C).

Ce n'est cependant pas la seule possibilité de déterminer le poids des castors. On aurait aussi pu remplacer $A + B$ par $C + 12$ à gauche de la première balance. Ceci donne ensuite $2C + 12$ kg $= 30$ kg, et on peut en déduire que $C = 9$ kg.

De manière plus formelle, les trois pesées peuvent être écrite comme un système d'équations :

I. $A + B + C = 30$ kg
II. $A + B - C = 12$ kg
III. $A - B + C = 8$ kg

Ces équations peuvent ensuite être soustraites les une aux autres. La différence I. – II. donne l'équation :

$$2C = 18 \text{ kg} \rightarrow C = 9 \text{ kg}$$

La différence I. – III. donne :

$$2B = 22 \text{ kg} \rightarrow B = 11 \text{ kg}$$

On déduit ensuite de I. que $A = 10$ kg.

C'est de l'informatique !

Tous les problèmes d'optimisation discrète de la classe NP peuvent être représentés par des équations et des inéquations (on parle aussi de d'*optimisation linéaire*). Les équations et inéquations sont appelées *contraintes* et doivent être satisfaites par les valeurs des variables. On optimise ensuite la valeur d'une fonction des variables tout en respectant les contraintes. Dans cet exercice, on a trois variables booléennes, x_A, x_B et x_C. Si $x_A = 1$, le castor A prend le bateau, sinon $x_A = 0$. On optimise la fonction linéaire $4x_A + 5x_B + 3x_C$, pour laquelle on cherche la valeur maximale. La seule contrainte est :

$$Poids(A) \cdot x_A + Poids(B) \cdot x_B + Poids(C) \cdot x_C \leq 20.$$

On ne peut formuler l'exercice complètement que si l'on connaît le poids des castors. Cette instance de problème est un cas particulier du *problème du sac à dos*. On doit mettre la plus grande valeur possible dans le sac à dos sans dépasser la valeur maximale.

Il y a 80 ans, ce genre de questions était encore du ressort des mathématiciens, mais comme des ordinateurs de plus en plus performants ont été à disposition, des méthodes de résolution (par exemple la méthode de *séparation et évaluation* ou des *plans sécants*) avec lesquelles de tels problèmes peuvent être résolus ont été développées. Aujourd'hui, ces méthodes de résolution sont utilisées par exemple dans l'optimisation de la production, la logistique ou les réseaux de transport public.

Malgré tout, la résolution de problèmes d'optimisation est encore un exercice difficile en pratique qui demande une modélisation adroite et des algorithmes spécialement développés pour la structure et la taille du problème. Souvent, plusieurs méthodes de résolution sont combinées.

Mots clés et sites web

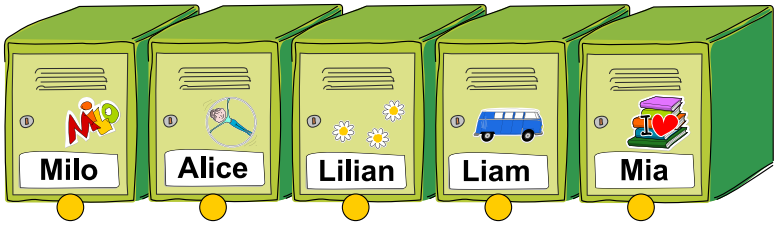
- Optimisation linéaire en nombre entiers : `https://fr.wikipedia.org/wiki/Optimisation_linéaire_en_nombres_entiers`
- Contrainte : `https://fr.wikipedia.org/wiki/Contrainte_(mathématiques)`
- Séparation et évaluation : `https://fr.wikipedia.org/wiki/Séparation_et_évaluation`
- Méthode des plans sécants : `https://fr.wikipedia.org/wiki/Méthode_des_plans_sécants`

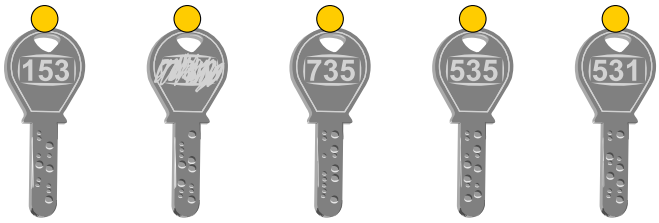
26. Casiers

Cinq enfants ont chacun un casier étiqueté à l'école. Un nombre à trois chiffres est gravé sur chacune des clés correspondantes. Malheureusement, le nombre sur l'une des clés est rayé.

Chaque nombre à trois chiffres représente les trois premières lettres d'un nom. Un chiffre représente toujours la même lettre, par exemple 8 pour « C » ou « c ».

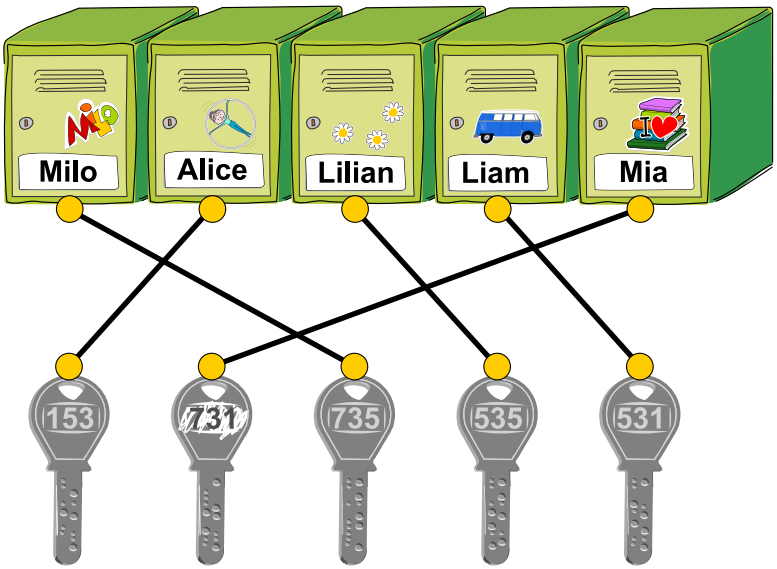
Relie les clés aux bons casiers. Pour cela, trace des lignes entre les points jaunes.





Solution

La bonne solution est illustrée ci-dessous :



Les quatre nombres connus sont 153, 735, 535 et 735. Les trois premières lettres des cinq noms sont MIL, ALI, LIL, LIA et MIA.

Il n'y a que LIL qui commence et se termine par la même lettre. Il doit donc y avoir un nombre à trois chiffres correspondant qui commence et se termine par le même chiffre, et il ne peut y avoir qu'un tel nombre. Le nombre 535 correspond à ce motif ; la clé 535 correspond donc à LIL. Cela veut dire que 5 représente L et 3 représente I. On peut maintenant voir que 531 doit correspondre à LIA, car il n'y a pas d'autre nom commençant par L. 1 représente donc la lettre A. De plus, 153 doit correspondre à ALI, car il n'y a pas d'autre nom avec un L en deuxième position. Il ne reste plus que le chiffre 7 et la lettre A qui ne sont pas attribués, ils doivent donc correspondre l'un à l'autre. On a ainsi l'attribution univoque 1 = A, 3 = I, 5 = L et 7 = M. 735 représente donc MIL et 531 LIA. On voit également que la clé avec le nombre rayé appartient à Mia et que ce nombre doit être 731.

Un méthode alternative pour trouver la bonne attribution est de compter la fréquence des chiffres et des lettres. Les lettres A et M apparaissent deux fois chacune dans MIL, ALI, LIL, LIA et MIA, et les lettres I et L cinq fois chacune. Malheureusement, cela ne suffit pas encore pour attribuer une lettre à chaque chiffre de manière univoque. On doit donc faire des observations supplémentaires telles que celles décrites plus haut.

C'est de l'informatique !

En informatique, les noms et les textes sont très souvent chiffrés à l'aide de nombres.

Dans la donnée de l'exercice, il est spécifié que l'on peut déduire les nombres sur les clés de manière univoque à partir des noms. Cela fonctionne car chaque lettre est chiffrée par exactement un chiffre et qu'il n'y a que peu de lettres. On parle d'un *chiffrement* (ou d'une *substitution*) *monoalphabétique*, car chaque lettre est toujours remplacée par le même symbole. Par contre, la donnée ne spécifie pas

quel chiffre correspond concrètement à quelle lettre. La solution montre cependant que l'on peut trouver la bonne attribution à l'aide de peu d'informations structurelles.

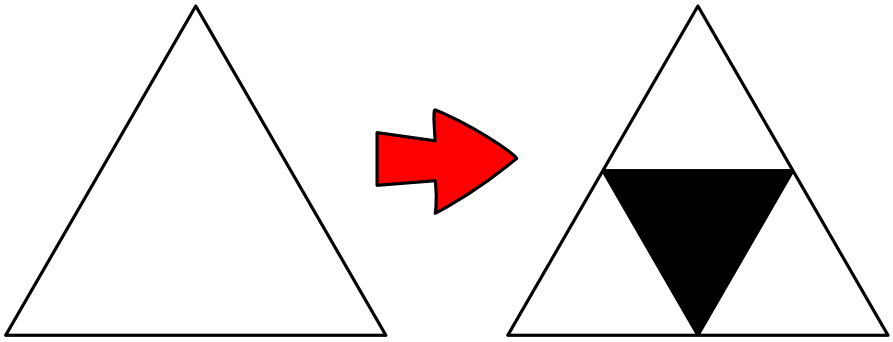
Si l'on n'utilise pas seulement dix chiffres pour le chiffrement, mais un symbole pour chaque lettre, on peut utiliser une telle substitution monoalphabétique comme un code secret simple. Malheureusement, la méthode de chiffrement par substitution monoalphabétique n'est pas très sûre, parce que l'on peut souvent déterminer l'attribution en utilisant quelques astuces. L'exercice en est un exemple. La *cryptographie* est un domaine important de l'informatique dans lequel des *chiffres* sont développés et analysés.

Mots clés et sites web

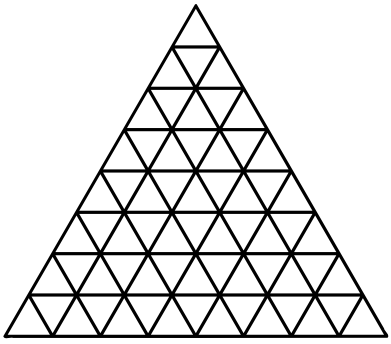
- Code, substitution monoalphabétique : `https://fr.wikipedia.org/wiki/Chiffrement_par_substitution#Substitution_monoalphabétique`
- Cryptographie : `https://fr.wikipedia.org/wiki/Cryptographie`

27. Triangle de Sierpiński

Pour obtenir un triangle de Sierpiński, on dessine d'abord un triangle équilatéral blanc, puis on procède étape par étape. À chaque étape, chaque triangle blanc existant est divisé en quatre triangles plus petits et celui du centre est coloré en noir, comme montré ci-dessous :

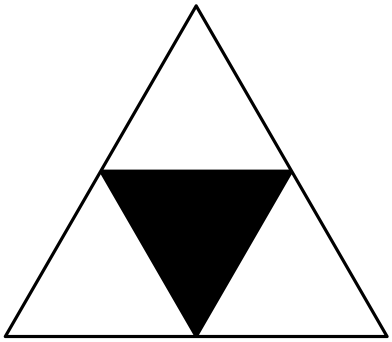


Dessine la figure obtenue après trois étapes. Pour cela, colorie les bons petits triangles en noir.

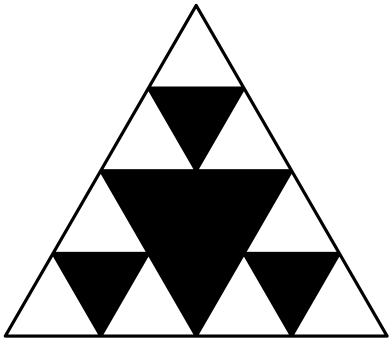


Solution

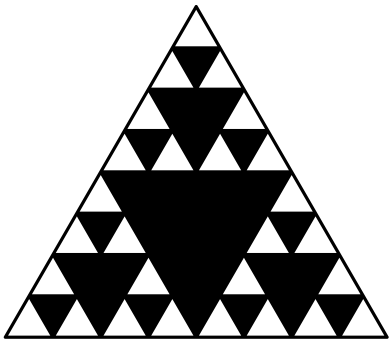
Après la première étape, le triangle central est noir et il y a trois triangles blancs :



Lors de la deuxième étape, les trois triangles blancs sont également divisés en quatre triangles plus petits, et chaque triangle central est coloré en noir. Il y a maintenant $3 \cdot 3 = 9$ petits triangles blancs :



Lors de la troisième et dernière étape, ces neufs triangles blancs sont à nouveau divisés en quatre triangles chacun, et chaque triangle central est coloré en noir. Il en résulte la figure suivante avec $3 \cdot 9 = 27$ triangles blancs :



C'est de l'informatique !

Le triangle de Sierpiński est une *fractale* qui a été décrite pour la première fois par le mathématicien polonais Waclaw Franciszek Sierpiński (1882–1969) en 1915. Les fractales sont des figures dans lesquelles apparaissent des éléments toujours plus petits, éléments qui sont semblables à la figure complète. C'est un travail très fastidieux de dessiner des images de fractales. Lorsque des ordinateurs

capable de faire les calculs nécessaires sont apparus au 20^e siècle, les fractales sont devenues très populaires. Le *flocon de Koch* et l'*ensemble de Mandelbrot* sont des fractales connues.

La construction du triangle de Sierpiński est récursive (du latin *re-currere* : se reproduire). Cela signifie que les règles de construction contiennent une instruction stipulant qu'il faut répéter l'application des règles. Dans cet exercice, cette instruction dit : « Divise le triangle blanc en quatre triangles plus petits, colore le triangle central en noir, puis répète cette instruction pour les triangles blancs résultants. » Une application de l'instruction s'appelle une *étape récursive*, et l'instruction demandant de réappliquer les règles s'appelle un *appel récursif* (dans l'exemple, il y a trois appels récursifs par étape récursive). Comme chaque appel récursif contient d'autres appels récursifs, on doit encore et toujours répéter l'étape récursive, ce qui peut durer indéfiniment. On peut éviter cela avec une condition de terminaison. Dans l'exemple, les appels récursifs s'arrêtent lorsque les triangles deviennent trop petits.

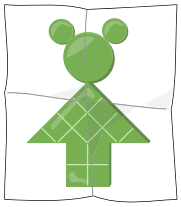
Le concept de la récursivité a un large domaine d'application en informatique, car beaucoup d'objects complexes – par exemple les fractales – peuvent être décrits de manière compacte grâce à la récursivité, et beaucoup de tâches complexes – par exemple les tours de Hanoï – peuvent être résolues à l'aide d'algorithmes récursifs très simples.

Mots clés et sites web

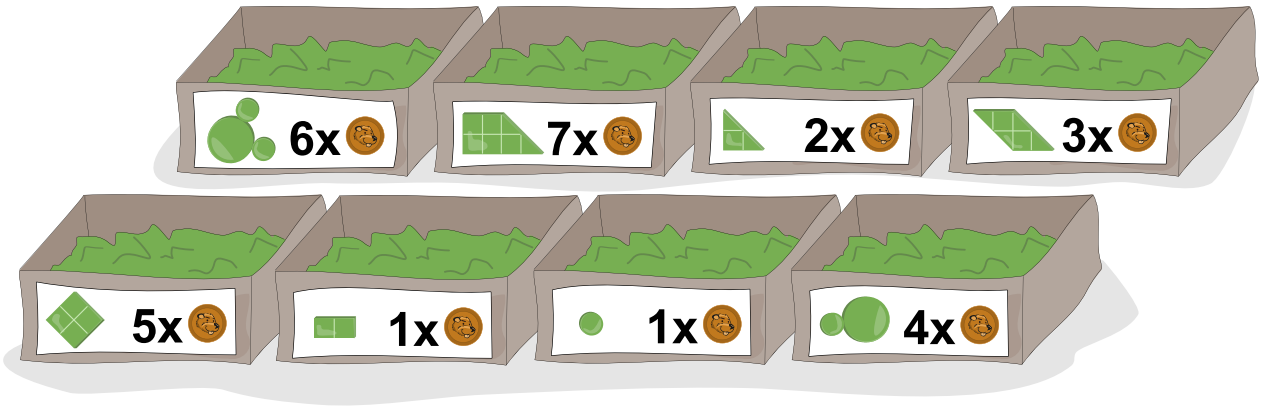
- Triangle de Sierpiński : `https://fr.wikipedia.org/wiki/Triangle_de_Sierpiński`
- Récursivité : `https://fr.wikipedia.org/wiki/Récursivité`
- Fractale : `https://fr.wikipedia.org/wiki/Fractale`
- `https://fr.wikipedia.org/wiki/Wacław_Sierpiński`
- `https://fr.wikipedia.org/wiki/Tours_de_Hanoï#Solution_récursive`
- `https://fr.wikipedia.org/wiki/Flocon_de_Koch`
- `https://fr.wikipedia.org/wiki/Ensemble_de_Mandelbrot`

28. Mosaïque

Giulia veut acheter des tesselles pour réaliser ce personnage en mosaïque :



Le magasin de jouets propose différentes tesselles en quantité au choix. Le prix par tesselle varie entre 1 et 7 pièces de monnaie.



Les tesselles peuvent être tournées comme désiré lors de l'assemblage, mais elles ne peuvent pas se chevaucher.

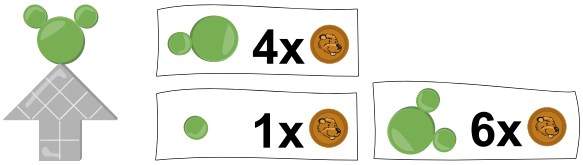
Combien de pièces de monnaie Giulia doit-elle dépenser si elle choisit l'option la moins chère ?

A) 13 pièces de monnaie
B) 14 pièces de monnaie
C) 16 pièces de monnaie
D) 20 pièces de monnaie

Solution

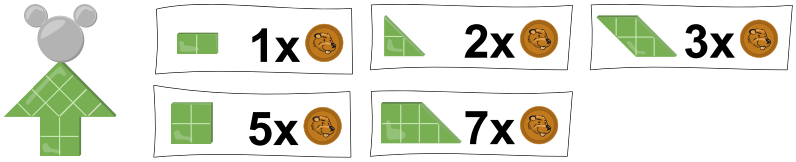
La bonne réponse est 13 pièces de monnaie.

L'une des méthodes pour résoudre cet exercice consiste à considérer séparément les différentes parties du personnage. Le plus simple est de commencer par la tête, qui ne peut être assemblée qu'avec des tesselles rondes :

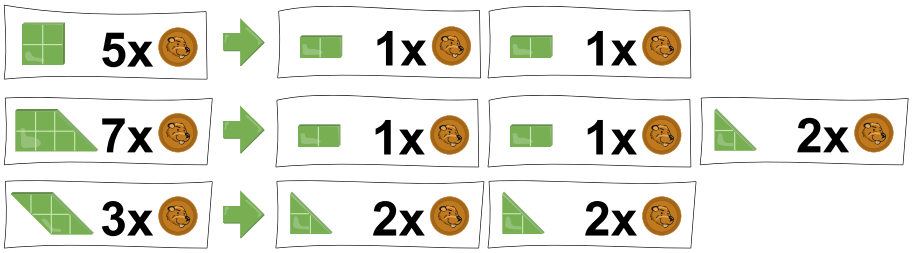


Il n'y a que deux possibilités pour assembler la tête : soit on utilise directement la tesselle adaptée pour 6 pièces de monnaie, soit on l'assemble à partir des deux autres tesselles ronde qui coûtent ensemble $4 + 1 = 5$ pièces de monnaie. Comme la deuxième option est moins chère, c'est celle que l'on utilise.

Le reste du personnage ne peut être assemblé qu'avec des tesselles anguleuses.



On pourrait maintenant essayer toutes les variantes possibles pour assembler le personnage et calculer le prix de chacune, mais ceci prend beaucoup de temps. Les observations suivantes concernant les tesselles anguleuse nous mènent rapidement à la solution :



- Une tesselle carrée coûtant 5 pièces de monnaie peut toujours être remplacée par deux rectangles à $1 + 1 = 2$ pièces de monnaie, ce qui est toujours moins cher.
- On pourrait également remplacer une tesselle carrée par deux tesselles triangulaires, ce qui serait un peu plus cher avec $2 + 2 = 4$ pièces de monnaies. C'est donc une moins bonne option.

Giulia n'achète donc jamais de carré même s'il irait bien à un certain endroit, mais toujours deux rectangles.

- Un trapèze coûtant 7 pièces de monnaie peut être assemblé avec un carré et un triangle. En remplaçant le carré par deux rectangles, on arrive à $1 + 1 + 2 = 4$ pièces de monnaie pour un trapèze.

Giulia n'achète donc jamais de trapèze même s'il irait bien à un endroit, mais l'assemble toujours à partir de deux rectangles et un triangle.

- On peut remplacer le parallélogramme à 3 pièces de monnaie par deux triangles à $2 + 2 = 4$ pièces de monnaie. Ce n'est pas une bonne option, car c'est plus cher que le parallélogramme. Un parallélogramme pourrait être utile à Giulia, mais cela n'est déterminé que par une analyse plus précise.

Version A		**Version B**	
	• Tête à 5 pièces de monnaie • Corps fait de 4 rectangles et 2 triangles : $1 + 1 + 1 + 1 + 2 + 2 = 8$ pièces		• Tête à 5 pièces de monnaie • Corps fait d'1 parallélogramme, 2 rectangles et 2 triangles : $3 + 1 + 1 + 2 + 2 = 9$ pièces

Si Giulia n'utilise pas de parallélogramme, elle a besoin de deux triangles pour assembler les pointes triangulaires à gauche et à droite du personnage. Elle peut assembler le reste avec des rectangles, comme dans la version A, qui coûte $5 + 8 = 13$ pièces de monnaie.

Le parallélogramme ne va qu'à un endroit du personnage comme montré dans la version B (ou reflété horizontalement). Si un parallélogramme est placé ainsi et que le reste de la figure est assemblé avec des rectangles et des triangles, le personnage coûte $5 + 9 = 14$ pièces de monnaie. Tous les autres placements du parallélogramme laisseraient des trous ne pouvant pas être remplis.

C'est donc clair que la version la moins chère coûte 13 pièces de monnaie.

C'est de l'informatique !

L'exercice consistant à assembler une figure précise avec des tesselles données peut vite devenir très compliqué même avec peu de pièces. Le jeu de puzzle Tangram en est un exemple.

Le problème ci-dessus est encore plus compliqué car il faut aussi optimiser le prix total des tesselles. En informatique, on appelle un tel problème un *problème d'optimisation*.

Le problème a été résolu grâce à un principe important en informatique : il s'agit de diviser un problème en problèmes plus petits que l'on peut résoudre indépendamment les uns des autres et dont les solutions peuvent se combiner pour obtenir la solution complète. Concrètement, le problème a été divisé en deux plus petits problèmes, un pour les tesselles rondes et un pour les tesselles anguleuse. Pour les tesselles anguleuses, on peut ensuite réutiliser partout la combinaison de tesselles la moins chère pour assembler un carré sans devoir y réfléchir à chaque fois. C'est la même chose pour le parallélogramme.

La division d'un problème en sous-problèmes indépendants est un concept très important en programmation. La réutilisation de solutions pour des problèmes apparaissant plusieurs fois permet de gagner beaucoup de temps. On parle ici du principe de *modularité*. La division en sous-problèmes est

aussi à la base des programmes suivant le principe *« diviser pour régner »* (« *divide et impera* » en latin, « *divide and conquer* » en anglais).

Mots clés et sites web

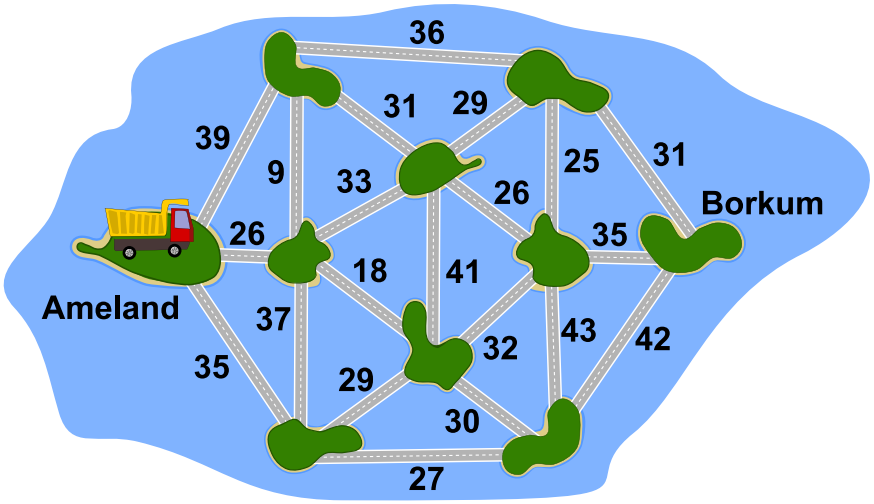
- Problème d'optimisation :
  `https://fr.wikipedia.org/wiki/Optimisation_(mathématiques)`
- Diviser pour régner :
  `https://fr.wikipedia.org/wiki/Diviser_pour_régner_(informatique)`
- Modularité : `https://fr.wikipedia.org/wiki/Notion_de_module`,
  `https://fr.wikipedia.org/wiki/Programmation_modulaire`
- Tangram : `https://fr.wikipedia.org/wiki/Tangram`

29. L'archipel des castors

Dans l'archipel des castors, il y a dix îles qui sont reliées par des ponts, comme sur la carte ci-dessous. Le nombre près de chaque pont indique le poids maximal en tonnes d'un camion pour qu'il puisse le traverser.

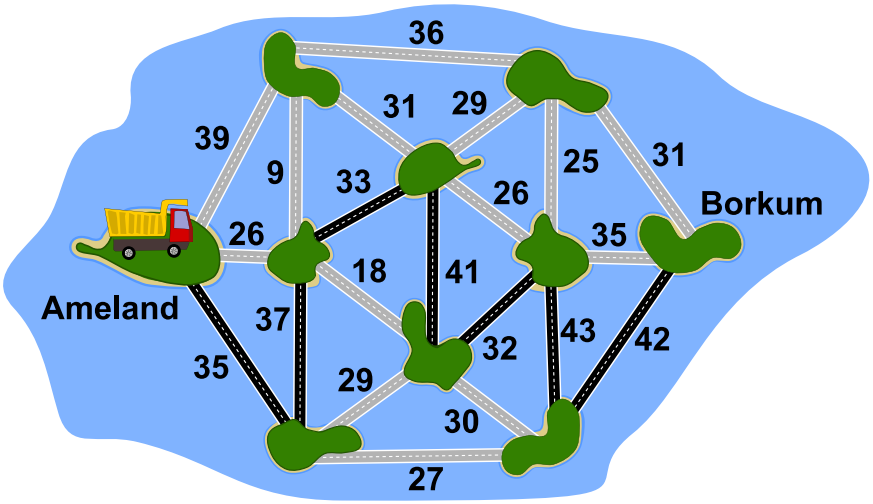
Le castor Knuth aimerait amener du sable sur une plage de l'île de Borkum. Il veut donc transporter autant de sable que possible de l'île d'Ameland à l'île de Borkum en un seul voyage. La longueur de la route à parcourir lui est égale, mais il ne veut prendre aucun pont plus d'une fois.

Quelle route devrait-il emprunter avec son camion pour atteindre Borkum ? Dessine-la sur la carte.

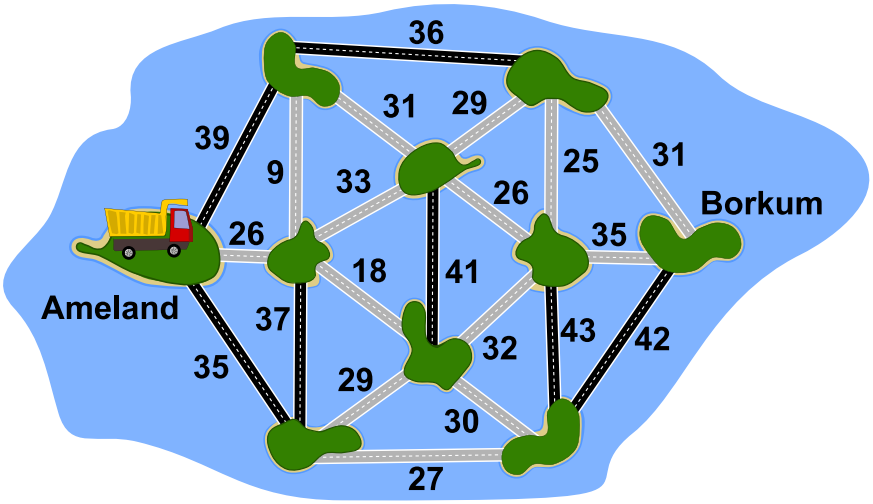


Solution

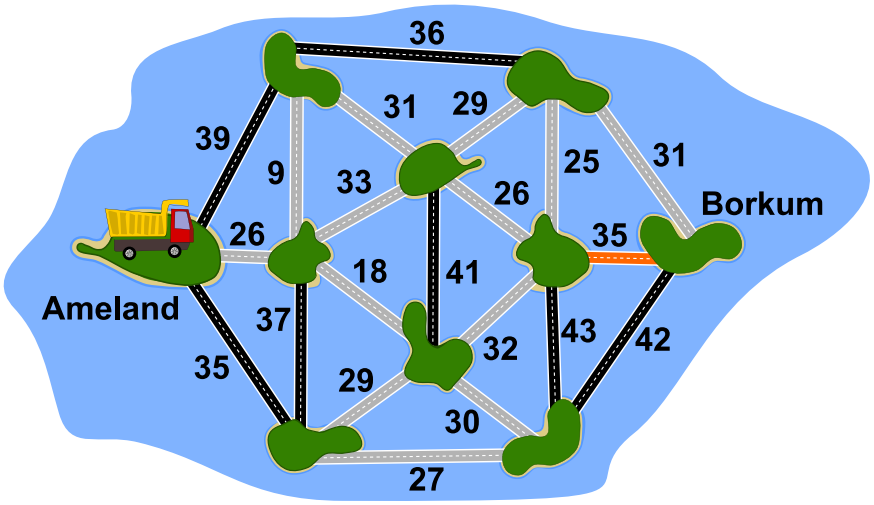
Le poids maximal d'un camion pour ce voyage est de 32 tonnes. Il suit par exemple la route suivante :



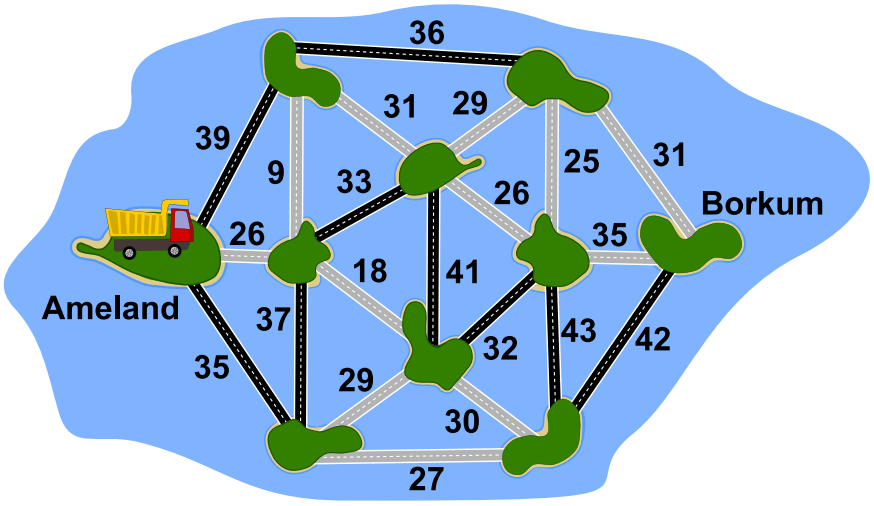
Afin de trouver ce chemin, nous pouvons par exemple commencer par virtuellement enlever tous les ponts de la carte. Nous les trions par capacité de charge. Nous commençons par ajouter à la carte les ponts ayant la plus grande capacité, puis ceux ayant une capacité de charge moindre, et ainsi de suite. Sur la carte suivante, les ponts avec une capacité de charge de 43, 42, 41, 39, 37, 36 et 35 tonnes sont indiqués en noir.



Si l'ajout d'un pont crée un cycle, donc un chemin qui tourne en rond, nous le laissons de côté, car toutes les îles de ce cycle peuvent déjà être atteintes en passant par des ponts avec une plus grande capacité. Dans le diagramme ci-dessous, le pont suivant avec une capacité de charge de 35 tonnes a été ajouté, mais il ne ferait que raccourcir une route déjà présente.



Nous répétons ce procédé jusqu'à ce que toutes les îles soient reliées. Il n'y a maintenant qu'une seule route possible entre chaque paire d'île et le pont avec la plus petite capacité de charge nous indique le poids maximal.



C'est de l'informatique !

Une application réelle de la solution de l'exercice de l'archipel des castors est d'identifier le « goulot d'étranglement » dans un réseau informatique, c'est-à-dire la vitesse de transfert maximale entre deux ordinateurs dans le réseau. Dans cet exercice, le goulot d'étranglement est le poids maximal d'un camion en route entre deux îles. Celui-ci est déterminé par la capacité de charge du pont le plus faible. Dans un réseau informatique, ce serait la connexion avec la plus petite bande passante.

Pour trouver une solution, on peut comme plus haut commencer par modéliser le réseau, donc le simplifier. Dans notre cas, l'*algorithme de Kruskal* est utilisé pour générer un *arbre couvrant* de poids maximal dans lequel le goulot d'étranglement est apparent.

Mots clés et sites web

- Graphe : `https://fr.wikipedia.org/wiki/Graphe_(mathématiques_discrètes)`
- Arbre couvrant de poids minimal : `https://fr.wikipedia.org/wiki/Arbre_couvrant`
- Algorithme de Kruskal : `https://fr.wikipedia.org/wiki/Algorithme_de_Kruskal`

30. Table incomplète

Les castors utilisent un code secret dans lequel chaque lettre est remplacée par un tout nouveau symbole. La table ci-dessous décrit comment les nouveaux symboles sont assemblés. Malheureusement, la table est incomplète car certaines parties ont été effacées.

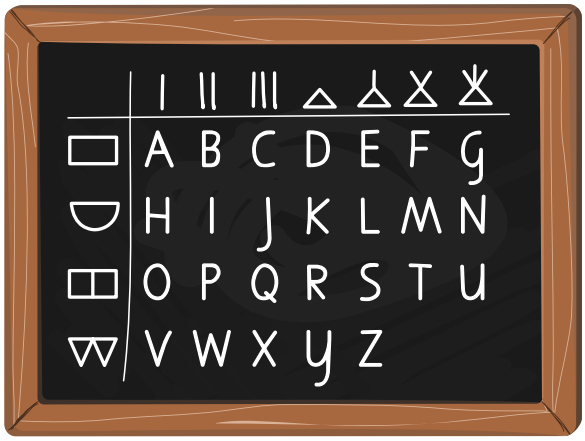
Reconstruis le texte original à partir du cryptogramme suivant (déchiffre le cryptogramme). Laquelle des quatre solutions proposées est-elle juste ?

A) INFORMATIQUE MALINE
B) ELECTRONIQUE MALINE
C) INFORMATION SECRETE
D) INFORMEZ EXACTEMENT

Solution

La bonne réponse est A), le texte clair est : INFORMATIQUE MALINE.

Voici la table de chiffrage complète :



C'est facile de compléter la table. Les lettres de l'alphabet latin sont écrites dans l'ordre, horizontalement et de gauche à droite. On remarque que la partie inférieure des nouveaux symboles correspond à l'intitulé des rangées et la partie supérieure à l'intitulé des colonnes de la table. La seule partie inférieure présente dans le cryptogramme qui manque dans la table est le ▭. C'est donc ce symbole qui est l'intitulé de la première rangée. On peut tout aussi rapidement déterminer les trois symboles manquants dans les colonnes.

Ce n'est cependant pas nécessaire de compléter la table. On peut placer les lettres que l'on peut directement lire dans la table incomplète. On obtient alors le texte à trous suivant :

I N _ O _ _ _ _ I _ _ _ _ _ L I N _

Ce texte à trous permet d'éliminer toutes les solutions sauf A) : B) ne commence pas par « IN », C) et D) ne finissent pas par « LIN_ ».

Une autre solution possible est de remarquer que le cryptogramme possède les deux mêmes symboles à son début et en avant-dernière position. La seule solution avec cette même répétition est la solution B).

C'est de l'informatique !

Garder des informations secrètes ou protéger des données est une tâche vielle de 4000 ans. D'innombrables écritures secrètes ont été développées et utilisées dans ce but. Aujourd'hui, la sécurité des données est l'un des thèmes majeurs de l'informatique. Une des méthodes pour empêcher la lecture non autorisée de données est de les *chiffrer*. Le chiffrement transforme un *texte clair* en *cryptogramme*. La reconstruction du texte clair à partir du cryptogramme s'appelle *déchiffrement*. L'étude des cryptogrammes s'appelle *cryptologie*.

Les cultures antiques utilisaient le plus souvent des écritures secrètes remplaçant des lettres par d'autres lettres ou de tout nouveaux symboles. L'écriture secrète utilisée ici a été développée spécialement pour le Castor Informatique, mais se base sur un concept venant de la Palestine antique. À l'époque, la règle de sécurité était que seules des écriture secrètes faciles à apprendre par cœur pouvaient être utilisées. C'était considéré comme un trop grand risque de garder une description écrite de l'écriture secrète. Une table comme celle utilisée ici est facile à apprendre par cœur. Le célèbre chiffre des francs-maçon se base sur ce principe.

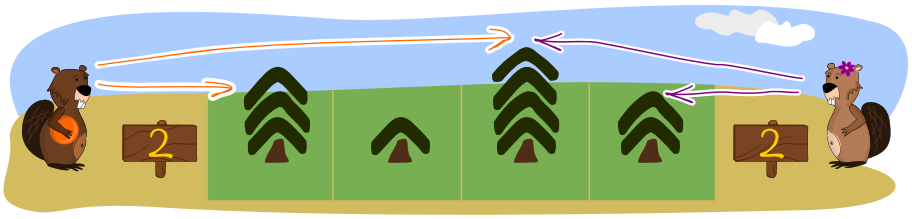
Mots clés et sites web

- Cryptologie : `https://fr.wikipedia.org/wiki/Cryptologie`
- Cryptogramme
- Chiffrer
- Déchiffrer

31. Sudoku boisé 4×4

Les castors plantent 16 arbres (quatre arbres de hauteur 4, quatre arbres de hauteur 3, quatre arbres de hauteur 2 et quatre arbres de hauteur 1) dans un champ de taille 4×4. Pour cela, ils suivent les règles suivantes :

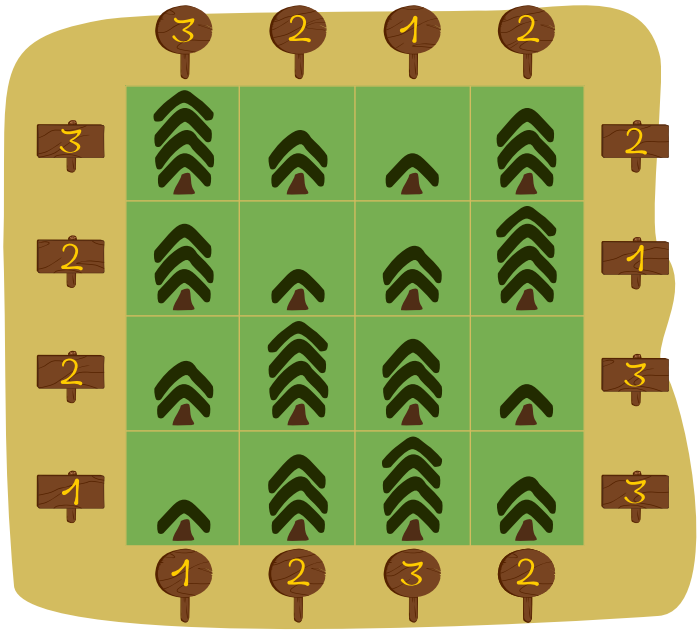
- dans chaque ligne, il y a exactement un arbre de chaque hauteur ;
- dans chaque colonne, il y a exactement un arbre de chaque hauteur.



Lorsque les castors observent une rangée d'arbres depuis l'une de ses extrémités, il ne peuvent **pas** voir les plus petits arbres qui sont cachés derrière de plus grands arbres. C'est écrit sur un panneau au bout de chaque rangée combien de sapins l'on peut voir depuis cet endroit-là. Les panneaux indiquant le nombre de sapins visibles sont plantés tout autour du champ.

Kubko a essayé de représenter le champ d'après sa description sur une feuille de papier. Il a reporté les chiffres sur les panneaux correctement, mais il a fait des erreurs en dessinant quatre des arbres.

Entoure les quatre positions auxquelles les arbres dessinés sont faux et note à côté la hauteur de l'arbre qui devrait s'y trouver.



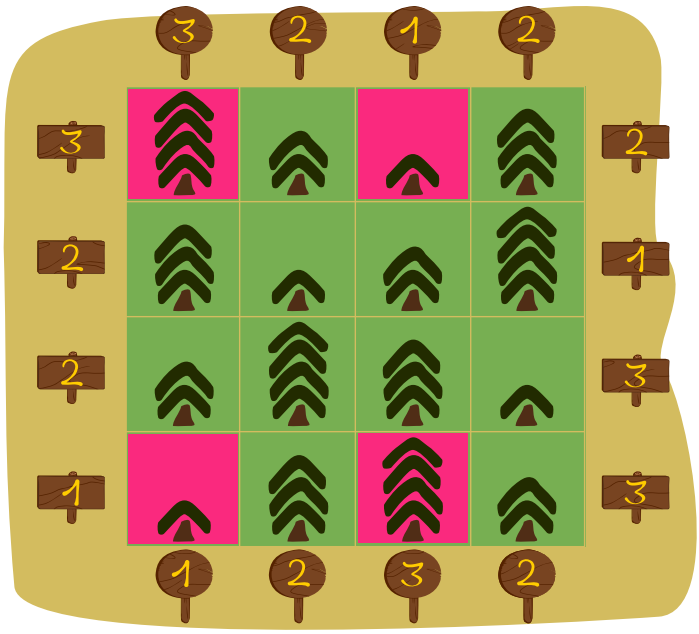
Solution

On remarque d'abord que les deux règles du « sudoku » ont été suivies : il y a exactement un arbre de chaque hauteur dans chaque rangée.

On peut ensuite vérifier pour quelles rangées les chiffres sur les panneaux sont justes et pour lesquels ils ne sont pas. On détermine ainsi que les chiffres pour les lignes 2 et 3 et les colonnes 2 et 4 sont justes. Les chiffres pour les autres rangées ne sont pas justes ; on appelle ces rangées *problématiques*.

Cela ne suffit pas encore. On aimerait savoir quelles positions causent les chiffres erronés. Pour cela, on remarque qu'il y a exactement quatre positions qui se trouvent en même temps dans une ligne et dans une colonne problématique. C'est les positions auxquelles les lignes problématiques (1 et 4) et les colonnes problématiques (1 et 3) se croisent.

On obtient la bonne solution en échangeant les deux arbres d'un ligne ou colonne se trouvant au croisement problématique de cette ligne et de cette colonne (marqués en rouge).



On peut vérifier que ceci est en effet la seule solution possible de la manière suivante : d'après l'énoncé de l'exercice, il y a exactement quatre arbres qui ne sont pas indiqués correctement. Lorsque l'on change un arbre à une position, il faut en changer au minimum deux autres pour que la règle du sudoku soit respectée : un arbre dans la colonne concernée et un dans la ligne concernée. On a donc déjà changé trois arbres. Les deux derniers changements demandent à leur tour un changement chacun dans chaque nouvelles ligne et colonne concernées. Comme l'on ne peut faire que quatre changements en tout, c'est possible uniquement si les deux derniers changements tombent sur la même position et ne font qu'un, ce qui n'arrive que si les quatre positions à changer sont disposées de manière rectangulaire. Comme il fait faire au moins un changement dans chaque rangée problématique, la solution ci-dessus est la seule possible.

C'est de l'informatique !

Cet exercice est centré sur trois compétences fondamentales pour les informaticiennes et informaticiens.

Premièrement, il s'agit de trouver une solution respectant certaines contrainte, ou si nécessaire de corriger une solution proposée.

Deuxièmement, il s'agit de la capacité de reconstruire des objets en se basant sur leur représentation à partir d'informations partielles. Ceci est lié à la génération d'objets (*représentation d'objets*) à partir d'informations disponibles limitées lorsque leur conformité aux lois est connue. On peut aussi utiliser de tels procédés dans la *compression de données*.

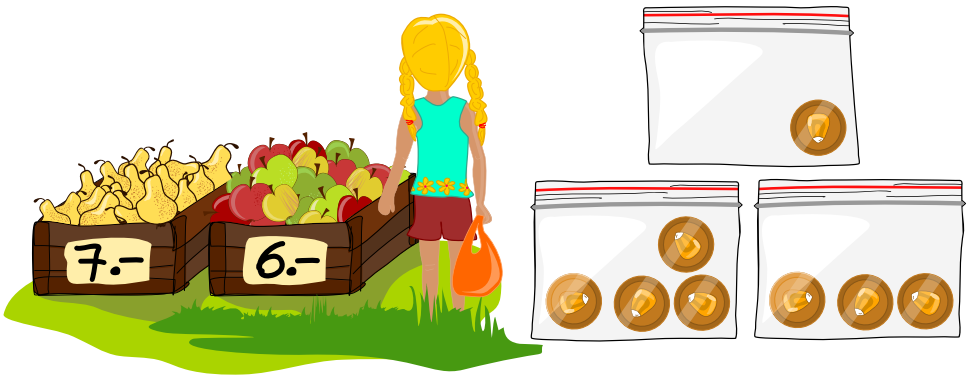
Troisièmement, on peut utiliser de tels champs d'arbres avec des panneaux pour créer des *codes correcteurs*. Des erreurs arrivant lors de l'entrée des données ou du transfert d'information peuvent ainsi être automatiquement reconnues ou même corrigées.

Mots clés et sites web

- Sudoku : `https://fr.wikipedia.org/wiki/Sudoku`
- Représentation d'objets
- Compression de données : `https://fr.wikipedia.org/wiki/Compression_de_données`
- Détection et correction d'erreurs : `https://fr.wikipedia.org/wiki/Code_correcteur`

32. Transport d'argent

Bina aime bien nager. Pour aller dans l'eau, elle met sa monnaie dans des sachets étanches pour que le métal ne commence pas à rouiller. Hier, Bina avait pris trois sachets avec 1, 3 et 4 pièces de monnaie. Comme cela, elle a pu payer une poire exactement (sans qu'on ne lui rende de monnaie) sans devoir ouvrir de sachet, mais pas de pomme.



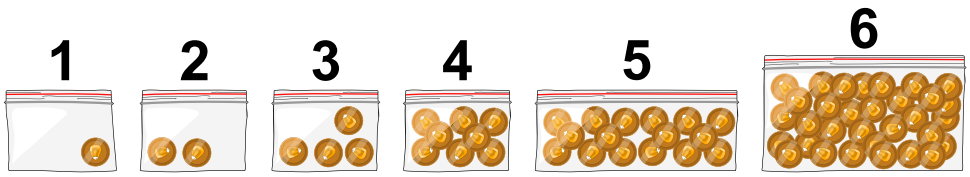
Aujourd'hui, Bina a pris 63 pièces pareilles. Elle aimeraient les répartir dans différents sachets de manière à pouvoir payer tous les montants entre 1 et 63 pièces exactement et sans devoir ouvrir de sachet.

Quel est le plus petit nombre de sachets dont Bina a besoin ?

A) 4 sachets
B) 5 sachets
C) 6 sachets
D) 7 sachets
E) 8 sachets
F) 15 sachets
G) 16 sachets
H) 31 sachets
I) 32 sachets ou plus

Solution

La bonne réponse est C) 6 sachets :



Bina peut répartir les sachets de la manière suivante :

- Sachet 1 : 1 pièce
- Sachet 2 : 2 pièces
- Sachet 3 : 4 pièces
- Sachet 4 : 8 pièces
- Sachet 5 : 16 pièces
- Sachet 6 : 32 pièces

Bina a donc ainsi $1 + 2 + 4 + 8 + 16 + 32 = 63$ pièces dans les sachets et peut payer chaque montant entre 1 et 63 pièces exactement sans qu'on ne lui rende de monnaie et sans devoir ouvrir de sachet.

Pour payer 13 pièces, elle peut par exemple utiliser les sachets 1, 3 et 4.

La table ci-dessous montre comment chaque montant peut être payé exactement en sélectionnant les bons sachets parmi les 6. Une cellule contient un 1 si Bina utilise le sachet correspondant pour payer et un 0 sinon.

Montant	**32**	**16**	**8**	**4**	**2**	**1**	**Montant**	**32**	**16**	**8**	**4**	**2**	**1**
0	0	0	0	0	0	0	**32**	1	0	0	0	0	0
1	0	0	0	0	0	1	**33**	1	0	0	0	0	1
2	0	0	0	0	1	0	**34**	1	0	0	0	1	0
3	0	0	0	0	1	1	**35**	1	0	0	0	1	1
4	0	0	0	1	0	0	**36**	1	0	0	1	0	0
5	0	0	0	1	0	1	**37**	1	0	0	1	0	1
6	0	0	0	1	1	0	**38**	1	0	0	1	1	0
7	0	0	0	1	1	1	**39**	1	0	0	1	1	1
8	0	0	1	0	0	0	**40**	1	0	1	0	0	0
9	0	0	1	0	0	1	**41**	1	0	1	0	0	1
10	0	0	1	0	1	0	**42**	1	0	1	0	1	0
11	0	0	1	0	1	1	**43**	1	0	1	0	1	1
12	0	0	1	1	0	0	**44**	1	0	1	1	0	0
13	0	0	1	1	0	1	**45**	1	0	1	1	0	1
14	0	0	1	1	1	0	**46**	1	0	1	1	1	0
15	0	0	1	1	1	1	**47**	1	0	1	1	1	1
16	0	1	0	0	0	0	**48**	1	1	0	0	0	0
17	0	1	0	0	0	1	**49**	1	1	0	0	0	1
18	0	1	0	0	1	0	**50**	1	1	0	0	1	0
19	0	1	0	0	1	1	**51**	1	1	0	0	1	1
20	0	1	0	1	0	0	**52**	1	1	0	1	0	0
21	0	1	0	1	0	1	**53**	1	1	0	1	0	1
22	0	1	0	1	1	0	**54**	1	1	0	1	1	0
23	0	1	0	1	1	1	**55**	1	1	0	1	1	1
24	0	1	1	0	0	0	**56**	1	1	1	0	0	0
25	0	1	1	0	0	1	**57**	1	1	1	0	0	1
26	0	1	1	0	1	0	**58**	1	1	1	0	1	0
27	0	1	1	0	1	1	**59**	1	1	1	0	1	1
28	0	1	1	1	0	0	**60**	1	1	1	1	0	0
29	0	1	1	1	0	1	**61**	1	1	1	1	0	1
30	0	1	1	1	1	0	**62**	1	1	1	1	1	0
31	0	1	1	1	1	1	**63**	1	1	1	1	1	1

Bina ne peut pas atteindre son but avec moins de 6 sachets. Elle peut utiliser ou non chaque sachet pour payer, il y a donc exactement deux possibilités par sachet. Avec 5 sachets ou moins, elle n'aurait au maximum que $2^5 = 2 \cdot 2 \cdot 2 \cdot 2 \cdot 2 = 32$ possibilités de les combiner. Elle pourrait donc payer exactement au maximum 32 montant différents, ce qui n'est pas suffisant pour tous les montants de 1 à 63 pièces.

C'est de l'informatique !

Cet exercice traite des *nombres binaires*. Les nombres binaires sont étudiés de manière différente en mathématiques et en informatique. En mathématiques, on se concentre surtout sur leurs propriétés, alors qu'en informatique, on s'intéresse à leurs applications. Les ordinateurs utilisent les nombres binaires pour représenter toutes sortes d'informations différentes : des documents, des images, des voix, des vidéos et des nombres, même les programmes et les apps que nous utilisons tous sont codées en nombres binaires. L'unité utilisée est le *bit* (de l'anglais « *b*inary dig*it* », chiffre binaire) qui peut valoir soit 0 soit 1. Un bit ne peut donc représenter que deux possibilités. Avec deux bits, on peut par contre déjà représenter 4 possibilités : 00, 01, 10 et 11. Dans l'exercice ci-dessus, Bina utilise 6 bits (sachets) afin de représenter $2^6 = 64$ montants.

Les ordinateurs rassemblent habituellement les bits en groupes de 8 ; un tel groupe de 8 s'appelle un octet. Un octet peut représenter $2^8 = 256$ nombres différents, de 0 à 255.

Mots clés et sites web

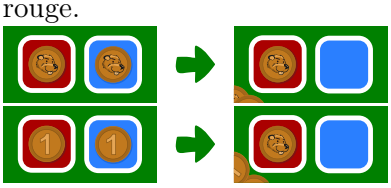
- Nombre binaire : `https://fr.wikipedia.org/wiki/Code_binaire`
- Représentation de données
- Logique
- `https://fr.wikipedia.org/wiki/Bit`, `https://fr.wikipedia.org/wiki/Octet`

33. Las Bebras

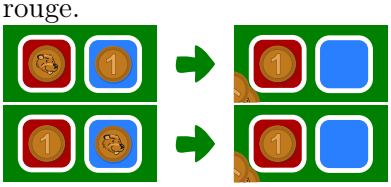
Au casino « Las Bebras », Gloria peut jouer avec des pièces de monnaie à la table de John. Gloria a 4 pièces de monnaie avec, d'un côté, une face, et de l'autre côté, et un chiffre 1. Gloria jette les deux premières pièces et en pose une sur la case rouge et l'autre sur la case bleue.

John échange les deux pièces contre une seule pièce qu'il pose sur la case rouge.

- Si les deux pièces sont pareilles, John met la nouvelle pièce face vers le haut sur la case rouge.



- Si les deux pièces sont différentes, John met la nouvelle pièce chiffre vers le haut sur la case rouge.



Gloria jette maintenant une nouvelle pièce et la met sur la case bleue. John échange à nouveau les pièces en suivant les même règles, et ainsi de suite jusqu'à ce que Gloria ait joué ses 4 pièces. Le jeu est terminé lorsque John pose la dernière pièce sur la case rouge. Si cette pièce est posée chiffre 1 vers le haut, Gloria a gagné !

Gloria jette les quatre pièces dans l'ordre indiqué de droite à gauche. Quelle suite permet à Gloria de gagner ?

A) 

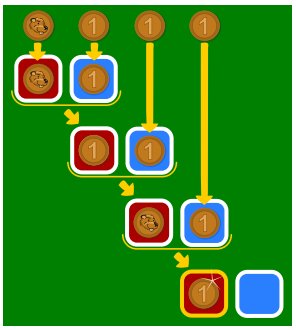
B) 

C) 

D) 

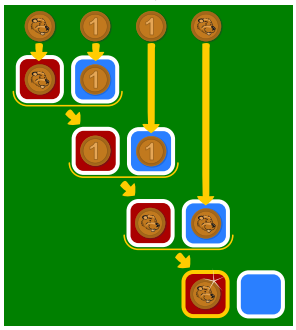
Solution

La bonne réponse est C). C'est la seule réponse pour laquelle la dernière pièce est posée chiffre vers le haut sur la case rouge.

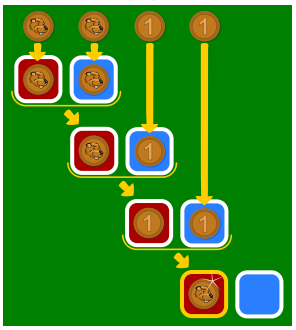


Pour toutes les autres suites, la dernière pièce est posée face vers le haut sur la case rouge.

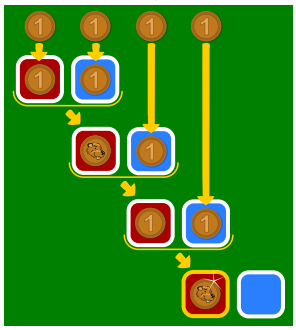
A)



B)



D)



Pour chacune des quatre pièces jouée par Gloria, il y a deux possibilités de les poser (ou), on peut donc jouer $2^4 = 16$ suites différentes avec 4 pièces. Si un nombre pair de pièces est posé face (ou chiffre) vers le haut dans la suite, la dernière pièce du jeu est posée face vers le haut sur la case rouge. Si un nombre impair de pièces est posé face (ou chiffre) vers le haut dans la suite, la dernière pièce du jeu est posée chiffre vers le haut sur la case rouge. Les suites de pièces avec un nombre impair de pièces posées face (ou chiffre) vers le haut sont donc les « suites gagnantes ». Il existe exactement 8 suites différentes ayant un nombre impair et 8 suites différentes ayant un nombre pair de pièces face (ou chiffre) vers le haut.

C'est de l'informatique !

Comme les ordinateurs sont des machines électroniques, l'électricité y est utilisée pour représenter les informations. Deux états peuvent être simplement représentés par la présence ou l'absence d'un courant électrique. Les informaticiens et informaticiennes représentent habituellement ces deux états par les chiffres 0 et 1. Nous appelons cela une *représentation binaire*. Une unité d'information est appelée un *bit*.

Nous pouvons effectuer des opérations sur de tels bits et les combiner, comme l'orientation de deux pièces de monnaie mène à une nouvelle orientation de pièce dans cet exercice.

L'une de ces opérations, appelée *OU exclusif* ou *XOR* (« *eXclusive OR* » en anglais), est présentée dans cet exercice et fonctionne de la manière suivante :

0 XOR 0 = 0
0 XOR 1 = 1
1 XOR 0 = 1
1 XOR 1 = 0

Nous rencontrons aussi de telles opérations dans notre vie quotidienne, par exemple lorsque deux interrupteurs qui allument et éteignent la même lampe se trouvent aux deux bouts d'un escalier. Si les deux interrupteurs sont enclenchés ou éteints, la lampe est allumée. Si l'un des interrupteurs est allumé et l'autre éteint, la lampe est éteinte.

Une porte logique XOR est une application électronique de l'opération XOR dans des ordinateurs. Une porte XOR a 1 comme valeur de sortie si exactement une des valeurs d'entrée est 1. Si les deux valeurs d'entrée sont égales, la valeur de sortie est 0.

L'opération XOR a plusieurs applications en informatique, par exemple :

- Elle nous dit si deux bits sont égaux ou inégaux.
- Elle nous dit si le nombre de bits valant 1 dans une suite de bits est pair ou impair (le résultat du XOR d'une suite de bits est « vrai » quand un nombre impair de bits sont « vrais »).
- En cryptographie, l'opération XOR est utilisée lors du chiffrement symétrique à l'aide de masques jetables.

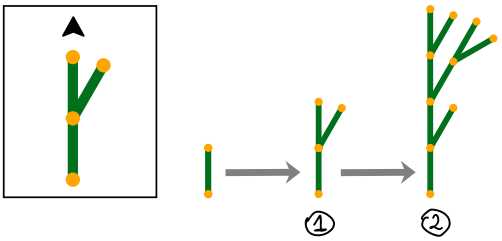
Mots clés et sites web

- Opération binaire : `https://fr.wikipedia.org/wiki/Opération_binaire`
- XOR : `https://fr.wikipedia.org/wiki/Fonction_OU_exclusif`
- Porte logique : `https://fr.wikipedia.org/wiki/Fonction_logique`

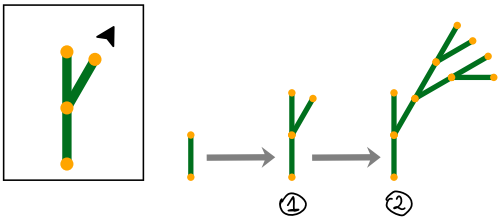
34. Arbres digitaux

Un arbre digital est fait de tronçons d'arbre comme celui-ci :  .

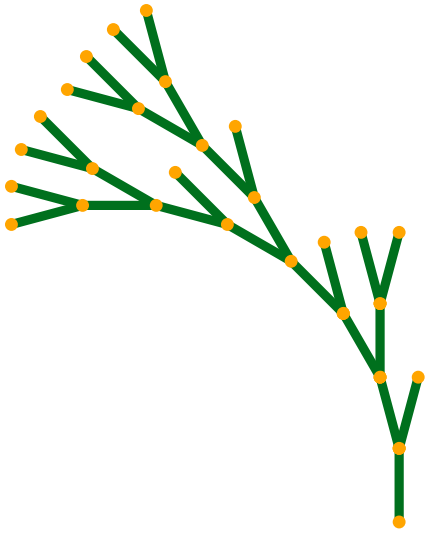
Il pousse étape par étape d'après une règle de croissance définie.
La règle de croissance indique de quelle manière un tronçon est remplacé par une structure composée de nouveaux tronçons. Lors de chaque étape, chaque tronçon est remplacé de cette manière. Une pointe de flèche indique où et dans quelle direction les tronçons sont assemblés.



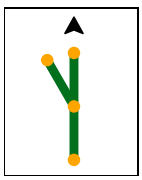
Les exemples à droite montrent deux règles de croissance et les deux premières étapes de croissance correspondantes.

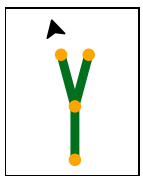


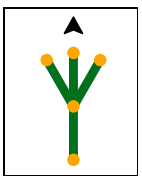
L'arbre suivant a poussé en trois étapes :

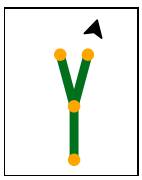


D'après quelle règle de croissance l'arbre digital a-t-il poussé ?

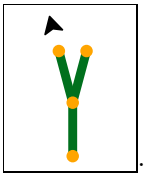
A) 

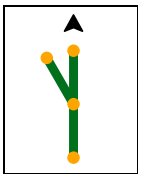
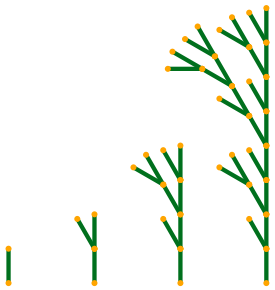
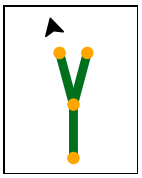
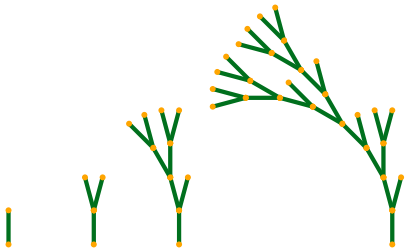
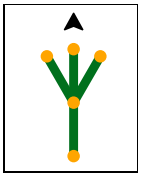
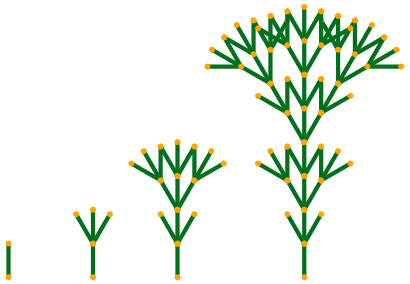
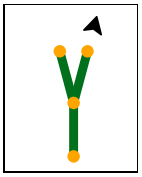
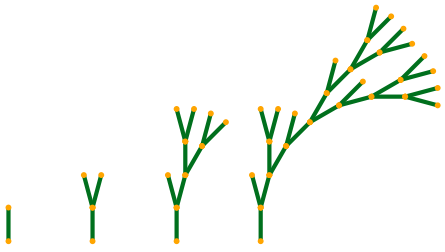
B) 

C) 

D) 

Solution

La bonne réponse est B)  .

Règle de croissance	**Trois étapes de croissance**	**Description**
		Le reste de l'arbre est toujours ajouté à la branche dirigée tout droit vers le haut. Il se forme ainsi un tronc droit avec des branches toutes orientées à gauche.
		Le reste de l'arbre est toujours ajouté à la branche dirigée vers la gauche et le haut. L'arbre est donc penché vers la gauche.
		Le reste de l'arbre est toujours ajouté à la branche du milieu. Les deux branchements à gauche et à droite génèrent une structure régulière et symétrique.
		Le reste de l'arbre est toujours ajouté à la branche dirigée vers la droite et le haut. L'arbre est donc penché vers la droite.

C'est de l'informatique !

Dans cet exercice, on voit comment l'application répétée de règles simples peuvent générer des structures compliquées. De telles figures formées de parties semblables à la figure complète sont aussi appelées des *fractales*. En informatique, on recourt très souvent aux fractales, par exemple pour créer des paysages ou des effets spéciaux pour des films.

En biologie, on utilise ce qu'on appelle des *systèmes de Lindenmayer* (ou *L-systèmes*) pour simuler la croissance des plantes. Ce système génère également des fractales. Dans cet exercice, nous avons vu des exemples très simples de L-systèmes.

Les arbres dans cet exercice sont générés par l'application d'une règle sur chaque tronçon d'arbre, puis à nouveau sur chaque tronçon ainsi généré, et ainsi de suite. De tels procédés sont appelés *récursifs*. La concept de la récursivité est important en informatique. Grâce à la récursivité, il est possible de décrire de manière très simple beaucoup de choses compliquées.

Mots clés et sites web

- Fractale : `https://fr.wikipedia.org/wiki/Fractale`
- L-système : `https://fr.wikipedia.org/wiki/L-Système`, `http://paulbourke.net/fractals/lsys/`
- Récursivité : `https://fr.wikipedia.org/wiki/Récursivité`

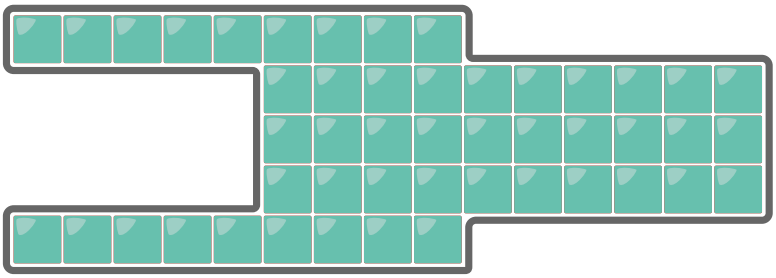
35. Chauffage au sol

Luis n'aime pas se changer dans la salle de bain froide le matin, c'est pourquoi il aimerait installer un chauffage au sol dans la nouvelle maison. Le chauffagiste lui conseille l'innovant « chauffage au sol à hotspots » : un hotspot est installé directement sous une catelle. Lorsque l'on allume le hotspot, cette catelle devient tout de suite chaude.

En une minute, la chaleur se propage à toutes les catelles voisines, c'est-à-dire à toutes les catelles qui touchent le bord ou un angle de la catelle déjà chauffée. Le nombre sur chaque catelle indique au bout de combien de minutes elle devient chaude.

Luis veut installer quatre hotspots dans sa salle de bain de manière à ce que toutes les catelles deviennent chaudes le plus vite possible.

Sous quelles quatre catelles le chauffagiste doit-il installer les quatre hotspots ?

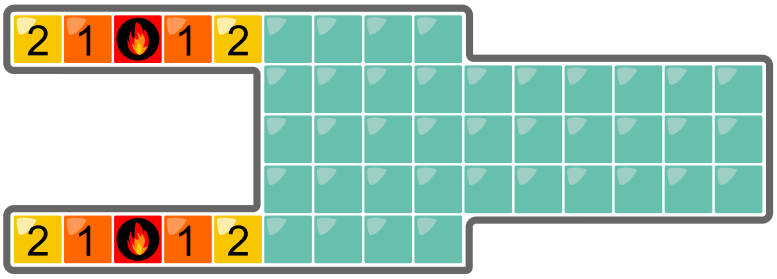


Solution

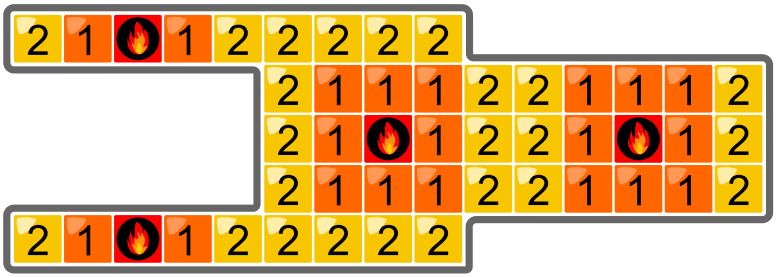
Lorsque les quatre hotspots sont installés comme dans l'image ci-dessous, toutes les catelles de la salle de bain sont chaudes deux minutes après avoir allumé le chauffage.

C'est optimal, car c'est impossible de chauffer toutes les catelles en une minute avec quatre hotspots. On peut le voir de la manière suivante. Chaque hotspot peut chauffer au maximum neuf catelles pendant la première minute, celle sous laquelle il se trouve et jusqu'à 8 catelles autour. Quatre hotspots peuvent donc chauffer au maximum $4 \cdot 9 = 36$ catelles pendant la première minute. La salle de bain a 48 catelles en tout, donc une minute ne peut pas suffire. Cela pourrait par contre marcher en deux minutes, pendant lesquelles $4 \cdot 25 = 100$ catelles pourraient théoriquement être chauffées.

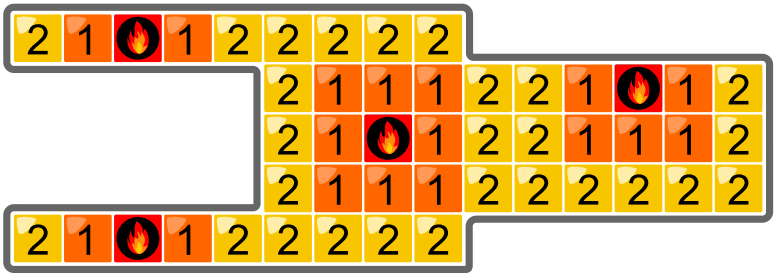
Il serait maintenant logique de commencer à répartir les hotspots dans les deux couloirs à gauche. En mettant un hotspot au milieu de chaque couloir, toutes les catelles des couloirs sont chauffées au bout de deux minutes :

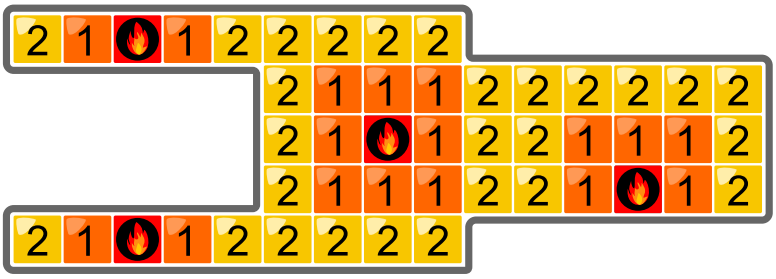


On peut placer les deux autres hotspots comme cela :



Les deux placements suivants sont également possibles :





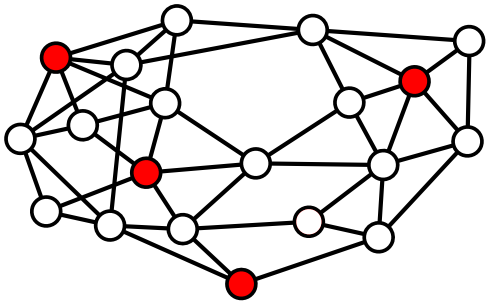
Si la salle de bain avait une forme différente, deux hotspots pourraient suffire à chauffer la même surface en deux minutes.

C'est de l'informatique !

Le problème résolu dans cet exercice est proche d'un problème d'optimisation très connu : on y cherche un petit nombre de *nœuds* dans un *graphe*, nœuds que l'on appelle *ensemble dominant*.

Un ensemble dominant est défini ainsi : chaque nœud du graphe doit soit faire partie de l'ensemble dominant, soit avoir un voisin qui en fait partie. Les catelles de la salle de bain peuvent être représentées avec des nœuds. Les nœuds sont reliés par des arêtes lorsque la catelle suivante est chauffée au bout d'une minute. Un ensemble dominant du graphe résultant indique alors les endroits auxquels des hotspots peuvent être installés pour chauffer la salle de bain en deux minutes.

Dans le cas général, c'est très difficile de trouver un ensemble dominant minimal, mais il existe des algorithmes efficients pour des graphes spéciaux. Le dessin suivant montre un exemple. Comme l'on peut voir, chaque nœud blanc a au moins un nœud rouge comme voisin. Les nœuds rouge sont donc un ensemble dominant.



Une application typique est le placement de bornes Wi-Fi dans un grand bâtiment. Les nœuds du graphe sont les pièces individuelles. Deux d'entre elles sont voisines dans le graphe si les deux pièces sont couvertes par une même borne. Les pièces formant un ensemble dominant minimal sont les endroits appropriés pour placer les bornes Wi-Fi.

Mots clés et sites web

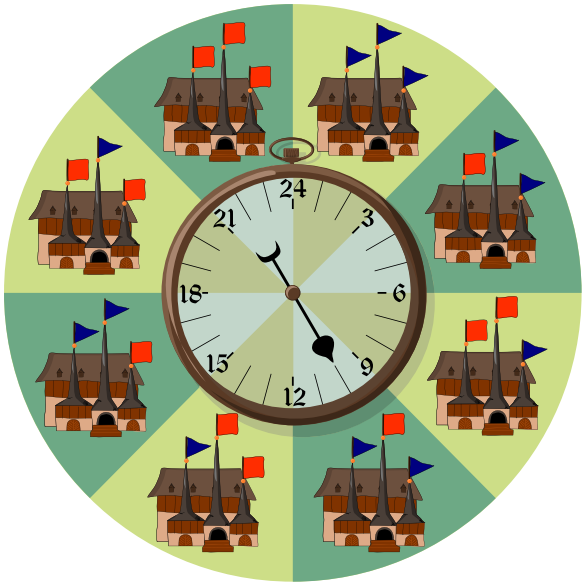
- Ensemble dominant : `https://fr.wikipedia.org/wiki/Ensemble_dominant`

36. Journée tranquille

Les castors vivant dans un petit village tranquille sont très détendus. Ils divisent leurs journées en seulement 8 tranches horaires de 3 heures chacune. La tranche horaire en cours est indiquée par trois drapeaux sur l'hôtel de ville comme représenté sur l'image ci-dessous. Les castors utilisent deux sortes de drapeaux, un carré rouge et un triangle bleu.

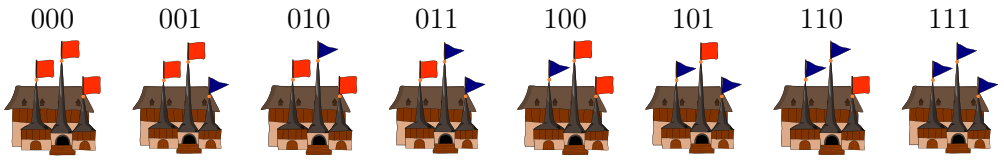
L'arrangement des drapeaux ci-dessus ne demande le changement que d'un seul drapeau à presque chaque transition. Il n'y a qu'à minuit où trois drapeaux doivent être changés d'un coup. Les castors aimeraient trouver un arrangement plus commode qui permette de ne changer qu'un seul drapeau à chaque transition.

Trouve un tel arrangement commode pour les castors et dessine les trois drapeaux de chaque tranche horaire.



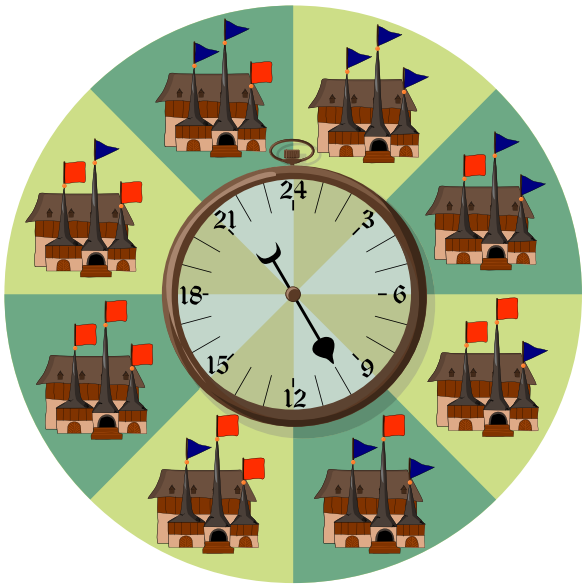
Solution

On peut utiliser des nombres binaires à trois chiffres pour représenter les 8 motifs de drapeaux : 0 représente un carré rouge et 1 un triangle bleu.



Les 8 motifs sont donc 000, 001, 010, 011, 100, 101, 110, 111. Nous devons à présent mettre ces nombres dans un ordre dans lequel les nombres voisins ainsi que le premier et dernier nombre ne diffèrent qu'à une seule position.

On peut y arriver par tâtonnement. Une solution possible est 111, 011, 001, 101, 100, 000, 010, 110. Voici l'horloge correspondante :



On peut trouver une solution de manière systématique avec la méthode suivante :

Nous ne considérons d'abord que les nombres qui commencent avec deux zéros, donc 000 et 001. Ici, il y a deux ordres possibles, et les deux remplissent la condition décrite plus haut. Nous choisissons 000, 001.

Maintenant, nous écrivons les deux mêmes nombres dans l'ordre inverse après les deux premiers (donc 001, 000), mais en changeant la deuxième position de 0 à 1 (donc 011, 010). Nous obtenons ainsi la suite de nombres 000, 001, 011, 010. Elle remplit également la condition.

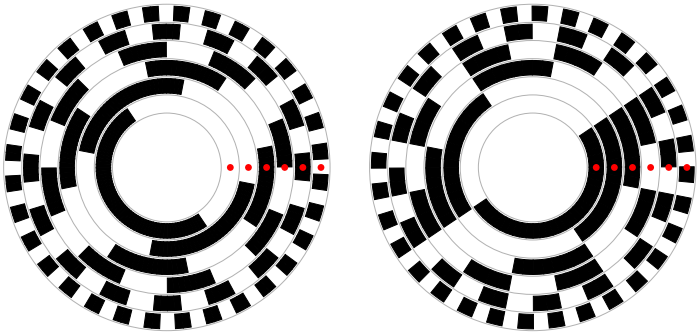
Nous écrivons à nouveau cette nouvelle suite de nombre à l'envers à la suite de la précédente en changeant cette fois la première position de 0 à 1. Nous obtenons ainsi 000, 001, 011, 010, 110, 111, 101, 100, ce qui remplit à nouveau notre condition. Nous avons trouvé la solution recherchée.

Cette méthode (la symétrie de la suite de nombre existante et le changement de la position supérieure de 0 à 1) peut être répétée autant de fois que nécessaire pour obtenir de tels arrangements pour n'importe quel nombre de drapeaux au lieu de trois. On peut se demander pourquoi cette méthode fonctionne toujours et si tous les motifs possibles sont toujours utilisés.

C'est de l'informatique !

Un tel arrangement de nombres binaires s'appelle le *code de Gray* et a beaucoup d'applications. Le fait qu'un seul bit diffère entre deux nombres voisins peut par exemple servir à économiser de l'énergie. Le changement de plusieurs bits demande plus d'énergie, et il y a souvent plusieurs bits qui changent en même temps lors de l'énumération ascendante normale dans le système binaire.

Une application connue du code de gray en ingénierie est la mesure des angles d'une plaque tournante (appelée roue codeuse). On dessine le code de gray sur la plaque comme montré en dessous, en blanc pour 0 et en noir pour 1. Les points rouges sont des détecteurs installés en ligne droite pouvant différencier le blanc du noir. Les détecteurs peuvent ainsi lire un nombre binaire qui code la valeur de l'angle de la roue.



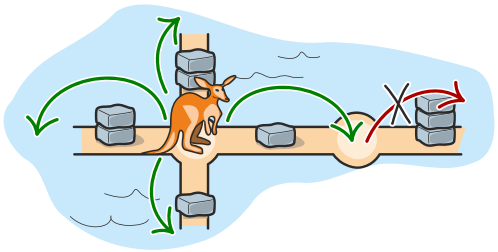
Sur l'image de gauche, on voit que le quatrième détecteur se trouve exactement à la limite entre le blanc et le noir. Le détecteur va donc lire soit 001010 soit 001110. Les deux options sont acceptables, étant donné que la valeur de l'angle se situe exactement au milieu des deux codes. Si l'on n'utilise pas de code de Gray, la situation est plus difficile. Considérons le code binaire normal sur l'image de droite. Ici, les codes 111010 et 111001 se suivent. Si les détecteurs se trouvent exactement entre ces deux codes, les deux derniers détecteurs ne peuvent pas différencier entre le blanc et le noir. Les détecteurs pourraient donc lire le code 111011 qui se trouve plus loin sur la roue. Dans le pire des cas, les détecteurs se trouvent à la limite entre le code blanc 000000 et le code noir 111111, et chaque détecteur peut arbitrairement lire soit 0, soit 1, ce qui rend la mesure de l'angle complètement inutilisable.

Mots clés et sites web

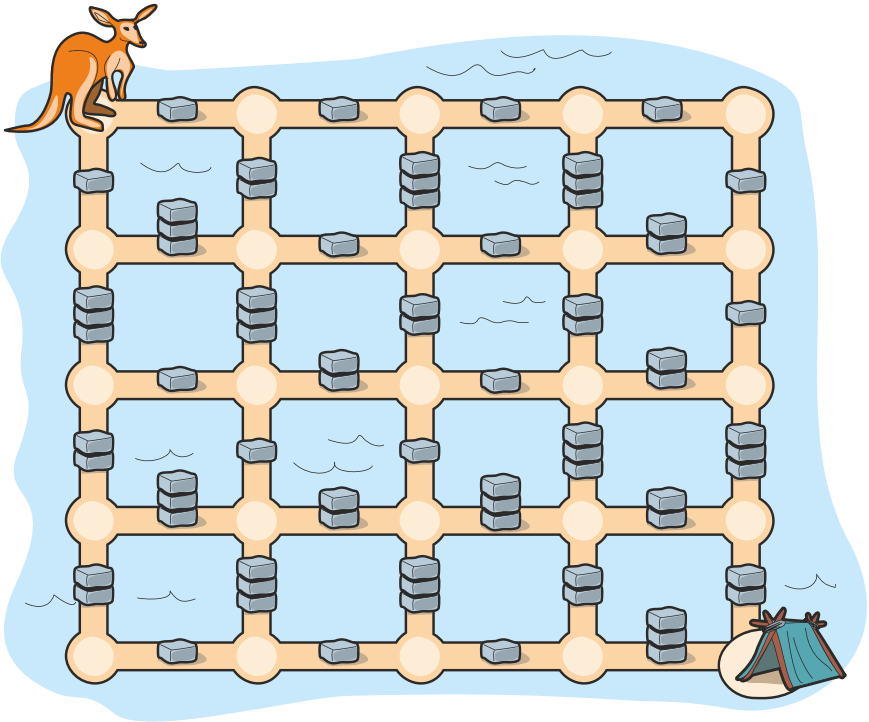
- Code de Gray : `https://fr.wikipedia.org/wiki/Code_de_Gray`
- Roue codeuse : `https://fr.wikipedia.org/wiki/Roue_codeuse`

37. Kangourou bondissant

Un kangourou saute jusqu'à la maison. Il ne peut sauter que sur le chemin et atteint le croisement suivant d'un grand saut. À un croisement, il peut sauter soit à gauche, soit à droite, soit vers le haut, soit vers le bas. Il n'arrive pas à sautant au dessus d'un tas de 3 cailloux.



Le kangourou aimerait rentrer à la maison par le chemin le plus court.

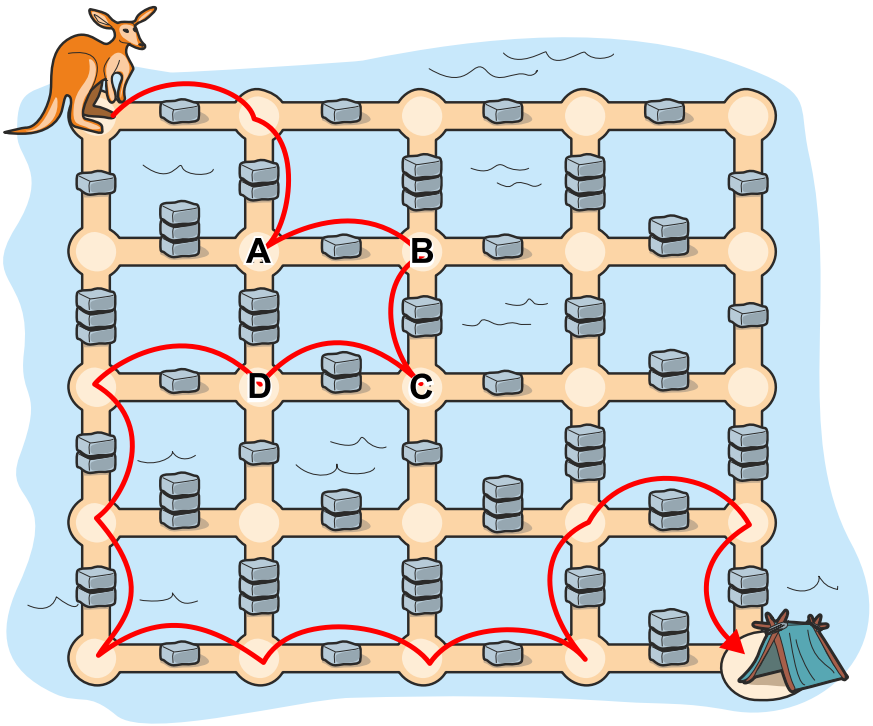


Combien de sauts le kangourou doit-il faire s'il rentre à la maison par le chemin le plus court ?

A) 10 sauts
B) 11 sauts
C) 12 sauts
D) 13 sauts
E) 14 sauts
F) 15 sauts
G) 16 sauts
H) 17 sauts
I) 18 sauts
J) 19 sauts
K) 20 sauts

Solution

La bonne réponse est E) 14 sauts :



Le plus simple est de commencer la recherche par la fin. On voit rapidement qu'il n'y a qu'un chemin possible sur une grande distance depuis l'arrivée, à savoir 9 sauts jusqu'au point D. Maintenant, on ne doit plus que trouver le chemin le plus court depuis le départ jusqu'au point D. En deux sauts, le kangourou arrive au point A. Il ne peut pas sauter directement du point A au point D, car il y a un tas de trois cailloux entre deux. Le détour le plus court pour aller de A à D passe par B et C, le kangourou doit pour cela faire 3 sauts. Le kangourou doit donc faire $2 + 3 + 9 = 14$ sauts en tout ; tous les autres chemins sont plus longs.

C'est de l'informatique !

On peut procéder de la manière suivante pour trouver n'importe quel chemin : on suit un chemin au choix pas à pas. Dès que l'on arrive à une impasse où toutes les directions sont bloquées ou que l'on arrive à un endroit déjà visité du chemin, on revient en arrière jusqu'à trouver une autre direction possible, et on essaie ensuite dans cette direction.

En informatique, cette méthode de résolution s'appelle *retour sur trace* ou *retour arrière* (« *backtrack* » en anglais). Elle est utilisée de manière variée dans différents algorithmes. Elle peut être utilisée pour trouver la solution de puzzles, de sudokus et d'autres problèmes d'optimisation combinatoire.

Cet exercice montre qu'il est parfois plus efficace de commencer par la fin pour trouver une solution. On parle alors d'une *recherche en arrière*. Dans notre cas, on a besoin de faire moins de retour sur

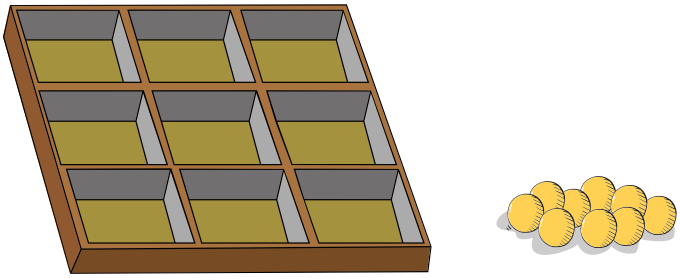
trace, car on a plus d'options du tout à la fin du chemin. On ne peut pas dire de manière générale qu'une recherche en arrière ou une recherche en avant est mieux, cela dépend du problème concret.

Mots clés et sites web

- Retour sur trace : `https://fr.wikipedia.org/wiki/Retour_sur_trace`

38. Des cases et des billes

Hira a une boite qui est divisée en 9 cases, et un nombre de billes illimité :



Hira met des billes dans les cases de la boîte. Elle suit les règles suivantes :

- elle met au maximum une bille dans chaque case ;
- le nombre de billes total dans chaque ligne et chaque colonne est pair quand elle a fini.

Combien de motifs différents Hira peut-elle créer ?

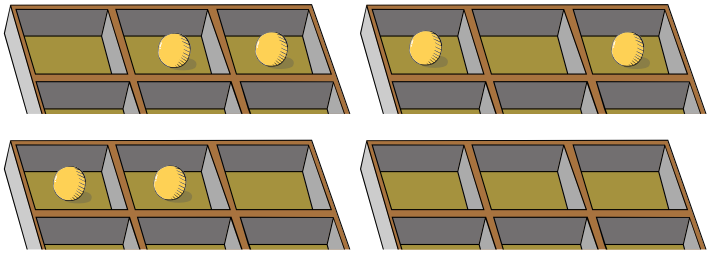
(La boîte ne peut pas être tournée. Le motif avec une bille en haut à gauche est par exemple différent du motif avec un bille en haut à droite.)

A) 12
B) 16
C) 64
D) 512

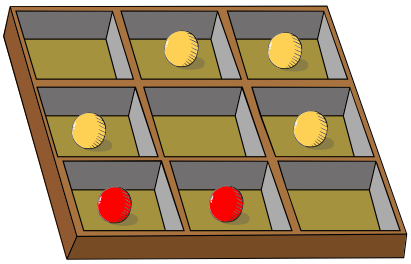
Solution

La bonne réponse est B) 16.

De combien de manières différente Hira peut-elle remplir la première ligne ? Il doit y avoir un nombre pair de billes dans la première ligne, donc 0 ou 2. Il y a donc 4 possibilités de remplir la première ligne :



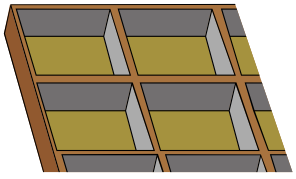
De la même manière, Hira a 4 possibilités de remplir la deuxième ligne. Pour la troisième ligne, elle ne peut plus choisir, car il doit aussi y avoir un nombre pair de bille dans chacune des trois colonnes. S'il y a un nombre impair de billes dans les deux cases du haut d'une colonne (donc exactement une bille), Hira doit mettre une bille dans la troisième case de cette colonne, comme illustré dans les deux premières ligne de l'exemple suivant (billes rouges) :



S'il y a un nombre pair de billes dans les deux premières cases d'une colonne (donc 0 ou 2 billes), elle ne peut pas mettre de bille dans la troisième case de cette colonne, comme c'est le cas dans la troisième colonne de l'exemple en dessus.

Comme le choix pour la première ligne est complètement indépendant du choix pour la deuxième ligne, Hira a 4 possibilités pour la première ligne, et a ensuite à nouveau 4 possibilités pour la deuxième ligne pour chacune de ces quatre possibilités. Elle a donc en tout $4 \cdot 4 = 16$ possibilités.

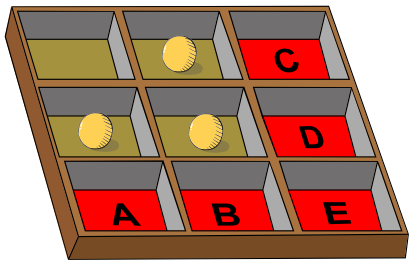
Un autre option pour compter les possibilités est la suivante : on commence par considérer une partie de la boîte faisant 2×2 cases.



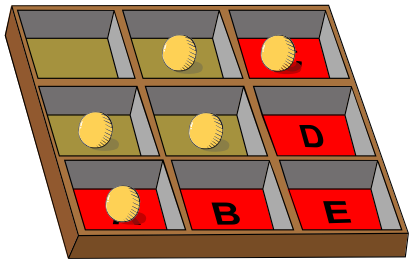
Dans cette partie de la boîte, il y a 4 cases qui peuvent chacune contenir une bille ou pas. Il y a donc $2^4 = 16$ possibilités de remplir cette partie de boîte avec des billes.

Une observation importante est la suivante : une fois que les billes ont été placées dans cette partie de la boîte, Hira n'a plus aucun choix concernant le remplissage des cases restantes. Pour chaque case restante au bord à droite ou dans la ligne du bas, Hira doit obligatoirement soit mettre un bille dans la case, soit la laisser vide, afin que le nombre total de bille soit pair.

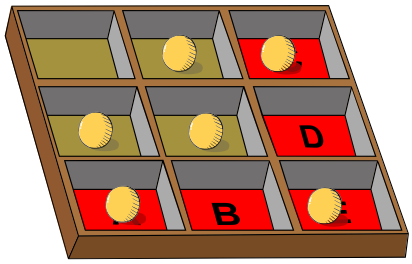
Hira pourrait par exemple remplir la partie de boîte de 2×2 que l'on considère comme cela :



Comme la première colonne ne contient qu'un bille, Hira doit mettre une bille dans la case A pour que le nombre de billes dans cette colonne soit pair. Dans la deuxième colonne, il y a déjà un nombre pair de billes, donc Hira ne peut pas mettre de billes dans la case B. Avec le même raisonnement, on voit que la case D doit rester vide et que Hira doit mettre une bille dans la case C.



Le nombre de billes dans $A + B$ est pair seulement lorsque le nombre de billes dans la partie de boîte de 2×2 cases est pair. Le même chose est vraie pour la somme de $C + D$. Si ces deux sommes sont paires, la case E doit rester vide ; si ces deux sommes sont impaires, Hira doit mettre une bille dans la case E.



Ceci montre que Hira peut mettre des billes de 16 façons différentes dans les cases de la boîte.

C'est de l'informatique !

Une tâche importante de l'informatique est la transmission de données de manière sûre. Une manière d'assurer la transmission de données contre les erreurs de transmission est d'instaurer une *convention de parité*.

Un *bit de parité* est calculé sur la base des données à transmettre et est ajouté à la fin des données. Le bit de parité peut à nouveau être calculé lors de la réception des données. Si les données ne correspondent pas au bit de parité, on sait qu'il y a eu une erreur lors de la transmission.

Dans cet exercice, les cases de la dernière ligne et colonne servent de bits de parité. Si le nombre de billes dans les cases est transmis en tant que données, le destinataire peut calculer la somme des lignes et des colonnes. Si celles-ci ne sont pas paires, le destinataire peut informer Hira qu'il y a eu une erreur lors de la transmission.

Un autre compétence informatique est la capacité à compter toutes les solutions ayant certaines propriétés et ainsi de déterminer leur nombre.

Mots clés et sites web

- Bit de parité :
  `https://fr.wikipedia.org/wiki/Somme_de_contrôle#Exemple_:_bit_de_parité`

A. Auteur·e·s des exercices

Serge Adam
Faisal Al-Sudani
Tony René Andersen
Michael Barot
Wilfried Baumann
Carlo Bellettini
Linda Björk Bergsveinsdóttir
Daniela Bezáková
Maksim Bolonkin
Andrey Brodnik
Lucia Budinská
Špela Cerar
Sarah Chan
Marios O. Choudary
Kris Coolsaet
Valentina Dagienė
Tolmantas Dagys
Christian Datzko
Susanne Datzko
Amirmohammad Djazbi
Hanspeter Erni
Nora A. Escherle
Lidia Feklistova
Fabian Frei
Gerald Futschek
Jens Gallenbacher
Tom Grubb
Yasemin Gulbahar
Husnul Hakim
Mathias Hiron
Juraj Hromkovič
Alisher Ikramov
Thomas Ioannou
Tiberiu Iorgulescu
Takeharu Ishizuka
Mile Jovanov
Ungyeol Jung
Vaidotas Kinčius
Sophie Koh
Dennis Komm
Ritambhra Korpal
Chia-Yi Ku
Regula Lacher
Taina Lehtimäki
Marielle Léonard
Judith Lin
Lynn Liu
Matija Lokar
Vu Van Luan
Hiroki Manabe
Pedro Marcelino
Hamed Mohebbi

Kwangsik Moon
Anna Morpurgo
Xavier Muñoz
Hiroyuki Nagataki
Vania Natali
Rana R. Natawigena
Tom Naughton
Ágnes Erdősné Németh
Andrei Nicolicioiu
Dejan Ozebek
Gabriel Parriaux
Elsa Pellet
Jean-Philippe Pellet
Melinda Phelps
Margot Phillipps
Hannah Piper
Wolfgang Pohl
Prathyush Ponnekanti
Raymond Chandra Putra
Susannah Quidilla
Pedro Ribeiro
Chris Roffey
Peter Rossmanith
Eljakim Schrijvers
Vipul Shah
Fei Shang
Wenpan Sheng
Maiko Shimabuku
Timur Sitdikov
Emil Stankov
Preethi Sudharsha
Maciej M. Sysło
Congyu Tian
Peter Tomcsányi
Monika Tomcsányiová
Meng-ting Tsai
Jiří Vaníček
Troy Vasiga
Fan Wang
Michael Weigend
Jonas Winckler
Michal Winczer
Yang Xing
Khairul Anwar Mohamad Zaki
Binru Zhi

B. Sponsoring : Concours 2020

http://www.haslerstiftung.ch/

http://www.baerli-biber.ch/

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

Standortförderung beim Amt für Wirtschaft und Arbeit Kanton Zürich

i-factory (Musée des transports, Lucerne)

http://www.ubs.com/

http://www.oxocard.ch/
OXOcard
OXON

https://educatec.ch/
educaTEC

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

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

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

`http://www.phlu.ch/`
Pädagogische Hochschule Luzern

`https://www.fhnw.ch/de/die-fhnw/hochschulen/ph`
Pädagogische Hochschule FHNW

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

`https://www.zhdk.ch/`
Zürcher Hochschule der Künste

C. Offres ultérieures

Devenez vous aussi membre de la SSIE
`http://svia-ssie-ssii.ch/la-societe/devenir-membre/`
et soutenez le Castor Informatique par votre adhésion
Peuvent devenir membre ordinaire de la SSIE toutes les personnes qui enseignent dans une école primaire, secondaire, professionnelle, un lycée, une haute école ou donnent des cours de formation ou de formation continue.
Les écoles, les associations et autres organisations peuvent être admises en tant que membre collectif.