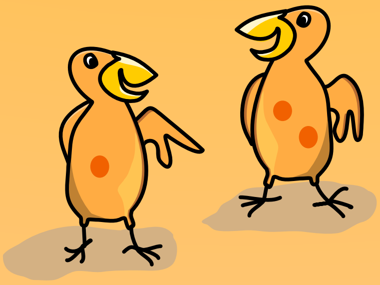
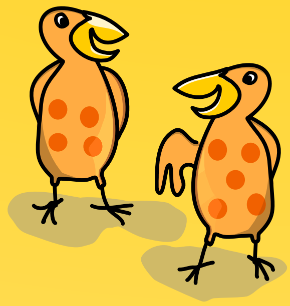
INFORMATIK-BIBER SCHWEIZ
CASTOR INFORMATIQUE SUISSE
CASTORO INFORMATICO SVIZZERA

Exercices et solutions 2021

Tous les âges





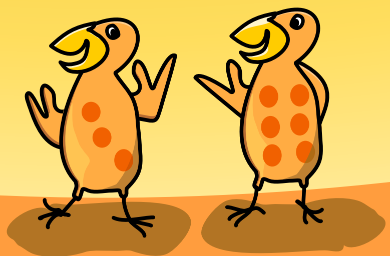
https://www.castor-informatique.ch/

Éditeurs :

Susanne Datzko, Elsa Pellet, Jean-Philippe Pellet,
Fabian Frei, Gabriel Parriaux

SS!E

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



Ont collaboré au Castor Informatique 2021

Masiar Babazadeh, Susanne Datzko, Fabian Frei, Martin Guggisberg, 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
Bernadette Spieler : Pädagogische Hochschule Zürich

Le choix des exercices a été fait en collaboration avec les organisateur de Bebras en Allemagne, Autriche, Hongrie, Slovaquie et Lithuanie. Nous remercions en particulier :
Valentina Dagienė, Tomas Šiaulys, Vaidotas Kinčius : Bebras.org
Wolfgang Pohl, Hannes Endreß, Ulrich Kiesmüller, Kirsten Schlüter, Michael Weigend : Bundesweite Informatikwettbewerbe (BWINF), Allemagne
Wilfried Baumann, Liam Baumann, Anoki Eischer, Thomas Galler, Benjamin Hirsch, Martin Kandlhofer, Katharina Resch-Schobel : Österreichische Computer Gesellschaft
Gerald Futschek, Florentina Voboril : Technische Universität Wien
Zsuzsa Pluhár : ELTE Informatikai Kar, Hongrie
Michal Winzcer : Université Comenius de Bratislava, Slovaquie

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

Pour le support pendant les semaines du concours, nous remercions en plus :
Hanspeter Erni : Direction, école secondaire de Rickenbach

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

Ces brochures sont dédiées à la mémoire de Martin Guggisberg.

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 2021 a été réalisé par la Société Suisse pour l'Informatique dans l'Enseignement (SSIE) et soutenu par la Fondation Hasler.

HASLERSTIFTUNG

Cette brochure a été produite le 24 août 2022 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 2021.

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 2021 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 pour 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. Les tampons de Mika

Mika a quatre tampons avec des images différentes. Elle prend chaque tampon dans sa main une fois et tamponne deux fois avec. Elle fait ainsi l'image suivante :

Quel tampon Mika a-t-elle utilisé en premier ?

A) 

B) 

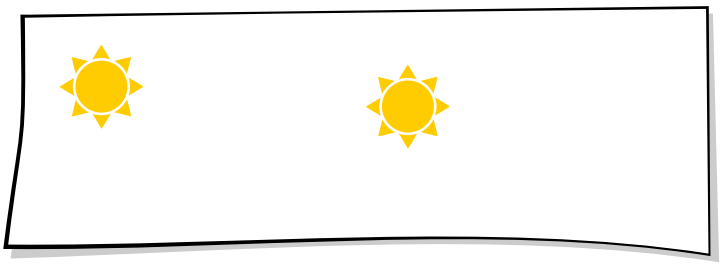
C) 

D) 

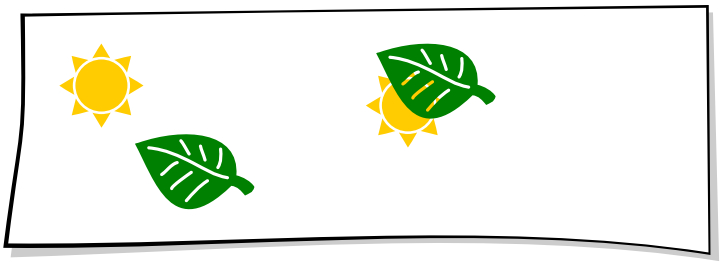
Solution

La bonne réponse est C : 

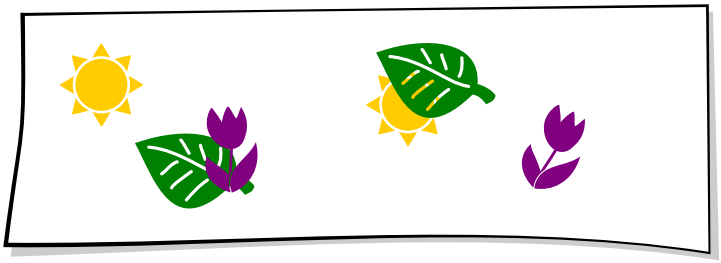
On peut reconnaître l'ordre dans lequel Mika a utilisé les tampons en regardant quelles images sont superposées à d'autres et quelles images sont en dessous d'autre images. Le soleil est en dessous de toutes les autres images : Mika a donc d'abord utilisé le tampon avec le soleil.



La feuille est superposée au soleil, mais est en dessous de la fleur et de la maison. Mika a donc utilisé le tampon avec la feuille en deuxième :



La fleur est superposée à la feuille et au soleil :



Mika ne peut pas avoir utilisé le tampon avec la fleur en dernier, car la fleur se trouve sous la maison à deux endroits. Mika a donc utilisé le tampon avec la fleur en troisième et celui avec la maison en dernier.

C'est de l'informatique !

Le dessin de Mika est une sorte d'image ou de *modèle* de la réalité utilisant des images tamponnées sur quatre plans : un plan avec des soleils, un plan avec des feuilles, un plan avec des fleurs et un plan avec des maisons. La superposition des images des différents plans permet à Mika de créer une illusion de profondeur et d'espace (tridimensionnel) sur une surface (bidimensionnelle) de papier.

Lorsque l'on fait un modèle, on ne représente en général que les aspects qui sont nécessaire à une certaine tâche ou une certaine fonction. La réalité est donc représentée de manière simplifiée. La modélisation est un principe important en informatique.

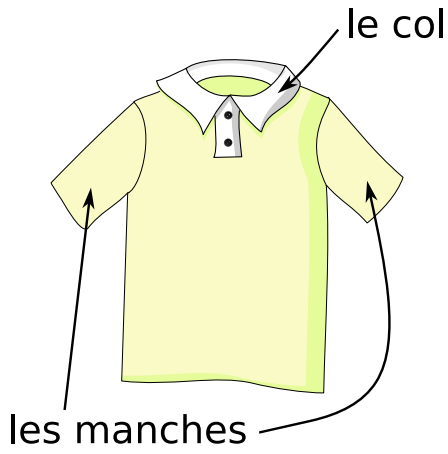
C'est ainsi que des programmes informatiques peuvent permettre d'analyser de manière rapide et précise des parties du monde réel et de mieux comprendre celles-ci. Pour créer un tel programme, il faut commencer par construire un modèle contenant les éléments du monde réel qui sont essentiels au problème étudié. Ce modèle permet de modéliser une partie du monde réel sous forme de croquis ou de programme. On peut ensuite obtenir de nouvelles connaissances sur une partie du monde réel à l'aide de ce programme.

Mots clés et sites web

- Modélisation : `https://fr.wikipedia.org/wiki/Modélisation`
- Calques, traitement d'images : `https://fr.wikipedia.org/wiki/Calque_(infographie)`

2. Le bon maillot

Anne prépare son sac pour aller au match. Aujourd'hui, elle doit prendre le maillot avec des manches claires et un col noir, mais sans rayures.



Quel maillot Anne met-elle dans son sac ?

A)

B)

C)

D)

Solution

La bonne réponse est le maillot B.

Les maillots A et D ne vont pas aujourd'hui parce qu'ils ont des manches noires, et le noir n'est pas une couleur claire.

Le maillot C a des rayures et ne va donc pas pour le match d'aujourd'hui.

Le maillot B est parfait pour aujourd'hui : il a des manches claires, un col noir et n'a pas de rayures.

C'est de l'informatique !

Dans cet exercice du castor, tu devais trouver dans un ensemble d'objets celui qui remplit ou ne remplit pas certaines *conditions*.

Ici, plusieurs sous-conditions ont été définies, comme par exemple la couleur des manches et le motif du tissu, et ont été combinées pour former une condition globale. Pour ce genre de combinaisons, on utilise en informatique des *connecteurs logiques*.

Lorsque toutes les sous-conditions doivent être remplies en même temps, on utilise le connecteur *ET* : « la couleur des manches est claire » *ET* « le col est noir ». S'il suffit qu'au moins une des sous-conditions soit remplie, on utilise le connecteur *OU*. Si l'une des sous-conditions ne peut pas être remplie, on peut utiliser le connecteur *NON*, comme par exemple *NON*(le maillot a des rayures).

Pour effectuer des recherches dans des bases de données, des langages de requête peuvent être utilisés pour formuler des conditions très complexes. Pour cela, les conditions doivent être clairement définies. Par exemple, la condition que les manches doivent être claires n'est pas forcément très bien définie. En informatique, dans un cas comme celui-ci, on écrit un programme ou une fonction qui vérifie si une couleur est claire ou pas. Pour cela, il faut avoir une définition exacte de quand une couleur est claire, sinon, c'est impossible d'écrire un programme qui fonctionne.

Mots clés et sites web

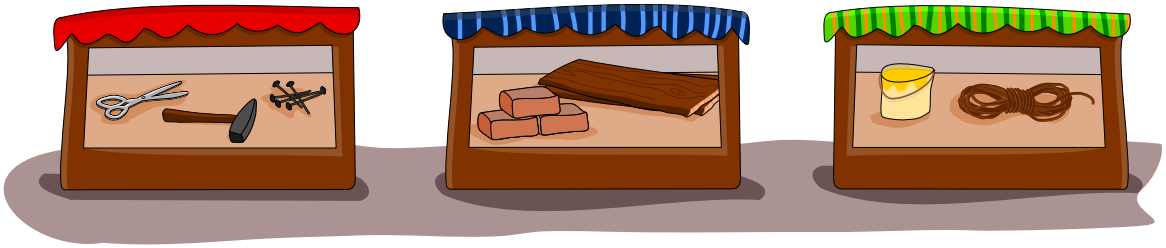
- Algèbre de Boole : `https://fr.wikipedia.org/wiki/Algèbre_de_Boole_(logique)`
- Connecteurs logiques : `https://fr.wikipedia.org/wiki/Connecteur_logique`

3. Construction de pont

Bella aimerait construire un pont par-dessus un ruisseau. Elle a besoin d'un marteau, de clous, de planches et d'une corde. Elle trouve un marteau et une corde à la cave.

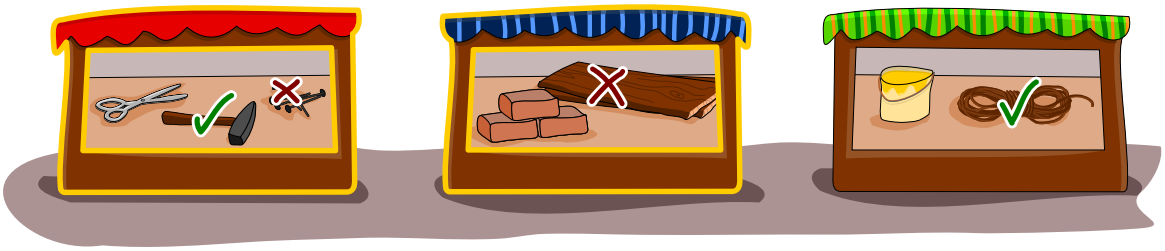
Elle doit acheter les autres objets. En bas, tu vois trois magasins et ce qu'ils vendent.

Dans quels magasins Bella peut-elle acheter les autres objets ?



Solution

La bonne réponse est :



C'est de l'informatique !

Dans cet exercice, les magasins vendent en tout sept ojects différents : des ciseaux, un marteau, des clous, des briques, des planches, une corde et un seau. C'est un grand ensemble d'objets ! Les objets que Bella doit acheter sont une *partie de cet ensemble*. On peut le représenter de la façon suivante : on montre tous les objets de l'ensemble complet et note pour chaque objet s'il appartient au sous-ensemble que Bella doit acheter ou pas :

De la même manière, on peut représenter quels objets sont vendus dans les magasins, par exemple dans le magasin de gauche :

On voit ainsi au premier coup d'œil ce que Bella peut acheter dans le magasin de gauche : Les clous sont marqué d'une coche verte dans le sous-ensemble de Bella et dans le sous-ensemble du magasin.

Les programmes informatiques doivent aussi souvent comparer des ensembles. Pour chaque objet pouvant être présent, on a besoin d'un *bit*. Un ordinateur peut enregistrer une de deux valeurs dans un bit, comme par exemple « oui » ou « non ». Dans notre cas, on enregistre si un objet appartient à un ensemble (« oui ») ou pas (« non »). Un programme peut ensuite comparer deux ensembles de la façon suivante : il vérifie si le bit correspondant à un objet vaut « oui » dans un ensemble et vaut aussi « oui » dans l'autre ensemble. Une telle comparaison de deux bits est très rapide pour un ordinateur. C'est pour cela qu'en informatique, les ensembles sont souvent décrits en utilisant des bits.

Mots clés et sites web

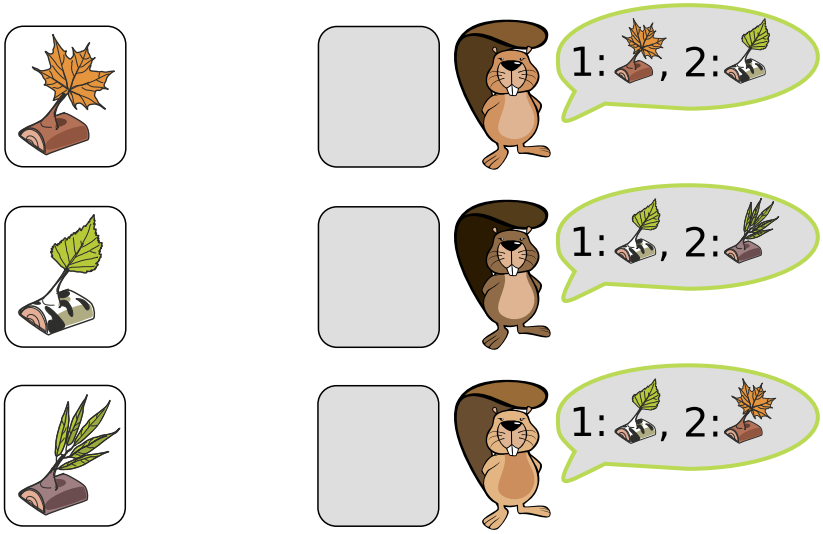
- Ensemble : `https://fr.wikipedia.org/wiki/Ensemble_(informatique)`
- Bits : `https://fr.wikipedia.org/wiki/Bit`
- Tableau de bits : `https://fr.wikipedia.org/wiki/Tableau_de_bits`

4. Cadeau favori

La famille castor a trois cadeaux pour ses trois enfants. Chaque enfant indique d'abord son cadeau favori, puis son second choix. Les cadeaux doivent être bien distribués :

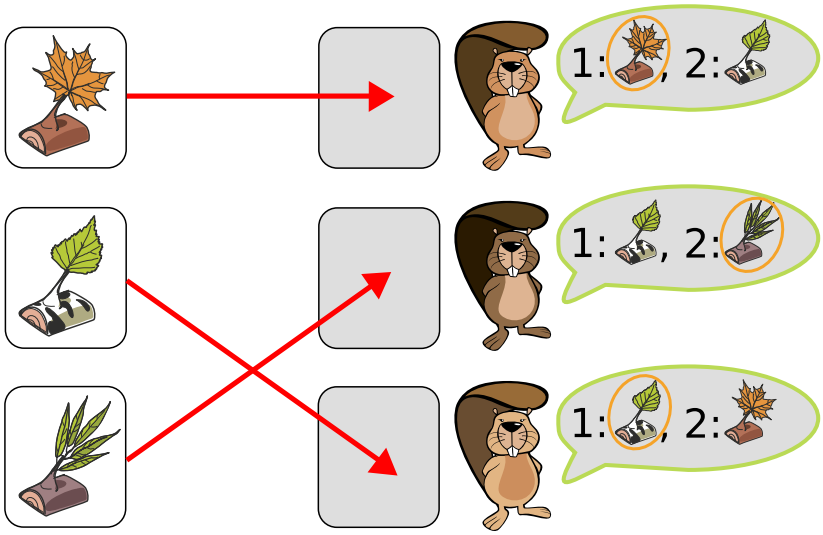
1. Le plus d'enfants possible doivent recevoir leur cadeau favori.
2. Les autres enfants doivent recevoir leur second choix.

Donne les bons cadeaux aux enfants.



Solution

Voici la seule manière de distribuer les cadeaux en respectant les deux conditions.



Seul le deuxième castor désire le troisième cadeau, c'est donc lui qui doit le recevoir. Sinon, un autre castor recevrait un cadeau qui n'est ni son cadeau favori, ni son deuxième choix. La distribution des deux autres cadeaux est claire : chaque castor reçoit son cadeau favori.

C'est de l'informatique !

Dans cet exercice, nous avons affaire à un *problème d'affectation* univoque : nous voulons affecter les cadeaux de manière à ce que tous les enfants recoivent un cadeau. Les enfants n'ont ici pas qu'un seul souhait, mais une liste de préférence. De tels problèmes d'affectation avec listes de préférence peuvent devenir très compliqués. L'informatique nous aide à résoudre de tels problèmes rapidement.

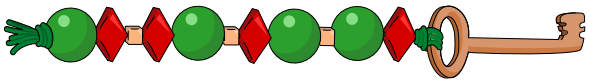
Une possibilité est de donner une valeur aux affectations : le cadeau favori a la valeur 1 et le deuxième choix la valeur 2. Un *couplage* (*matching* en anglais) est optimal s'il n'existe pas d'autre couplage avec plus de premiers choix distribués. Un tel couplage est appelé *couplage parfait de poids minimum*. Il existe beaucoup de problèmes d'affectation. L'un deux est appelé *problème des marriages stables* (*Stable Marriage Problem* en anglais). Intéressant ? L'informatique est une branche très variée !

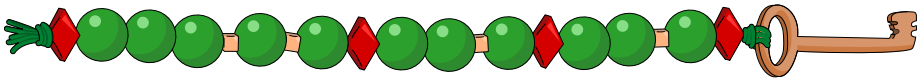
Mots clés et sites web

- Problème d'affectation : `https://fr.wikipedia.org/wiki/Problème_d'affectation`
- Couplage : `https://fr.wikipedia.org/wiki/Couplage_(théorie_des_graphes)`

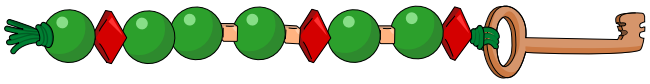
5. Porte-clés

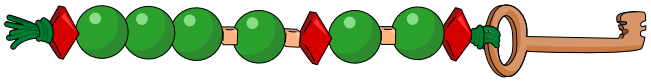
ANNA, BELLA et LENA font des porte-clés avec leur nom. Elles utilisent deux sortes de perles pour les lettres :  et  . Différentes lettres sont séparées par la perle  .

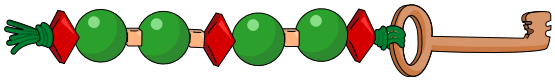
ANNA 

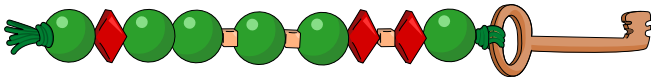
BELLA 

Quel porte-clés LENA a-t-elle fait ?

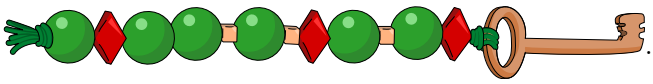
A) 

B) 

C) 

D) 

Solution

La bonne réponse est A)  .

Le mot LENA commence par un L. La lettre L est la troisième et quatrième lettre du mot BELLA, on peut donc reconnaître la suite de perles utilisée pour la lettre L dans BELLA :  . Seules les réponses A et D) commencent par cette lettre et peuvent être la solution. La deuxième lettre de Lena, le E, est aussi la deuxième lettre de BELLA, on peut trouve donc la perle  dans les deux porte-clés. Les réponses A) et D) ont les deux la perle  comme deuxième lettre, les deux peuvent donc toujours être la solution. On veut ensuite trouver quelles perles sont utilisées pour la lettre N. On trouve la suite de perles  dans le porte-clé d'ANNA. On ne trouve ces perles pour la lettre suivante que dans la réponse A).

Une autre possibilité pour trouver quelles perles sont utilisées pour le porte-clés de LENA est de faire un tableau montrant les perles correspondant aux lettres connues. Le porte-clés d'ANNA nous donne la suite de perles  pour la lettre A et la suite de perles  pour la lettre N. Le porte-clés de BELLA nous donne la suite de perles pour B :  , pour E :  et pour L :  .

Lettres	Perles
A	
N	
B	
E	
L	

On peut donc faire le porte-clés de LENA avec les suites de perles  ,  ,  et  si on sépare les différentes lettres avec la perle  . On obtient le porte-clés  , ce qui correspond à la réponse A). Si l'on décode les autres porte-clés à l'aide du tableau, on trouve BENA pour B), NENA pour C) et LEAN pour D).

C'est de l'informatique !

L'information est *encodée* pour pouvoir transférer des messages sous certaines conditions ou pour transmettre des informations de manière secrète (*chiffrée*). Le codage de cet exercice est basé sur l'alphabet Morse. le point • de l'alphabet Morse est représenté par la perle ronde  et le trait – par . La lettre A s'écrit • – en alphabet Morse, donc  en perles. Pour encoder des textes quelconques, il faut un code pour chaque lettre de l'alphabet.

L'alphabet Morse a été créé au XIX^e siècle. Samuel Morse inventa un télégraphe électromagnétique simple en 1837. Le code utilisé à l'époque ne comprenait que les dix chiffres de 0 à 9 ; les chiffres transmis devaient être traduits en lettres puis en mots à l'aide d'une tableau. Albert Lewis Vail, un collègue de Morse, developpa dès 1838 le premier code comprenant aussi des lettres. Le code fut développé pour pouvoir transmettre des textes par voie acoustique, optique ou électromagnétique à

l'aide d'un télégraphe. Un point représente une courte durée de transmission et un trait une durée trois fois plus longue. Un pause sépare les lettres, et une pause plus longue sépare les mots. L'alphabet Morse est encore utilisé aujourd'hui pour le signal SOS. Un SOS en Morse • • • — — — • • • (3x court, 3x long, 3x court) peut être transmis facilement en criant, tapant ou avec une lampe de poche.

Dans le traitement de données informatiques, les caractères sont encodés par des valeurs numériques pour être transmis ou enregistrés.

Mots clés et sites web

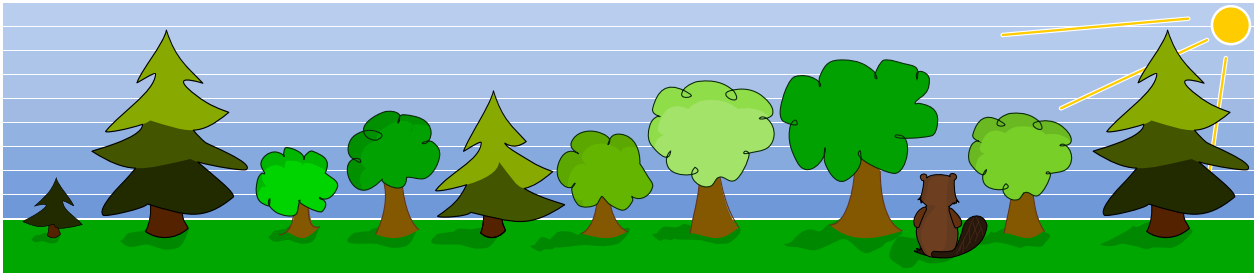
- Codage des caractères : `https://fr.wikipedia.org/wiki/Codage_des_caractères`
- Alphabet Morse : `https://fr.wikipedia.org/wiki/Code_Morse_international`

6. Timber !

Un castor aimerait construire un barrage. Afin de toujours abattre les bons arbres, il s'est fixé deux règles. Il n'abat un arbre que si :

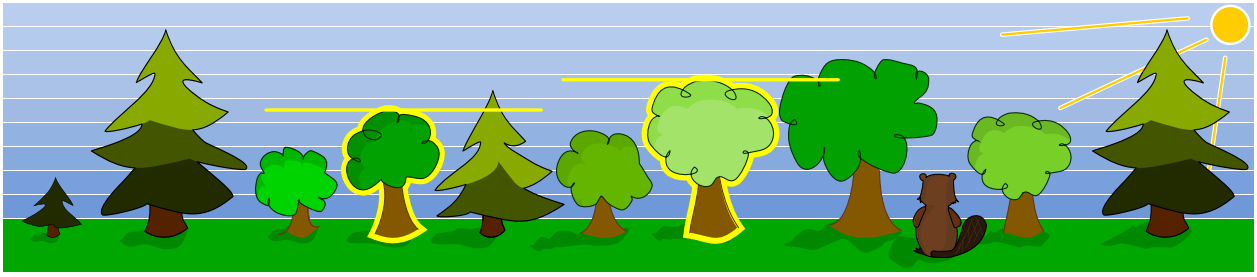
- un arbre plus petit pousse directement à sa gauche et
- un arbre plus grand pousse directement à sa droite.

Quels arbres le castor va-t-il abattre ?



Solution

Seuls les arbres à la quatrième et septième positions remplissent les conditions données : il y a un arbre plus petit directement à gauche ET un arbre plus grand directement à droite.



C'est de l'informatique !

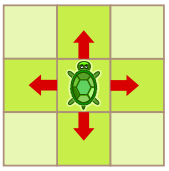
En informatique, il faut souvent résoudre des problèmes qui sont spécifiés par une série de *contraintes* logiques. La tâche consiste à trouver une solution qui respecte toutes les contraintes. Des problème plus complexes que celui-ci peuvent être résolus en utilisant des *opérateurs logiques* pour combiner les contraintes. La conjonction ($\wedge$ ou encore opérateur ET) donne par exemple le résultat « vrai » pour l'expression A $\wedge$ B lorsque les deux contraintes A et B sont vraies. Dans cet exercice, ce serait donc : « l'arbre de gauche est plus petit » $\wedge$ « l'arbre de droite est plus grand ». On retrouve ce principe fondamental dans presque tous les domaines de l'informatique, par exemple dans beaucoup d'algorithmes de tri comme le *tri à bulles* lors duquel la satisfaction aux contraintes de plusieurs objets d'une liste est évaluée avant de les déplacer, si nécessaire, à une autre position. Ce procédé est répété jusqu'à ce que la liste soit complètement triée.

Mots clés et sites web

- Pensée algorithmique (*algorithmic thinking*)
- Opérateur logique : `https://fr.wikipedia.org/wiki/Connecteur_logique`
- Tri à bulles : `https://fr.wikipedia.org/wiki/Tri_à_bulles`
- Tri : `https://sorting.at/`
- Problème de satisfaction de contraintes : `https://fr.wikipedia.org/wiki/Problème_de_satisfaction_de_contraintes`

7. Chemins tortueux

Une tortue doit brouter plusieurs jardins. Chaque jardin est divisé en carrés qui sont recouverts soit de gazon, soit de pierres. La tortue ne peut pas traverser un carré avec des pierres, mais elle peut passer d'une case d'herbe à une autre case d'herbe qui se trouve directement à côté.

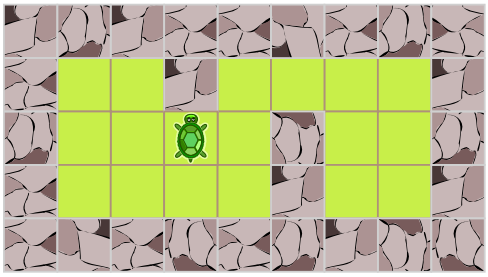


La tortue doit complètement brouter les jardins. elle commence sur la case sur laquelle elle est sur l'image. À la fin, elle doit avoir passé exactement une fois sur chaque case d'herbe.

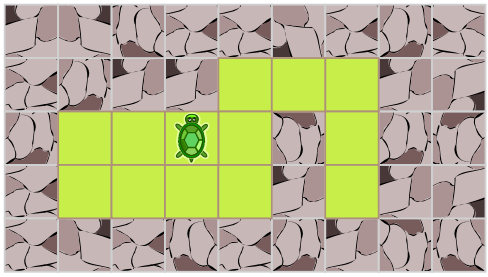
Il y a malheureusement un jardin qu'elle ne peut pas brouter complètement de cette façon.

De quel jardin s'agit-il ?

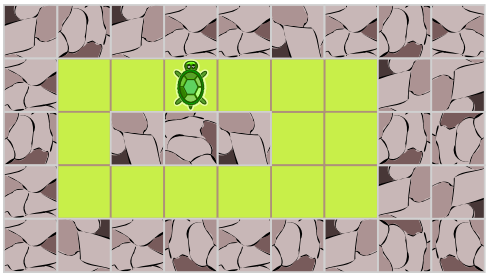
A)



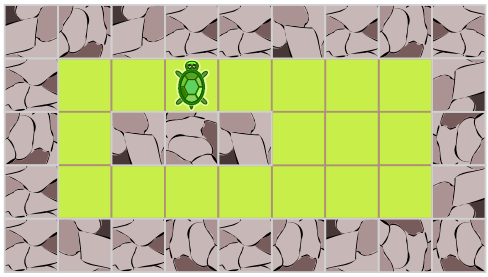
B)



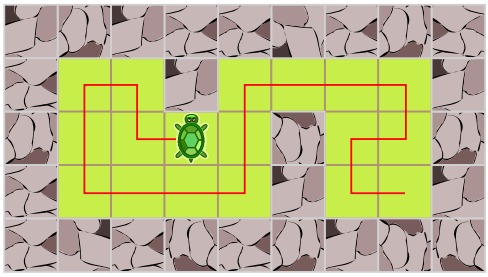
C)

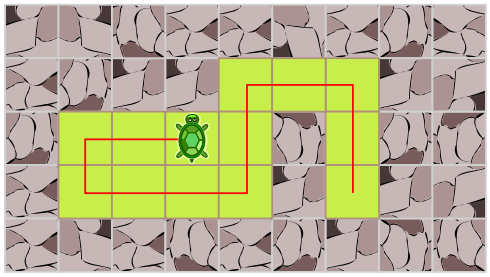


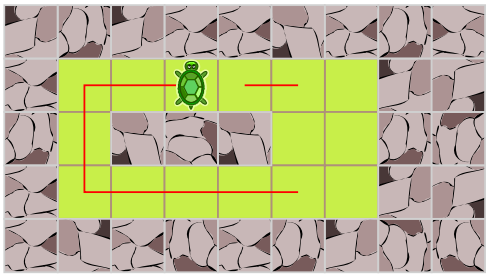
D)

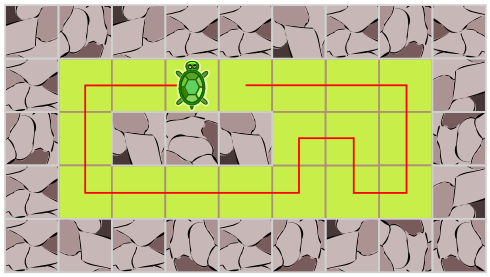


Solution

A) 

B) 

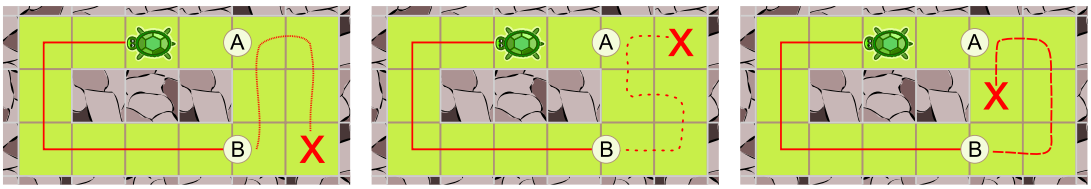
C) 

D) 

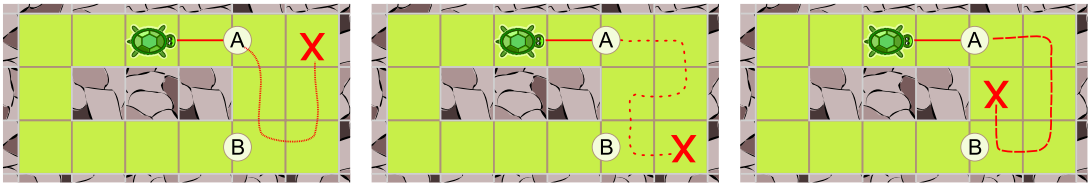
La tortue peut brouter les jardins A, B et D complètement.

La tortue ne peut pas brouter le jardin C de cette façon. La tortue a deux possibilités depuis son point de départ :

- Si elle part d'abord vers la gauche, elle arrive au point B. Depuis là, elle devrait brouter les 6 cases de droite de manière à terminer au point A, mais aucun des chemins possibles depuis le point B ne se termine au point A.



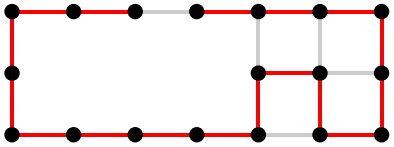
- Si la tortue part d'abord vers la droite, alle arrive au point B et devrait brouter les 6 cases de manière à atteindre le point A à la fin. On peut utiliser le même argument que si dessus en inversant le haut et le bas. Il n'y a donc pas non plus de chemin adapté.



C'est de l'informatique !

La tortue doit trouver un chemin à travers un jardin en passant exactement une fois par chaque case d'herbe. Le problème de cet exercice est un problème du type *chemin hamiltonien*

Les carrés d'herbes du jardin de la tortues peuvent être considérés comme des *nœuds*. On représente alors le jardin D comme ceci :



Au XIX^e siècle, Sir William Hamilton se demanda s'il existait pour de telle structures (appelées *graphes* en informatique et en mathématiques) un chemin les long des arêtes passant exactement une fois par chaque nœud. C'est pour cela que l'on appele un tel chemin un *chemin hamiltonien*. La question de savoir si un chemin hamiltonien existe dans un certain graphe est en règle générale très difficile à résoudre. Personne de connait d'*algorithme* permettant de déterminer si un chemin hamiltonien existe dans un graphe quelconque de manière efficiente (assez rapidement pour que ce soit utile). Cela est vrai de tous les problèmes dits *NP-complets*, dont le problème du chemin hamiltonien est le plus connu.

Mots clés et sites web

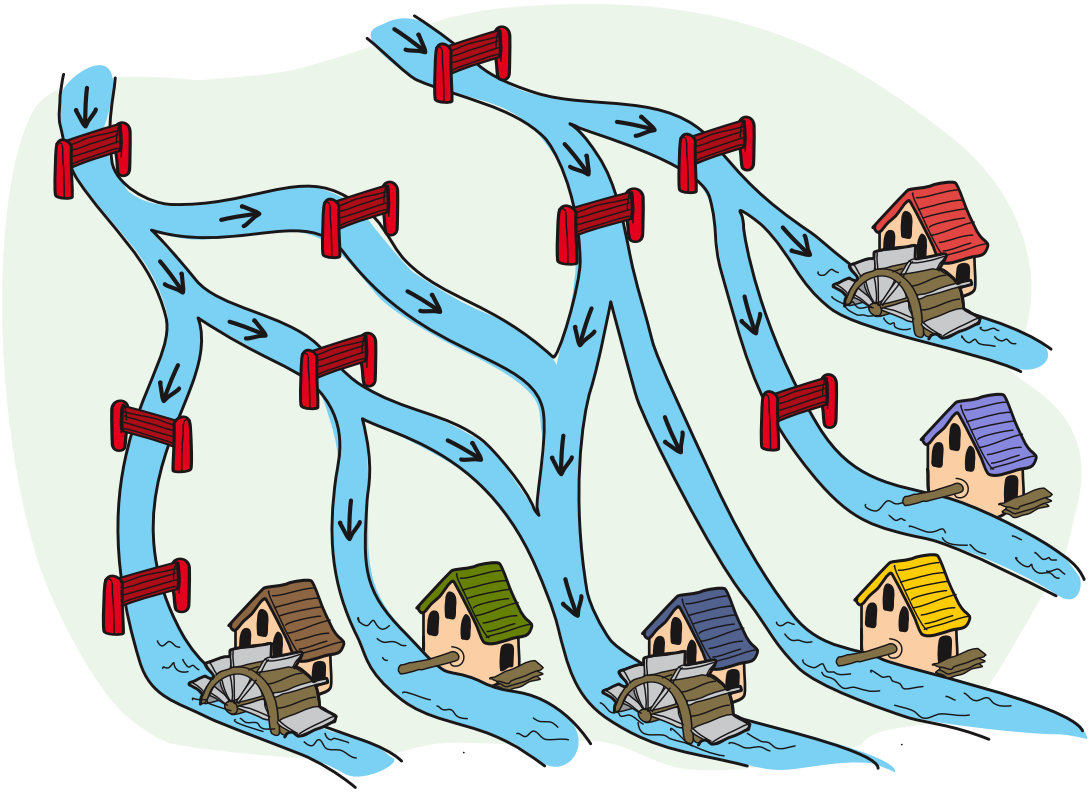
- Graphe hamiltonien, chemin hamiltonien :
  `https://fr.wikipedia.org/wiki/Graphe_hamiltonien`

8. Les moulins de castor Max

Le meunier Max a six moulins. Il doit encore fixer la roue de trois d'entre eux. Pour cela, il doit empêcher l'eau d'arriver à ces moulins. L'eau doit par contre continuer de couler jusqu'aux autres moulins.

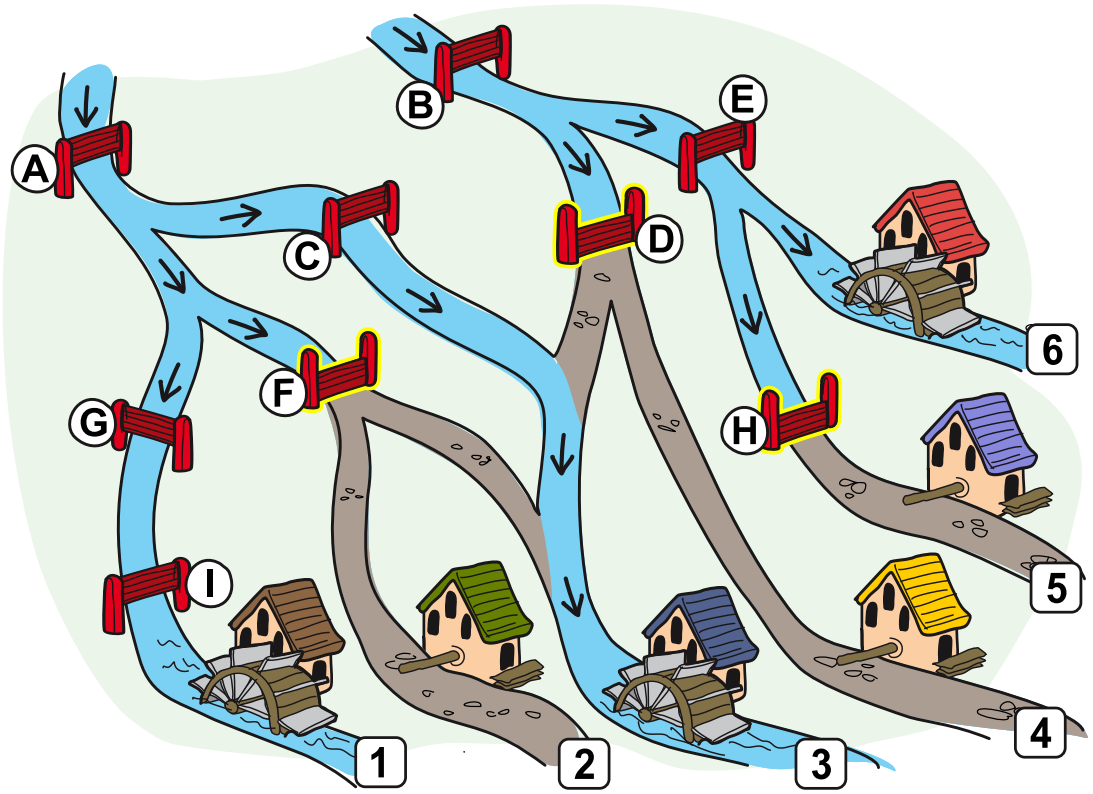
L'eau ne peut couler que vers le bas. Un clapet fermé empêche l'eau de couler.

Quels clapets faut-il fermer ?



Solution

La bonne réponse est qu'il faut fermer les trois clapets qui sont nommés D, F et H dans le dessin suivant.



C'est la seule possibilité permettant de couper l'eau aux moulins 2, 4 et 5 tout en continuer d'alimenter en eau les trois moulins 1, 3 et 6 :

- Les clapets A, G et I doivent rester ouverts, car sinon le moulin 1 n'est pas alimenté en eau.
- Les clapets B et E doivent également rester ouverts pour que le moulin 6 soit alimenté en eau.
- Comme les clapets B et E sont ouverts, le clapet H doit être fermé pour éviter que l'eau n'arrive au moulin 5.
- Comme le clapet A est ouvert, le clapet F doit être fermé pour éviter que l'eau n'arrive au moulin 2.
- Comme le clapet B est ouvert, le clapet D doit être fermé pour éviter que l'eau n'arrive au moulin 4.
- Comme les clapets D et F sont fermés, le clapet C doit être ouvert pour que le moulin 3 soit alimenté en eau.

C'est de l'informatique !

Dans cet exercice, l'écoulement de l'eau est régulé par des *conditions*. Par exemple, l'eau ne coule jusqu'au moulin 6 que si les deux clapets B et E sont ouverts. Voici un autre exemple un peu plus

complexe : l'eau ne coule jusqu'au moulin 3 que si au moins l'une des deux ou les deux conditions suivantes sont remplies :

- Le clapet A est ouvert et l'un des deux clapets C ou F est ouvert.
- Les deux clapets B et D sont ouverts.

De telles combinaisons de conditions sont obtenues à l'aide des *opérateurs logiques* ET (symbolisé par $\wedge$) ou OU (symbolisé par $\vee$). De tels opérateurs connectent des valeurs de vérité comme vrai et faux. Si A et B sont deux valeurs de vérité, on peut indiquer quelles valeurs de vérité les expressions « A ET B » et « A OU B » ont en fonction de A et B :

A	**B**	**A ET B**	**A OU B**
faux	faux	faux	faux
vrai	faux	faux	vrai
faux	vrai	faux	vrai
vrai	vrai	vrai	vrai

En informatique (et en mathématiques), l'expression « A OU B » est donc aussi considérée comme juste si A et B sont les deux justes. L'affirmation « le moulin 6 est alimenté » est équivalente à :

> « le clapet B est ouvert » ET « le clapet E est ouvert ».

Dans le deuxième exemple, l'affirmation « le moulin 3 est alimenté » est équivalente à :

> (« le clapet A et ouvert » ET (« le clapet C est ouvert » OU « le clapet F est ouvert ")) OU (« le clapet B est ouvert » ET « le clapet D est ouvert »).

En programmation, il est important de formuler les conditions de manière exacte. Chaque ET et chaque OU combine deux affirmations. Les parenthèses déterminent dans quel ordre les affirmations sont combinées. Les combinaisons à l'aide d'opérateurs logiques et de parenthèses sont utiles pour formuler des conditions complexes. Des conditions sont utilisées aussi bien pour des branchements avec `if` que pour des boucles `while` afin de guider le déroulement d'un programme.

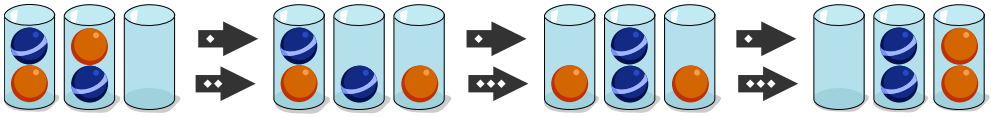
Mots clés et sites web

- Instruction conditionnelle : `https://fr.wikipedia.org/wiki/Instruction_conditionnelle_(programmation)`
- Variable booléenne : `https://fr.wikipedia.org/wiki/Booléen`
- Algèbre de Boole : `https://fr.wikipedia.org/wiki/Algèbre_de_Boole_(logique)`

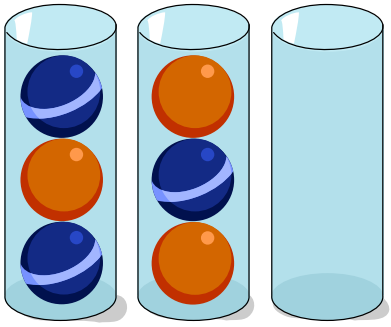
9. Jeu de balles

Les castors aimeraient trier des balles par couleur. À la fin, toutes les balles doivent se trouver dans deux verres. Toutes les balles qui se trouvent dans un verre doivent avoir la même couleur. Ils doivent suivre les règles suivantes :

- Règle 1 : Seule la balle la plus haute d'un verre peut être déplacée dans un autre verre.
- Règle 2 : Une balle peut toujours être mise dans un verre vide.
- Règle 3 : Une balle peut être mise dans un verre non vide uniquement s'il y reste de la place et que la balle en dessous a la même couleur que la balle déplacée.

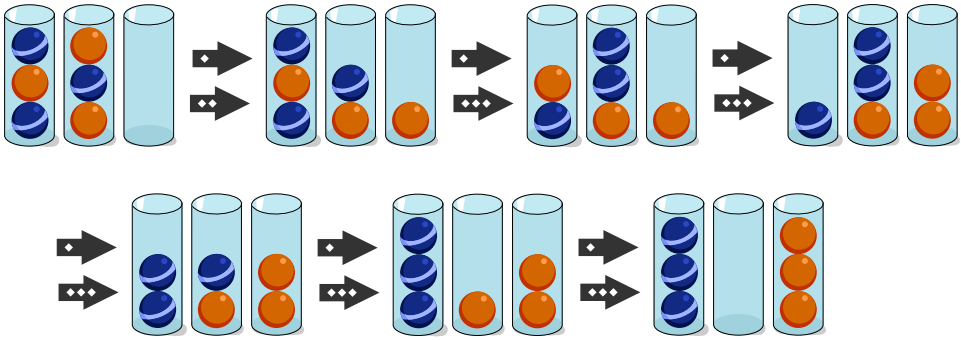


Trie les balles en les déplaçant d'après les trois règles.



Solution

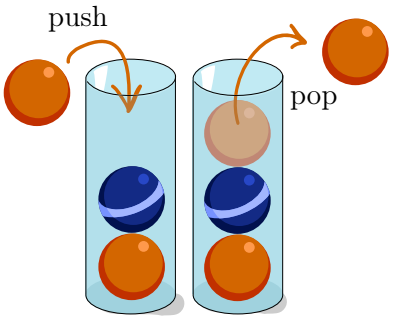
Les balles peuvent être déplacées dans l'ordre suivant :



Il faut au moins six étapes pour réarranger les balles. Il existe d'autres possibilités de réarranger les balles en seulement six étapes.

C'est de l'informatique !

Dans cet exercice, tu déplaces les balles comme un ordinateur gère les données enregistrées dans une *pile* : il ne peut qu'*empiler* (*push* en anglais) un élément en haut de la pile et *dépiler* (*pop* en anlgais) l'élément du haut de la pile.



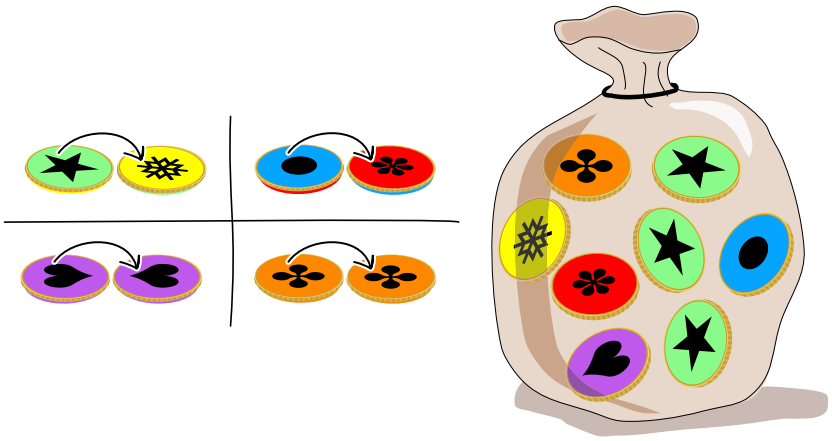
L'ordinateur ne peut accéder aux éléments du bas que si les balles du dessus ont d'abord été retirées. L'élément qui a été enregistré en dernier va être retiré en premier par l'ordinateur. Ce principe est appelé *dernier arrivé, premier sorti* en informatique.

Mots clés et sites web

- Pile : `https://fr.wikipedia.org/wiki/Pile_(informatique)`

10. Sac de pièces

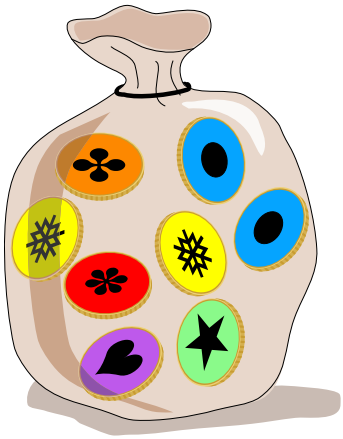
Il existe quatre sortes de pièces de monnaie différentes dans le pays d'Émile. Tu peux voir ici les deux côtés de ces pièces ainsi que le sac d'Émile avec ses pièces.



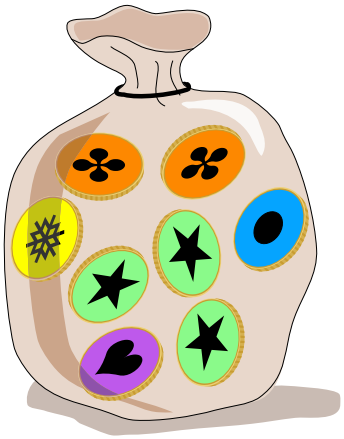
Émile secoue son sac de pièces.

Quel sac est celui d'Émile ?

A)



B)



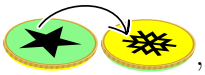
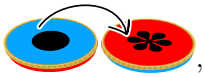
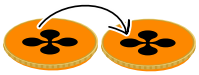
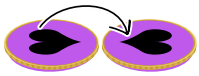
C)

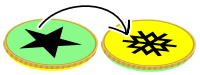
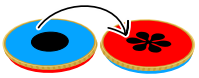
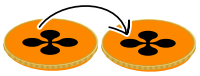
D)

Solution

La bonne réponse est C :

Dans le sac d'Émile, il y a :

- 4 pièces  ,
- 2 pièces  ,
- une pièce 
- et une pièce  .

	Sac d'Émile	Sac A	Sac B	**Sac C**	Sac D
	**4**	3	4	**4**	2
	**2**	3	1	**2**	4
	**1**	1	2	**1**	1
	**1**	1	1	**1**	1

Seul le sac C contient le même nombre de chaque sorte de pièces que le sac d'Émile. C'est donc la bonne solution.

C'est de l'informatique !

Dans cet exercice, il faut reconnaître les différentes sortes de pièces sans en voir les deux côtés. On n'a qu'une information partielle. Dans un système informatique, les objets du monde réel sont enregistrés avec leurs caractéristiques essentielles. Souvent, il suffit de connaître une partie de ces caractéristiques pour pouvoir reconnaître un objet. La caméra d'un véhicule autonome ne voit qu'une partie de la

réalité, mais doit quand même être en mesure de reconnaître des véhicules et usagers de la route et de réagir au traffic de manière adaptée. L'intelligence artificielle des systèmes informatiques apprend peu à peu et de mieux en mieux à reconnaître des objets à partir de fragments, comme les êtres humains le font.

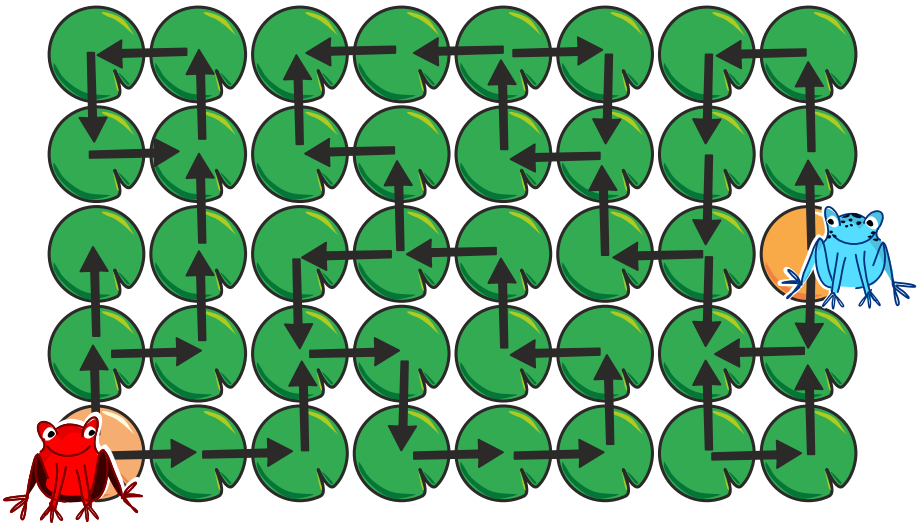
Mots clés et sites web

- Multiensemble : `https://fr.wikipedia.org/wiki/Multiensemble`
- Informations non structurées : `https://fr.wikipedia.org/wiki/Informations_non_structurées`

11. Rencontre

Sur un étang couvert de nénuphars, deux grenouilles peuvent se déplacer en sautant de feuille en feuille — mais seulement en suivant le sens des flèches.

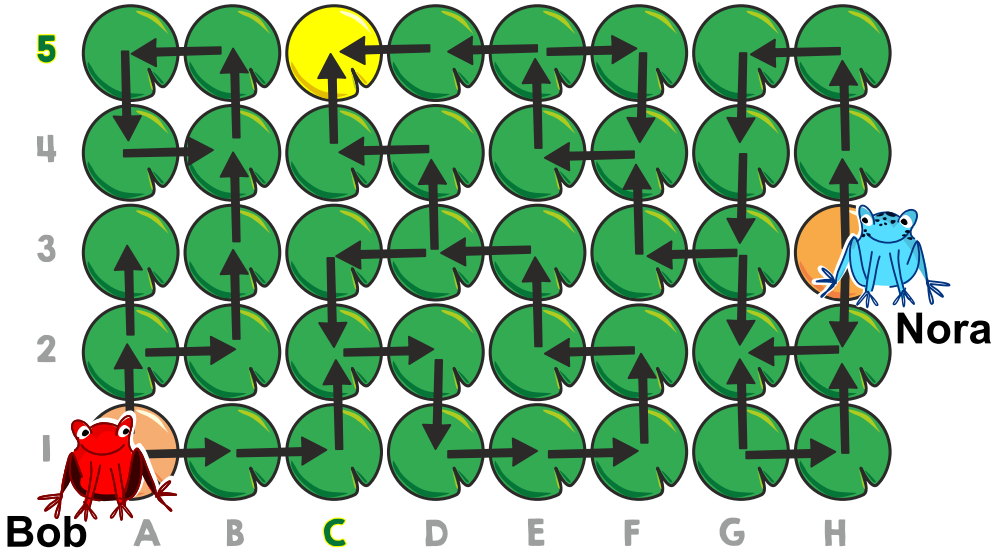
Sur quelle feuille les deux grenouilles peuvent-elles se rencontrer ?



Man kann auf die Blätter klicken. Klickt man auf ein Blatt, wird dieses ausgewählt und gleichzeitig ein bereits ausgewähltes Blatt wieder deaktiviert.

Solution

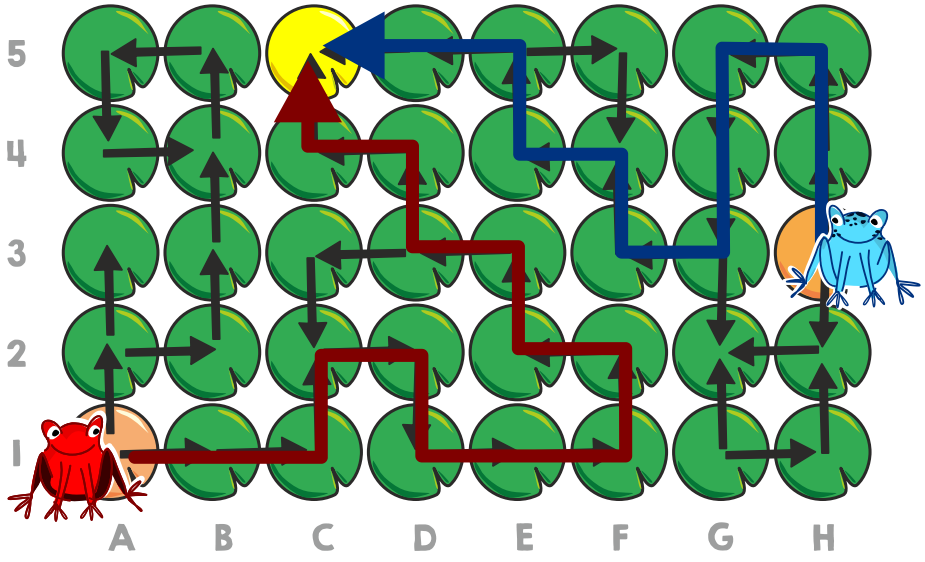
Les grenouilles ne peuvent se rencontrer que sur la feuille C5.



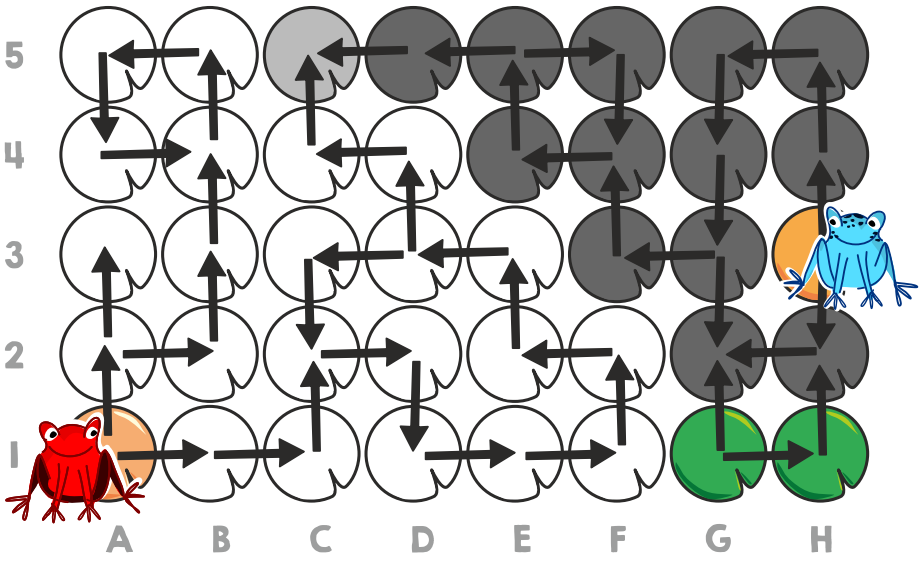
Bob, la grenouille rouge, a deux possibilités depuis sa feuille de départ : si elle va vers le « haut », elle arrive soit dans le cul de sac sur A3 ou elle reste bloquée dans le cercle qui commence sur B4. Si elle va vers la « droite » (en direction de B1), elle peut sauter jusqu'à D3. Depuis D3, elle peut soit tourner en rond en partant vers la « gauche », soit aller vers le « haut » jusqu'à un autre cul de sac sur C5.

Nora, la grenouille bleue, a aussi le choix entre deux directions depuis sa feuille de départ. Si elle va vers le « bas », elle arrive dans le cul de sac sur G2. Si elle va vers le « haut », elle peut sauter jusqu'à G3. Depuis là, elle peut soit arriver dans le cul de sac sur G2 ou partir à « gauche » et atteindre E5. Depuis E5, elle peut soit tourner en rond, soit partir à « gauche » et arriver dans le cul de sac sur C5.

Nous savons que Bob peut aussi atteindre C5, elles peuvent donc s'y rencontrer. Le dessin montre les chemins qu'elles suivent pour y arriver.



Cela ne garantit pas encore qu'elle ne puissent pas aussi se rencontrer ailleurs. Le dessin suivant montre toutes le feuilles que Bob (en blanc) en Nora (en gris foncé) peuvent atteindre en suivant les flèches de toutes les manières possibles. On voit que seule la feuille C5 peut être atteinte par les deux grenouilles.



C'est de l'informatique !

Comment peut-on générer la dernière image ? Les feuilles pouvant être atteintes par une grenouille peuvent être trouvées à l'aide d'un *parcours en largeur* ou d'un *parcours en profondeur*. Ce sont deux des méthodes les plus importantes en informatique. Grâce à elles, on peut déterminer quelles sont les feuilles blanches et les feuilles gris foncé. Il ne reste ensuite plus qu'à trouver les feuilles pouvant être atteintes par les deux grenouilles.

Mots clés et sites web

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

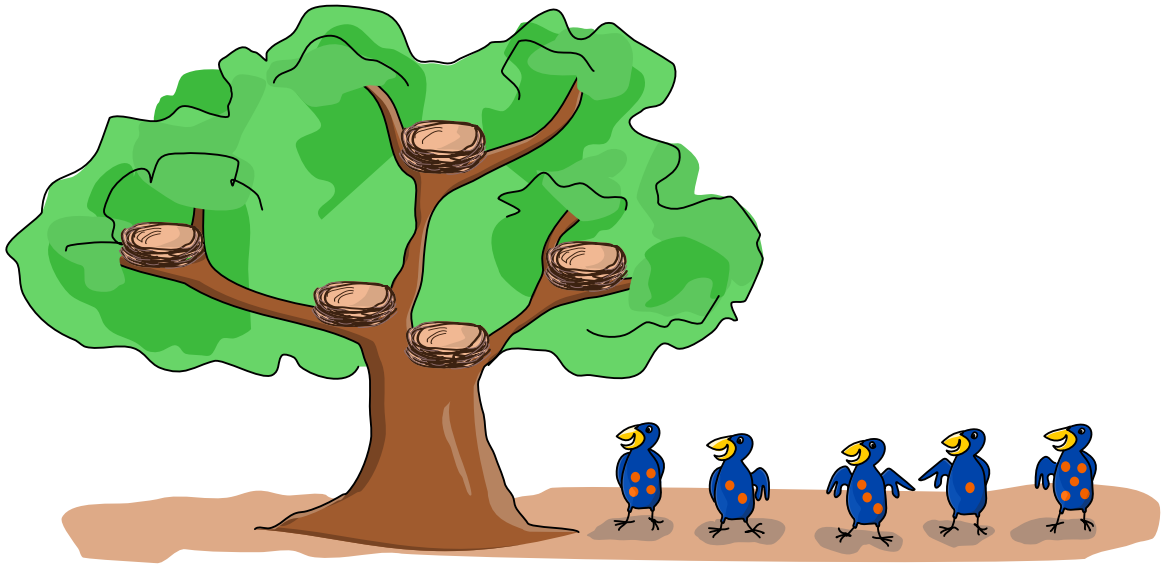
12. Emménagement

Les tachetés sont des oiseaux qui ont des points sur leurs plumes. Cinq tachetés sont à côté d'un arbre. Ils grimpent dans l'arbre l'un après l'autre — de gauche à droite — et emménagent dans les nids vides. Le tacheté avec quatre points commence. Chaque tacheté procède comme suit :

Il commence en bas de l'arbre. Il effectue les étapes suivantes jusqu'à ce qu'il ait trouvé un nid vide :

1. Il grimpe jusqu'à ce qu'il trouve un nid.
2. Si le nid est vide, il y emménage et reste là.
3. Sinon, il continue à grimper :
   - vers la gauche si le tacheté dans le nid a plus de points que lui ;
   - vers la droite si le tacheté dans le nid a le même nombre ou moins de points que lui.

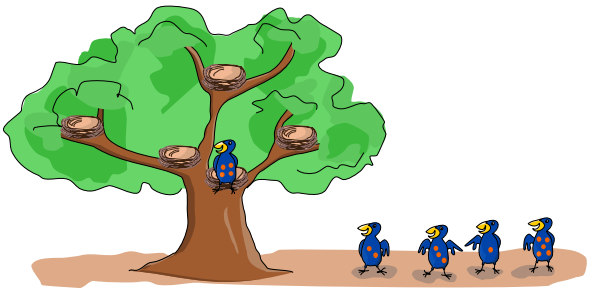
Où se trouvent les tachetés à la fin ? Place chaque tacheté dans le bon nid.



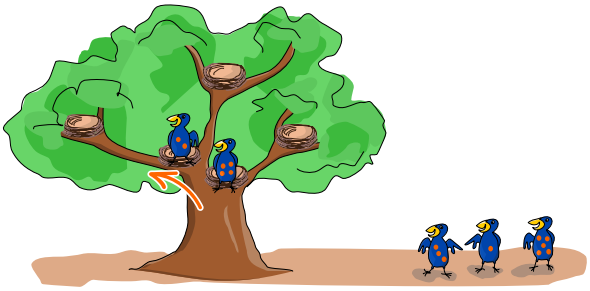
Solution

On obtient la bonne solution de la manière suivante :

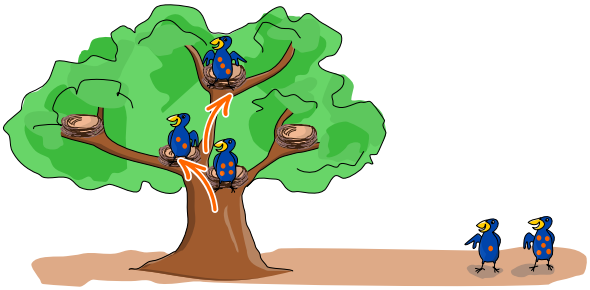
Le premier tacheté, celui à 4 points, arrive dans le nid du bas et y emménage.



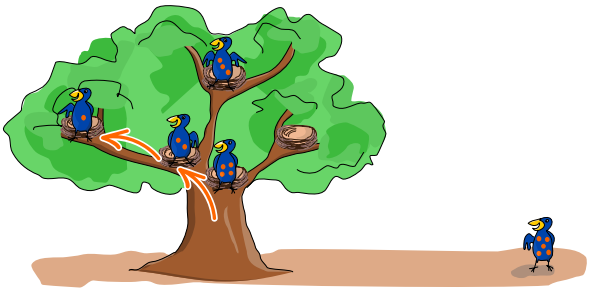
Le deuxième tacheté a 2 points. Le nid du bas est maintenant occupé par le premier tacheté à 4 points. Comme 4 est plus grand que 2, le deuxième tacheté continue à grimper vers la gauche et emménage dans le premier nid vide.



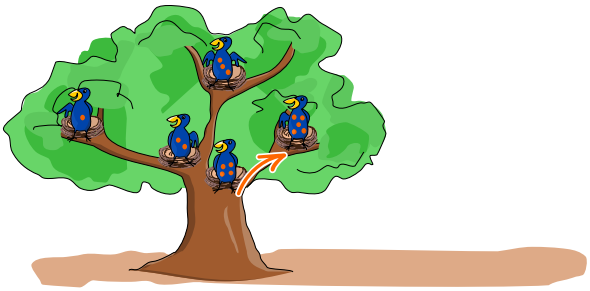
Le troisième tacheté a 3 points. Il grimpe vers la gauche après le nid du bas dans lequel est le tacheté à 4 points, car 4 est plus grand que 3. Le tacheté à 2 points est dans le nid suivant. Comme 3 est plus grand que 2, le troisième tacheté continue de grimper vers la droite. Il emménage dans le prochain nid vide. C'est le nid le plus haut.



Le quatrième tacheté a 1 point. Comme tous les autres tachetés ont plus de points que lui, il grimpe vers la gauche après chaque nid occupé. Il arrive ainsi dans le nid tout à gauche et y emménage.



Le dernier tacheté a 5 points. Comme aucun tacheté n'a plus de points que lui, il grimpe vers la droite après chaque nid occupé. Il fait cela une fois après le nid du bas et emménage ensuite dans le nid tout à droite.



C'est de l'informatique !

La manière dont les tachetés choisissent leur nid a un avantage intéressant : on peut vite trouver un tacheté particulier. Si le tacheté que tu cherches a moins de points que celui que tu es en train

de regarder, tu dois continuer à chercher à sa gauche. Sinon, tu continue à chercher à sa droite. A chaque évaluation d'un oiseau, tu peux ainsi réduire le domaine de recherche à une de deux moitiés. C'est pour cela que tu vas rapidement trouver ton tacheté.

Il y a beaucoup de manières d'organiser des données ; on parle de différentes *structures de données*. La structure de donnée utilisée dans cet exercice est un *arbre binaire de recherche*. Le mot « binaire » vient du mot latin *bis* qui veut dire « deux fois ». En effet, il y a au maximum deux branches plus petites qui partent d'une branche (là où se trouve un nid dans cet exercice). Les arbres binaires de recherche sont utilisés en programmation lorsque beaucoup de données doivent être trouvées rapidement. Ils sont en général beaucoup plus grands que le petit arbre de l'exercice. Il y a encore une différence supplémentaire : l'arbre de cet erercice acceuille un nombre fixe de cinq tachetés, alors que l'on peut en général toujours ajouter des données à un arbre binaire de recherche. On ajoute pour cela simplement une nouvelle branche au bout d'une branche existante, ce qui agrandit l'arbre. Les stuctures de données pouvant être modifiées de cette façon s'appellent *structures de données dynamiques*.

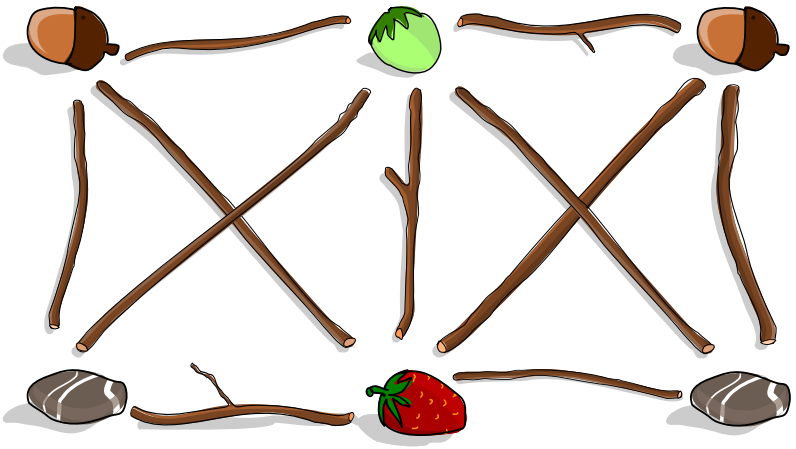
Mots clés et sites web

- Arbre binaire de recherche :
  `https://fr.wikipedia.org/wiki/Arbre_binaire_de_recherche`
- Structure de données : `https://fr.wikipedia.org/wiki/Structure_de_données`

13. Voleur de fraise

Anja veut créer une œuvre d'art dans le jardin et a ramassé pour cela différents objets : plusieurs glands, noisettes et cailloux ainsi qu'une fraise. Elle met quelques objets dans l'herbe.

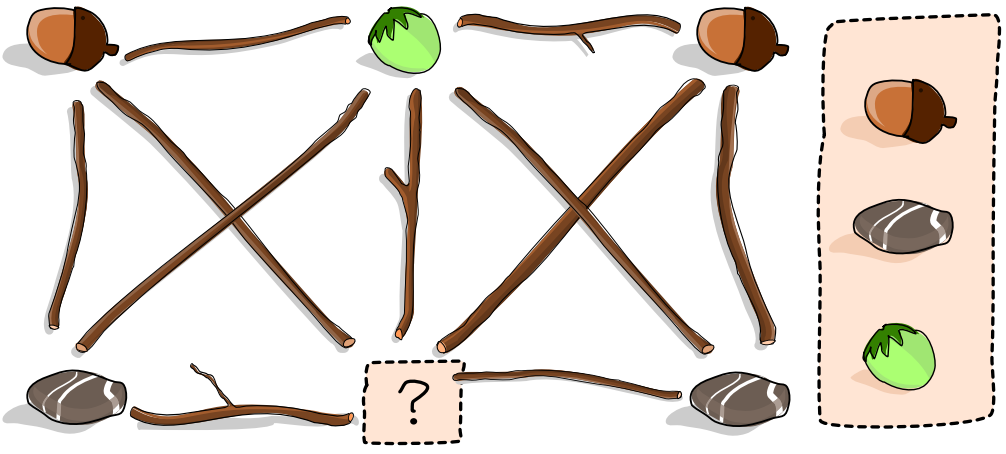
Ensuite, Anja dispose des branches entre ces objets. Elle suit pour cela la règle suivante : une branche ne doit pas se trouver entre deux objets pareils, par exemple entre deux glands. Voici l'œuvre d'art terminée :



Le frère d'Anja vient et mange la fraise pendant qu'elle n'est pas là.

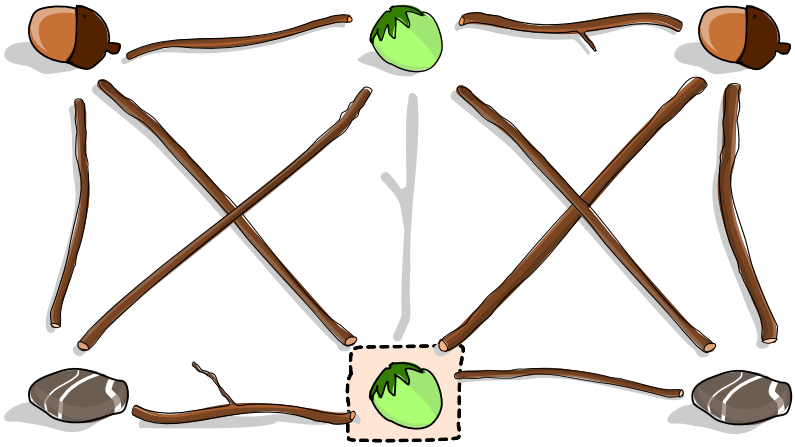
Peux-tu aider le frère d'Anja à dissimuler son méfait ?

Place un autre objet à la place de la fraise et enlève exactement une branche. L'œuvre d'art modifiée doit respecter la règle d'Anja.



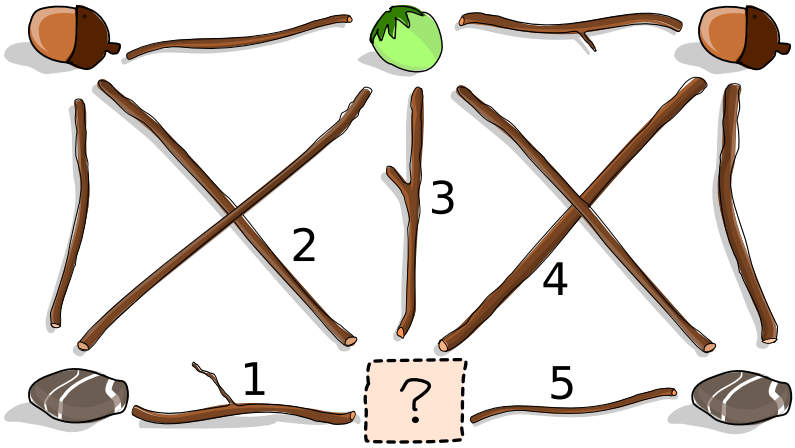
Solution

Lorsque l'on remplace la fraise par une noisette, la branche 3 ne respecte plus la règle d'Anja : elle se trouve entre deux objets pareils, à savoir deux noisettes. Cette branche doit donc être enlevée.



Les deux autres remplacements possibles nécessitent d'enlever plus d'une branche :

- Si la fraise est remplacée par un gland, il faut enlever les branches 2 et 4.
- Si la fraise est remplacée par un caillou, il faut enlever les branches 1 et 5.



C'est de l'informatique !

L'œuvre d'Anja peut être représentée par un *graphe*. Un graphe est composé de *nœuds* (les emplacements des objets) et d'*arêtes* (les branches) qui relient deux objets chacune. Les graphes sont des outils polyvalents et sont souvent utilisés lors de la modélisation de problèmes en informatique.

Lorsque deux nœuds sont directement reliés par une arête, ils sont *voisins* l'un de l'autre. Un groupe de nœuds dans lequel chaque nœud est voisin de chaque autre nœuds est appelé une *clique*. Nous avons deux cliques de quatre nœuds dans notre graphe : la moitié gauche et la moitié droite du graphe (la noisette en haut et le point d'interrogation en bas appartiennent aux deux cliques).

La règle d'Anja implique que tous les nœuds d'une clique doivent être occupés par des objets différents. Pour respecter la règle, nous avons donc besoin d'autant d'objets différents qu'il y a de nœuds dans une clique. Nous n'avons cependant plus que trois objets différents une fois que la fraise a été enlevée. Il ne peut donc rester que des cliques comportant au maximum 3 nœuds si la règle doit être respectée. Il faut donc enlever une arête (une branche) afin de détruire les deux cliques à quatre nœuds.

La règle d'Anja correspond à une règle du problème de *coloration de graphe* : on assigne une couleur à chaque nœud d'un graphe, et les nœuds voisins doivent être de couleurs différentes (les couleurs correspondent ici aux différents types d'objets). Le but est souvent d'utiliser le moins de couleurs possibles. Le problème consistant à colorer un graphe avec le moins de couleurs possibles a beaucoup d'applications. Quelques exemple sont la planification d'une compétition, l'élaboration d'un plan de table et même la résolution d'un sudoku.

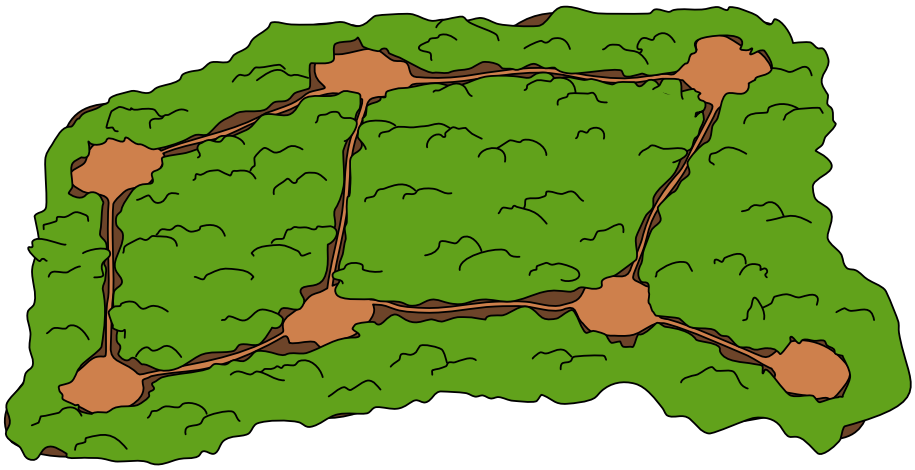
Mots clés et sites web

- Coloration de graphe : `https://fr.wikipedia.org/wiki/Coloration_de_graphe`
- Coloration des arêtes d'un graphe : `https://fr.wikipedia.org/wiki/Coloration_des_arêtes_d'un_graphe`
- Clique : `https://fr.wikipedia.org/wiki/Clique_(théorie_des_graphes)`

14. Gardes forestiers

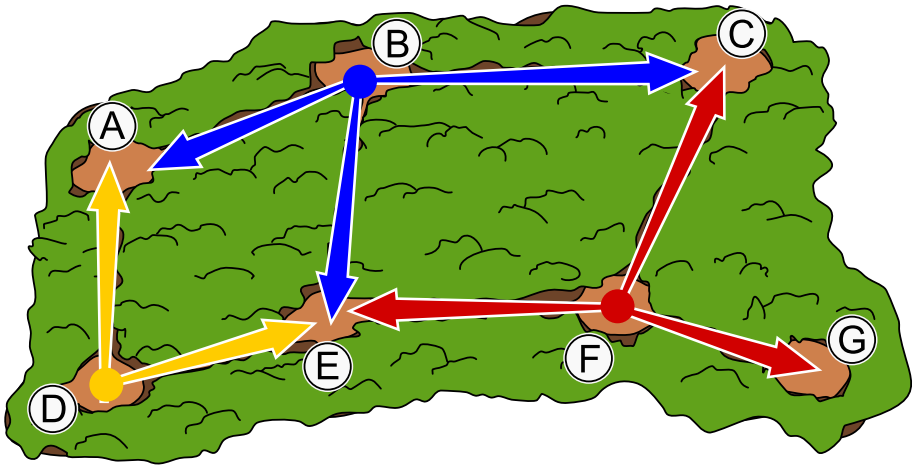
Les gardes forestiers veulent observer les animaux sur les sentiers de la forêt. Depuis chaque clairière, ils peuvent voir tous les sentiers allant de cette clairière à une clairière suivante. Il doit y avoir aussi peu de gardes forestiers que possible qui surveillent les sentiers.

Choisis des clairières afin que tous les sentiers puissent être surveillés par aussi peu de gardes forestiers que possible



Solution

L'image montre la solution minimale permettant aux gardes forestiers de surveiller tous les sentiers à partir de trois clairières.



Il y a huit sentiers à surveiller. Si seuls deux gardes forestiers suffisaient pour surveiller tous les sentiers, il devrait y avoir une clairière de laquelle partent au moins quatre sentiers, mais il n'y a pas de telle clairière dans cette forêt. Deux gardes forestiers ne suffisent donc pas.

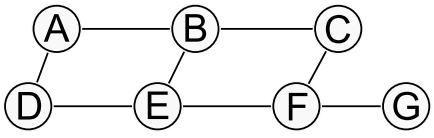
Il faut donc au minimum trois gardes forestiers pour assurer la surveillance de tous les sentiers. La solution présentée ici a donc le plus petit nombre de gardes forestiers possible. Il n'y a pas d'autre solution avec exactement trois gardes forestiers.

Nous pouvons déduire du nombre de sentiers et du fait qu'il n'y a aucune clairière de laquelle plus de trois sentiers partent que chaque garde forestier doit surveiller au moins deux sentiers qu'aucun autre garde forestier ne surveille.

Un garde forestier doit être placé sur la clairière F afin de surveiller le sentier entre les clairières F et G. Pour surveiller le sentier entre les clairières B et C, le deuxième garde forestier doit observer la forêt depuis la clairière B. Le dernier garde forestier doit être dans la clairière D pour que les deux derniers sentiers puissent être surveillés par un seul garde forestier. On obtient ainsi la solution donnée ici et il n'en existe pas d'autre.

C'est de l'informatique !

Les relations entre des choses (par exemple des sentiers entre des clairières) peuvent être représentées par ce que l'on appelle un *graphe*. Un graphe est constitué de *nœuds* (ici, les clairières) et d'*arêtes* (ici, les sentiers) représentées par des lignes reliant les nœuds. Le graphe de cet exercice ressemble à ceci :



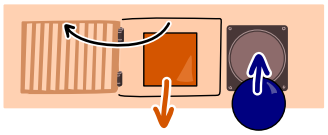
Dans cet exercice du castor, il faut trouver le plus petit nombre de nœuds afin que chacune des arêtes commence ou finisse sur l'un des ces nœuds. Les informaticien·nes appelent un tel ensemble de nœuds une *couverture par sommets* ou une *transversale* (engl. *minimal vortex cover*). Dans la vie quotidienne, on recontre de tels problèmes de couverture par sommets lorsque l'on cherche les meilleurs emplacements pour des lampadaires ou pour des caméras de surveillance, par exemple.

Mots clés et sites web

- Graphe : `https://fr.wikipedia.org/wiki/Graphe_(mathématiques_discrètes)`
- Couverture par sommets : `https://fr.wikipedia.org/wiki/Problème_de_couverture_par_sommets`

15. Casse-tête d'anniversaire

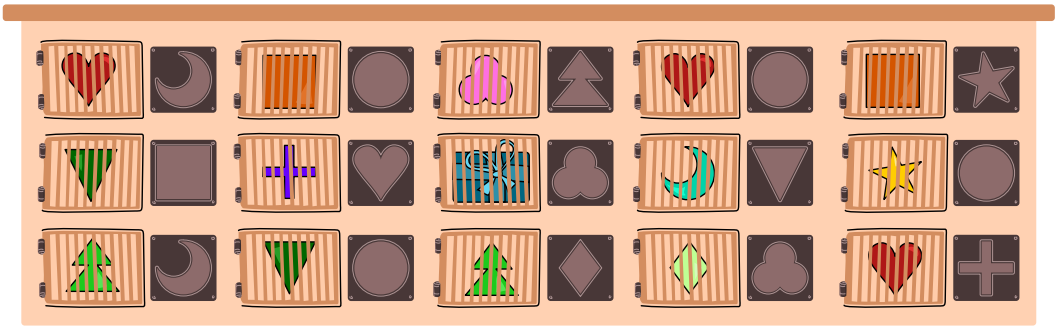
Pour son anniversaire, Bastien reçoit une boîte avec 15 portes. Il y a un autre cadeau derrière la porte du milieu et des plots de différentes formes derrière les autres portes. Chaque porte est associée à un trou qui se trouve à sa droite. Bastien peut ouvrir une porte en mettant un plot de la même forme dans le trou — comme une clé dans une serrure.



Au début, Bastien a ce plot rond : 

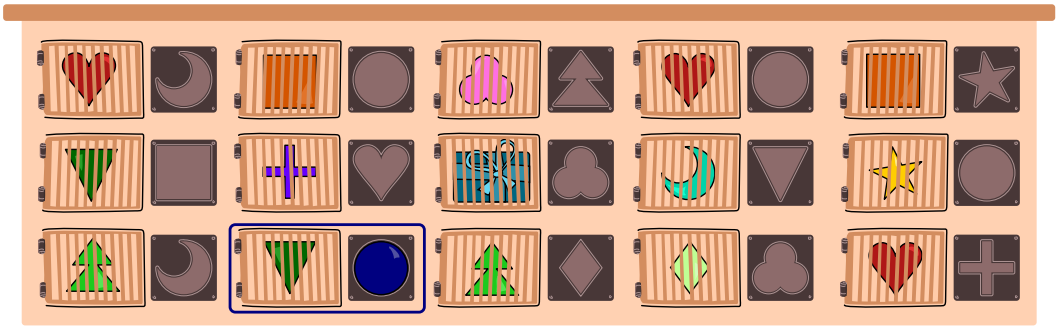
Il veut ouvrir au maximum cinq portes pour obtenir le cadeau.

Quelle porte Bastien doit-il ouvrir en premier ?

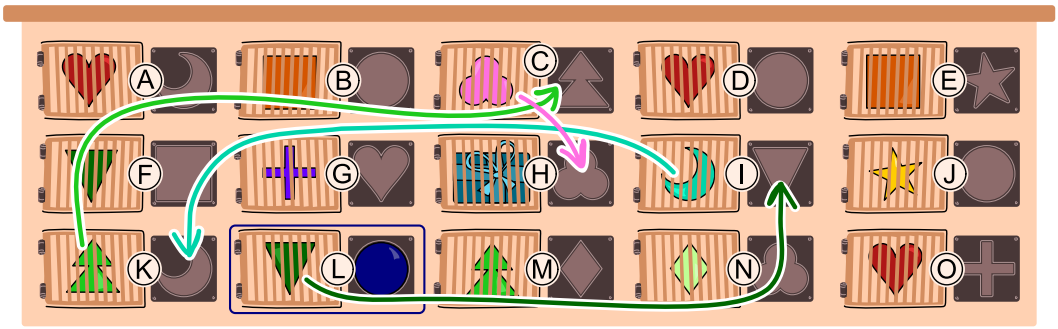


Solution

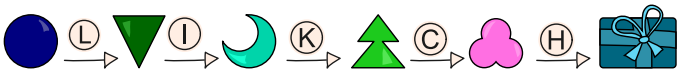
Bastien doit commencer par ouvrir la porte encadrée en bleu :



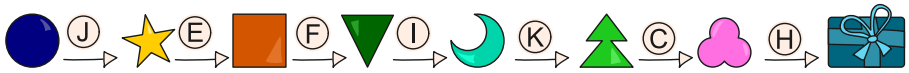
Dans l'illustration ci-dessous, chaque porte correspond a une lettre et les flèches montrent comment Bastien obtient le cadeau en n'ouvrant que cinq portes au maximum.



On peut également représenter l'ordre dans lequel Bastien ouvre les cinq portes de la façon suivante :



Il y a aussi d'autres chemins menant au cadeau, par exemple celui-ci :

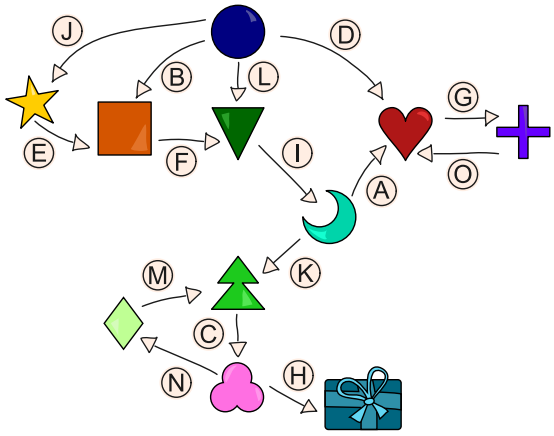


Mais ces chemins sont tous trop longs : il faut ouvrir plus de cinq portes. C'est un gros travail que d'essayer toutes les possibilités.

Dans notre cas, on trouve le chemin le plus court et donc la bonne solution en faisant ce que l'on appelle une *recherche inversée* : on commence par la porte derrière laquelle se trouve le cadeau et on regarde ensuite de quel plot on a besoin.

C'est de l'informatique !

Avec un peu plus de temps et de travail, on peut également représenter la situation de cet exercice sous forme de *graphe* :



Un graphe est composé de *nœuds* (cercles) et *arêtes* (lignes) entre les nœuds. Ici, nous avons un nœud pour chaque forme et pour le cadeau. Les arêtes sont dans notre cas des flèches (aussi appelées arêtes *orientées*) qui correspondent aux portes. Chaque arête mène de la forme qui ouvre une porte à la forme se trouvant derrière la porte.

En informatique, on travaille volontiers avec des graphes. D'un côté, ils permettent souvent de représenter des relations abstraites de manière claire. D'un autre côté, il existe des algorithmes pour répondre à des questions liées aux graphes de manière efficace. Le travail nécessaire à l'élaboration du graphe en vaut vite la peine pour des problèmes compliqués.

Dans cet exercice, nous cherchons un chemin de longueur 5 au maximum entre le plot reçu  et le cadeau  . Un bon algorithme pour cela est le *parcours en largeur*. Il fonctionne aussi bien pour les graphes avec des arêtes orientées comme dans l'exercice que pour les graphes non orientés.

Mots clés et sites web

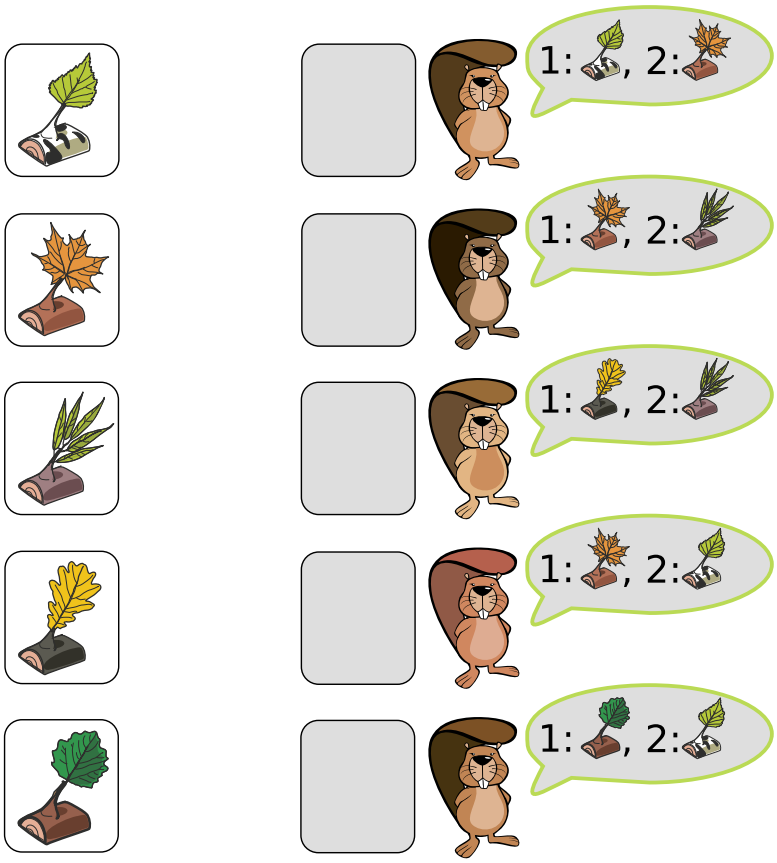
- Graphe orienté : `https://fr.wikipedia.org/wiki/Graphe_orienté`
- Algorithme de parcours en largeur : `https://fr.wikipedia.org/wiki/Algorithme_de_parcours_en_largeur`

16. Lieblingsgeschenk

La famille castor a cinq cadeaux pour ses cinq enfants. Chaque enfant indique d'abord son cadeau favori, puis son second choix. Les cadeaux doivent être bien distribués :

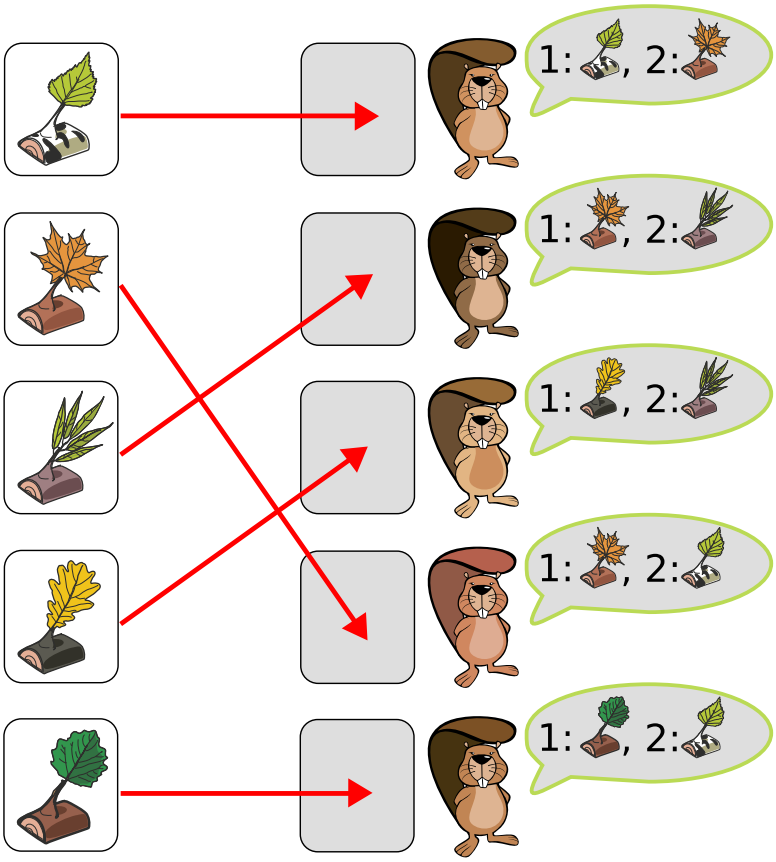
1. Le plus d'enfants possible doivent recevoir leur cadeau favori.
2. Les autres enfants doivent recevoir leur second choix.

Donne les bons cadeaux aux enfants.



Solution

Voici la seule manière de distribuer les cadeaux en respectant les deux conditions.



La distribution du graphique ci-dessus donne leur cadeau favori à quatre castors et son deuxième choix à un castor. Tous les castors ne peuvent pas recevoir leur cadeau favori, car deux castors ont le même. Il n'y a donc pas d'autre distribution donnant leur cadeau favori à plus de castors. Tu peux remarquer que si tu commences la distribution par le haut et donnes son cadeau favori au deuxième castor, le quatrième castor ne pourra avoir aucun de ses deux cadeaux préférés. Il ne suffit donc pas de donner à chaque castor le meilleur cadeau disponible dans cet exercice.

Une méthode de résolution consiste à distribuer d'abord tous les cadeaux qui ne sont le favori que d'un seul castor. Il ne reste ensuite que deux castors avec le même cadeau favori. On regarde ensuite lequel des deux deuxièmes choix est disponible, et on donne à l'autre castor son cadeau favori.

C'est de l'informatique !

Dans cet exercice, nous avons affaire à un *problème d'affectation* univoque : nous voulons affecter les cadeaux de manière à ce que tous les enfants recoivent un cadeau. Les enfants n'ont ici pas qu'un seul souhait, mais une liste de préférence. De tels problèmes d'affectation avec listes de préférence peuvent devenir très compliqués. L'informatique nous aide à résoudre de tels problèmes rapidement.

Une possibilité est de donner une valeur aux affectations : le cadeau favori a la valeur 1 et le deuxième choix la valeur 2. Un *couplage* (*matching* en anglais) est optimal s'il n'existe pas d'autre couplage avec plus de premiers choix distribués. Un tel couplage est appelé *couplage parfait de poids minimum*. Il existe beaucoup de problèmes d'affectation. L'un deux est appelé *problème des marriages stables* (*Stable Marriage Problem* en anglais). Cela t'intéresse ? Alors tu devrais étudier l'informatique !

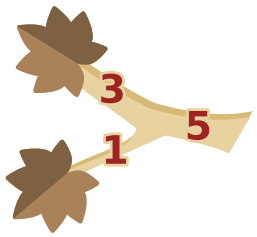
Mots clés et sites web

- Problème d'affectation : `https://fr.wikipedia.org/wiki/Problème_d'affectation`
- Couplage : `https://fr.wikipedia.org/wiki/Couplage_(théorie_des_graphes)`

17. Sauvetage d'arbre

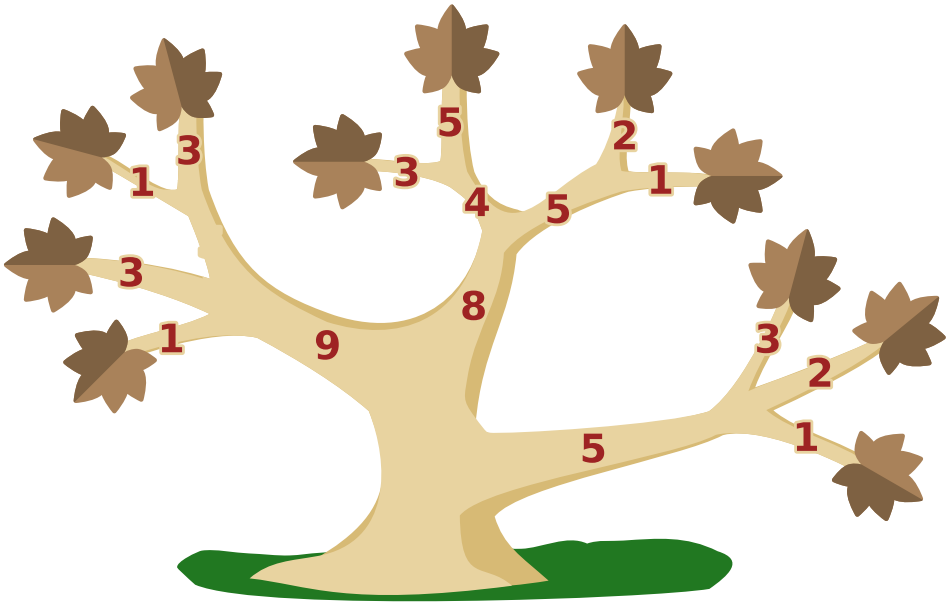
Un des arbres du jardin de Bruno est malade : toutes ses feuilles sont sèches. Bruno veut le sauver. Pour cela, il doit scier certaines branches de façon à enlever toutes les feuilles. De nouvelles branches avec de nouvelles feuilles peuvent ensuite pousser.

Bruno aimerait terminer le plus vite possible. L'image montre un exemple :



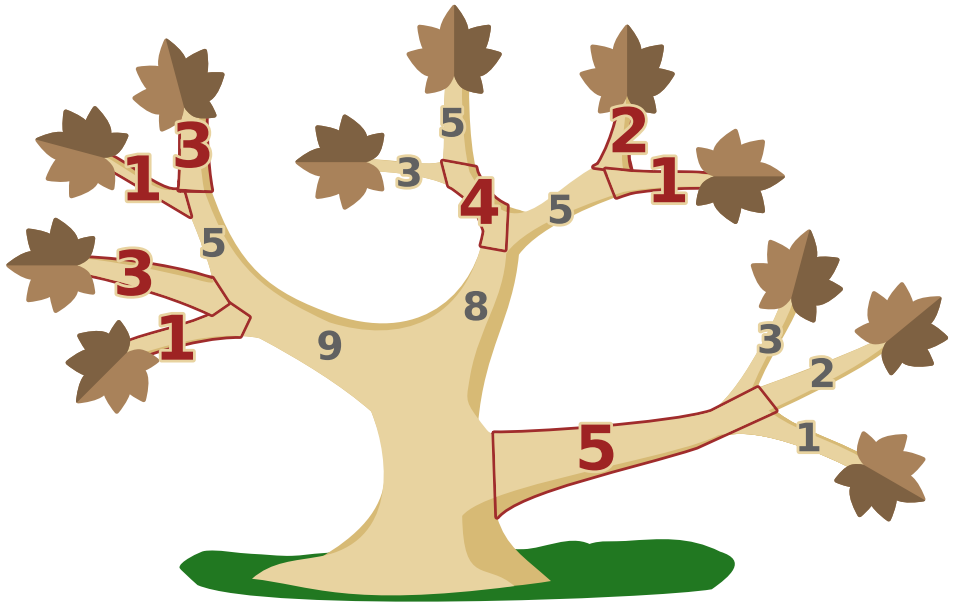
Pour enlever les deux feuilles, Bruno peut soit scier les deux branches portant les feuilles, soit scier la branche de laquelle partent les deux autres branches. Les chiffres sur chaque branche indiquent combien de temps est nécessaire pour la scier. Bruno va donc scier les deux branches avec des feuilles, étant donnée que $3 + 1 < 5$. Tu peux voir l'arbre complet ci-dessous.

Quelles branches Bruno va-t-il scier pour terminer le plus vite possible ?



Solution

La bonne solution est la suivante : Bruno scie les branches encadrées en rouge pour terminer le plus vite possible.



Mais pourquoi est-ce la bonne solution ? On peut commencer par calculer de combien de temps Bruno a besoin pour scier uniquement les branches portant des feuilles — il aurait ainsi terminé :

$$1 + 3 + 1 + 3 + 3 + 5 + 2 + 1 + 3 + 2 + 1 = 25$$

Maintenant, on va en direction du tronc et vérifie à chaque étape si ce serait plus rapide de scier la branche à laquelle les branches précédentes sont directement ou indirectement reliées. On dérive le calcul suivant de la première de ces étapes (la fonction « min » calcule le minimum de ses arguments) :

$$\begin{aligned} & 1 + 3 + \min(5, 1 + 3) + \min(4, 3 + 5) + \min(5, 2 + 1) + \min(5, 3 + 2 + 1) \\ = \; & 1 + 3 + (1 + 3) + 4 + (2 + 1) + 5 \\ = \; & 20 \end{aligned}$$

On ne calcule pas tout de suite le temps total afin de mieux voir quelles branches doivent être sciées. L'étape suivant nous mène déjà au tronc :

$$\begin{aligned} & \min(9, 1 + 3 + 1 + 3) + \min(8, 4 + 2 + 1) + 5 \\ = \; & (1 + 3 + 1 + 3) + (4 + 2 + 1) + 5 \\ = \; & 20 \end{aligned}$$

Bruno ne peut pas terminer plus rapidement.

C'est de l'informatique !

Imaginons que les branches sciées ne tombent pas tout de suite de l'arbre — comme lorsque l'on résout cet exercice à l'écran. On pourrait alors dire que l'arbre est séparé en deux parties lorsque l'on scie : une partie comporte tous les morceaux sciés, et surtout toutes les feuilles, et l'autre partie comporte le tronc et les branches qui n'ont pas été sciées. Cette séparation ou *coupe* de l'arbre est minimale par rapport au temps nécessaire à Bruno pour enlever toutes les feuilles.

L'informatique traite aussi d'arbres, qui y sont utilisés pour réprésenter des objets reliés entre eux de manière spécifique. Les objets sont appelés *nœuds* et les liens entre eux *arêtes*. Il n'existe toujours qu'un seul chemin entre deux nœuds le long des arêtes — comme il n'existe qu'un seul chemin entre une feuille ou un branchement jusqu'au tronc le long des branches. Si l'on renonce à cette contrainte, on parle plus généralement d'un *graphe*.

Pour les graphes généraux, ce n'est pas si facile de calculer la *coupe minimale*, c'est à dire la séparation en deux ou plusieurs parties à un coût minimal, que pour un arbre comme nous l'avons fait ici, mais pas trop compliqué non plus. Heureusement, car il existe des applications intéressantes. Les coupes minimales peuvent être utilisées pour diviser des images en parties similaires. Dans les *réseaux de flot*, un type de graphe spécial avec lequel on peut, entre autres, modéliser le flot des données dans des réseaux, le coût d'une coupe minimale correspond au flot maximal dans le réseau complet.

Mots clés et sites web

- Arbre : `https://fr.wikipedia.org/wiki/Arbre_(théorie_des_graphes)`
- Coupe : `https://fr.wikipedia.org/wiki/Coupe_(théorie_des_graphes)`
- Réseau de flot : `https://fr.wikipedia.org/wiki/Réseau_de_flot`
- Théorème flot-max/coupe-min : `https://fr.wikipedia.org/wiki/Théorème_flot-max/coupe-min`

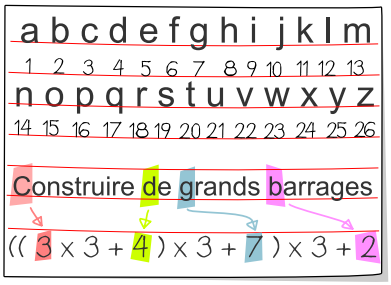
18. Bibliothèque

Susi est à la bibliothèque des castors avec Tim. Ils veulent emprunter un livre appelé « Construire de grands barrages ».

Tim va vers l'étagère 1, regarde dans la rangée 4 et sort le livre du casier 0. Susi est impressionnée. Tim explique à Susi comment on détermine l'emplacement d'un livre :

On prend la première lettre de chaque mot du titre du livre et détermine sa position dans l'alphabet. On additionne ensuite ces positions après avoir multiplié par 3 la valeur obtenue à l'addition précédente.

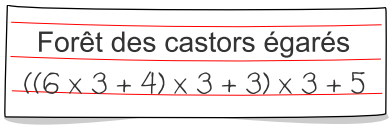
Le livre désiré donne 140. L'emplacement du livre est ainsi tout de suite clair.



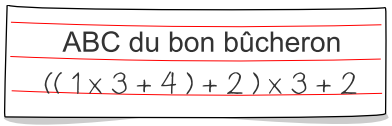
Susi écrit maintenant les calculs correspondant à ses livres préférés, mais elle a fait une erreur pour l'un d'entre eux.

Lequel ?

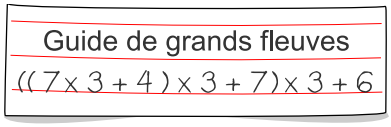
A)



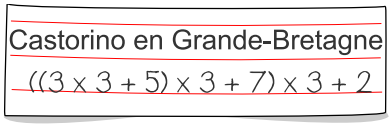
B)



C)

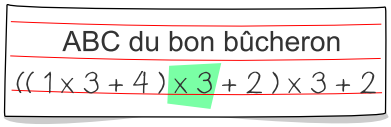


D)



Solution

Susi a presque tout fait juste : elle a toujours additionné les bonnes positions et a multiplié les résultats intermédiaires par 3 — avec une exception : elle a oublié une multiplication dans la réponse B.



C'est de l'informatique !

Le système d'expressions correspondant à l'emplacement des livres permet aux visiteurs de la bibliothèque de déterminer l'endroit exact où les livres sont rangés. Comme ça, personne ne doit chercher longtemps. La bibliothèque et les visiteurs doivent cependant faire attention à une chose : différents livres peuvent avoir la même expression et donc le même résultat. Par exemple, les livres « Guide des grands fleuves » et « Guide des grandes familles » sont dans le même casier. Les casiers doivent donc être assez grands ou pouvoir être agrandis selon les besoins.

C'est aussi une bonne idée que l'endroit auxquel les données sont enregistrées dans un ordinateur puisse être calculé directement à partir des données elles-même. Pour cela, des *fonctions de hachage* ont été développées en en informatique : des fonctions mathématiques qui calculent une valeur à partir du contenu des données ou d'une partie des données, valeur qui indique directement l'emplacement mémoire — comme dans cet exercice du castor. De bonnes fonctions de hachage minimisent le nombre de fois où la même valeur est calculée. Si un tel conflit a lieu, l'informatique dispose de bonnes méthodes pour le gérer.

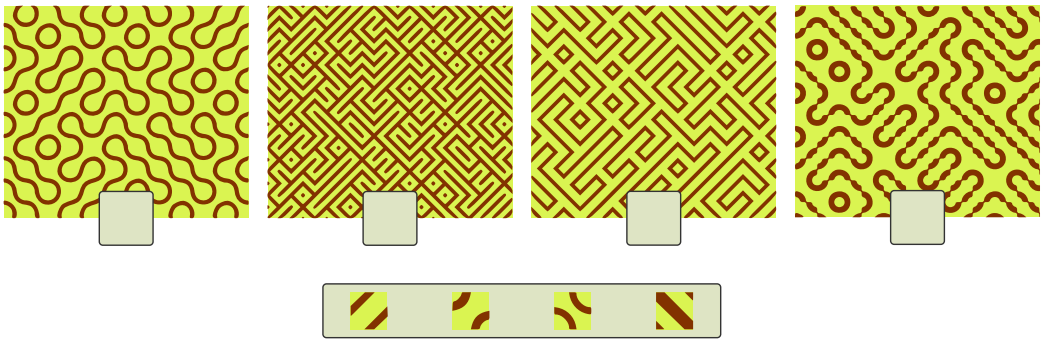
Mots clés et sites web

- Fonction de hachage : `https://fr.wikipedia.org/wiki/Fonction_de_hachage`
- Table de hachage : `https://fr.wikipedia.org/wiki/Table_de_hachage`

19. Pavage de Truchet

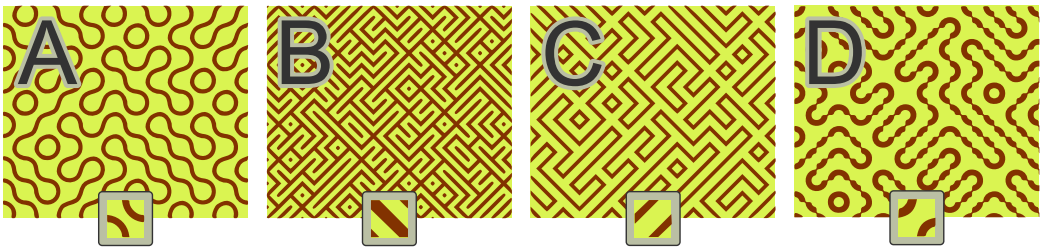
Les motifs suivants ont été créés en n'utilisant qu'un seul type de pavé. Les images des pavés individuels sont agrandies.

Assigne chaque pavé au motif correspondant.

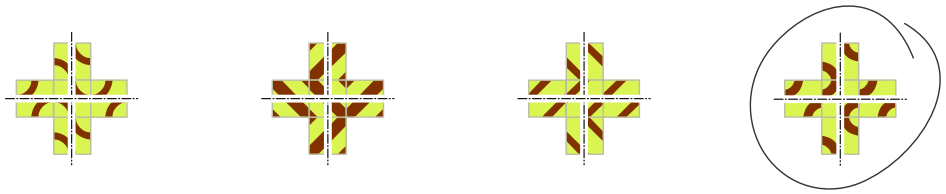


Solution

Voici la bonne attribution :



En mettant cinq mêmes pavés côte à côte et en comparant les différents pavés, on voit de claires différences :



Le pavé  est le seul pavé dont les quatre côtés ne vont pas exatement ensemble. C'est le seul moyen pour obtenir des lignes de largueurs différentes comme dans le motif D. Le pavé  est le seul avec lequel on peut créer des points carrés comme sur le motif B, en mettant bout à bout quatre coins avec un triangle. De plus, il a la plus grande proportion de brun par rapport au jaune, comme le motif B. Il ne reste donc que le pavé  pour former le motif A aux formes arrondies, alors que les lignes droites du motif C ne peuvent être créées qu'avec le pavé  .

C'est de l'informatique !

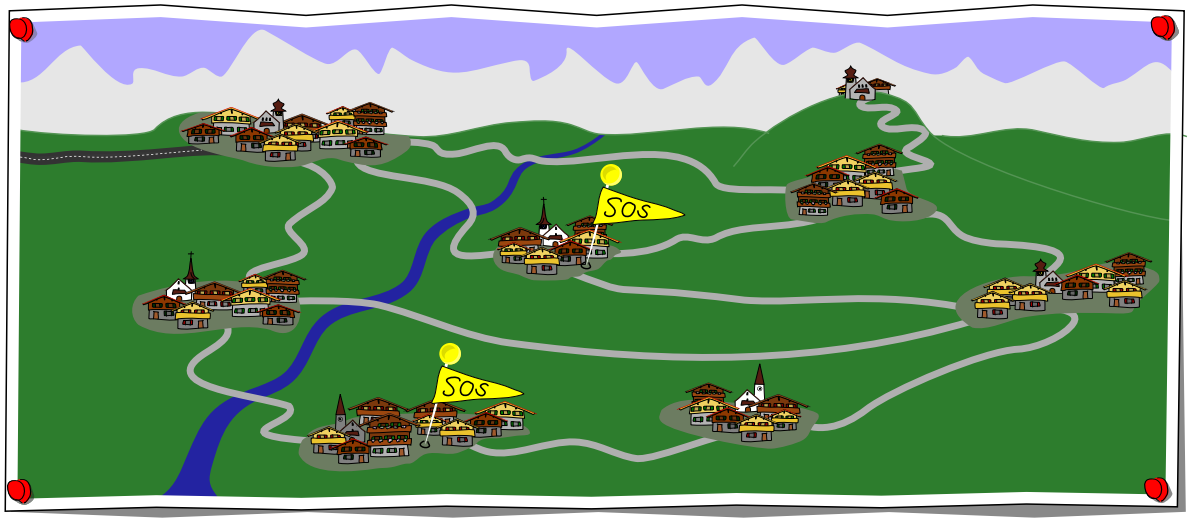
Ces pavés sont nommé d'après Sébastien Truchet (* 1657 ; † 1729), qui en a développé différentes variantes. Les pavés ayant quatre côtés pareils forment un sous-ensemble des pavés de Truchet (mais les pavés de Truchet ne doivent pas forcément avoir quatre côtés pareils, comme dans trois des motifs de cet exercice). Le fait que des motifs complets peuvent être générés à partir d'éléments très simple est une propriété intéressante que l'on rencontre souvent en informatique. Les pavés de Truchet sont étudiés en mathématiques et en informatique, et sont utilisé dans les jeux vidéos pour créér des labyrinthes et des décors.

Mots clés et sites web

- Pavés de Truchet : `https://en.wikipedia.org/wiki/Truchet_tiles`
- Sébastien Truchet : `https://fr.wikipedia.org/wiki/Sébastien_Truchet`

20. Villages isolés

Plusieurs villages de montagne sont approvisionnés par la grande ville grâce au réseau de routes suivant :

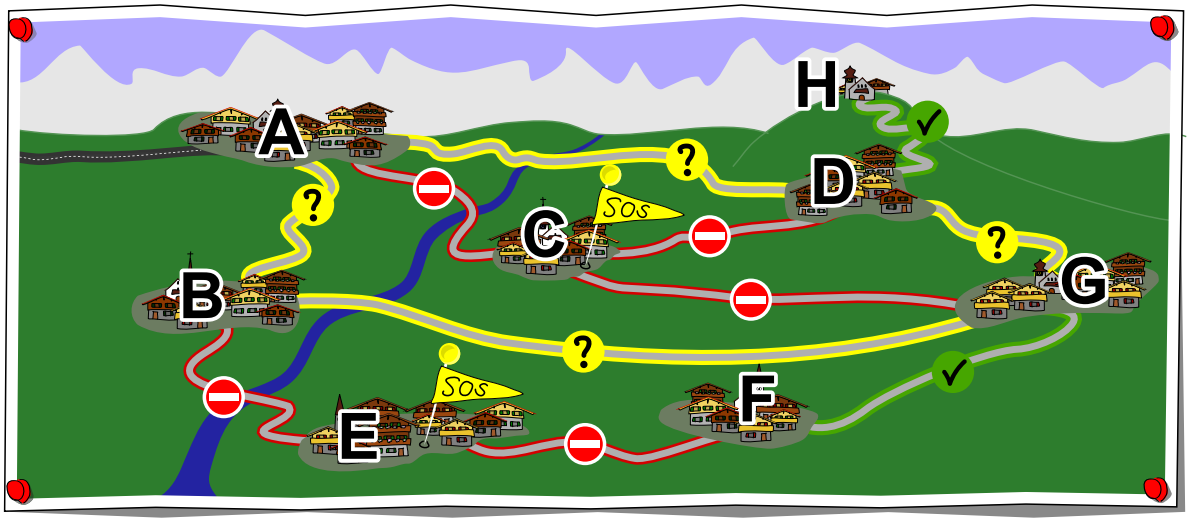


Après une tempête, plusieurs de ces villages ne sont plus du tout accessibles et le signalent avec un drapeau « SOS ». On peut en déduire que certaines des routes sont bloquées.

Indique pour chaque route du réseau entre les villages si elle est (1) bloquée ⛔, (2) ouverte ✅ ou (3) si on ne peut pas le savoir sans informations supplémentaires ❓.

Solution

La carte suivante montre ce que l'on sait sur les connexions dans le réseau routier :



Nous commençons par déterminer quelles routes sont bloquées. Les deux routes menant au village E sont bloquées, car le village E serait accessible si ce n'était pas le cas. De même, les trois routes menant au village C sont bloquées, car il serait accessible sinon.

Ensuite, nous cherchons les routes qui doivent être ouvertes. La route entre les villages G et F doit être ouverte, car le village F ne serait pas accessible autrement, la route entre les villages F et E étant bloquée. La route entre l'église H et le village D doit aussi être ouverte, vu que H est accessible et qu'on ne peut y accéder qu'en passant par D.

Il ne reste que les routes qui sont peut-être accessibles. Comme les village B, G et D sont reliés plusieurs fois au village A, on ne peut pas déterminer quelles routes parmis celles qui restent sont ouvertes. On pourrait par exemple accéder au village B par le village A, mais aussi par le village G. C'est la même chose pour le village D. On peut accéder on village G par le village B ou par le village D. N'importe laquelle des routes dans le circuit A - B - G - D - A pourrait donc être fermée et les quatre villages resteraient accessibles.

C'est de l'informatique !

Comme dans les réseaux routiers, il peut aussi il y avoir des connexions fautives, surchargées ou défectueuses dans les réseaux informatiques. Pour éviter les pannes, on planifie souvent des mesures de sécurité, comme par exemple plusieurs connexion menant au même endroit. On appelle cela la *redondance*.

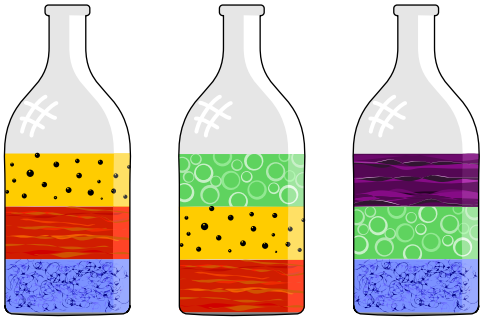
La correction de dysfonctionnements d'un système est une tâche que les informaticiens doivent souvent effectuer ; pas seulement dans les réseaux informatiques, mais aussi lors de développement de programmes. Pour corriger un problème, il faut identifier sa source exacte ; c'est un processus qui se fait en général étape par étape. Certains programmeurs disent que l'on ne peut jamais trouver toutes les erreurs et tous les bugs d'un programme.

Mots clés et sites web

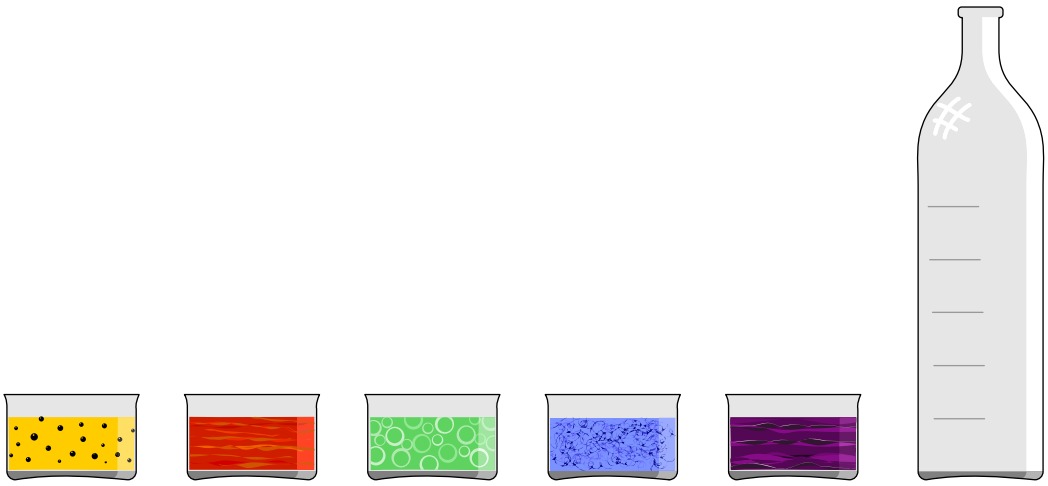
- Redondance : `https://fr.wikipedia.org/wiki/Redondance_(ingénierie)`
- Bug : `https://fr.wikipedia.org/wiki/Bug_(informatique)`

21. Couches de liquides

Marc a des des bouteilles qui contiennent chacune trois liquides formant des couches superposées. Il sait que les liquides à densité plus faible se mettent toujours au dessus des liquides à densité plus forte. Il aimerait maintenant voir à quoi une grande bouteille dans laquelle on met tous les liquides colorés ressemble.

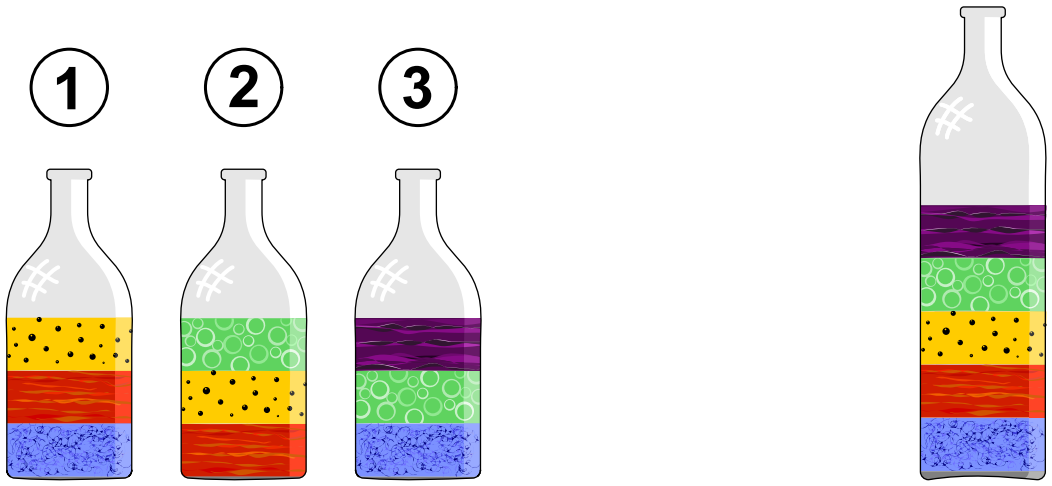


Arrange les cinq couches de liquides colorés dans la bouteille dans leur ordre final.



Solution

L'image montre le bon arrangement des couches de liquide dans la grande bouteille.



Tu trouves dans quel ordre se trouvent les couches de liquide de la façon suivante : étape par étape, tu enlèves dans ta tête les liquides qui ne sont pas au dessus d'autres liquides dans aucune des trois bouteilles données, et les verses dans la grande bouteille.

Au départ, le liquide bleu est tout au fond des bouteilles 1 et 3 et jamais au dessus d'une autre couche de liquide. Le liquide rouge est tout au fond de la bouteille 2, mais au-dessus du liquide bleu dans la bouteille 1 et doit donc avoir une densité plus faible que le liquide bleu. On enlève donc le liquide bleu des bouteilles et le verse dans la grande bouteille.

La liquide rouge est maintenant le seul qui n'est pas au dessus d'un autre liquide. On l'enlève des bouteilles 1 et 2 et le met dans la grande bouteille. Ensuite viennent le liquide jaune, puis le vert et finalement le violet, qui a la plus faible densité et au dessus duquel ne se trouve aucun autre liquide.

C'est de l'informatique !

Dans cet exercice, tu as évalué l'arrangement des liquides dans les trois bouteilles et trié les liquides par densité.

Une substance a beaucoup de propriétés mesurables, par exemple la température d'ébulition, la température de fusion, la conductivité et la densité. Dans le cas présent, la densité a été utilisée comme critère pour trier des substances.

Le tri de données joue un rôle important dans beaucoup de programmes informatiques. La méthode utilisée dans cet exercice pour déterminer l'ordre des couches de liquide s'appelle *tri topologique*. Elle est utilisée pour trier des objets lorsque l'on connait la *relation d'ordre* entre certains des objets (si l'on sait déjà que certains objets en précèdent ou suivent d'autres).

Mots clés et sites web

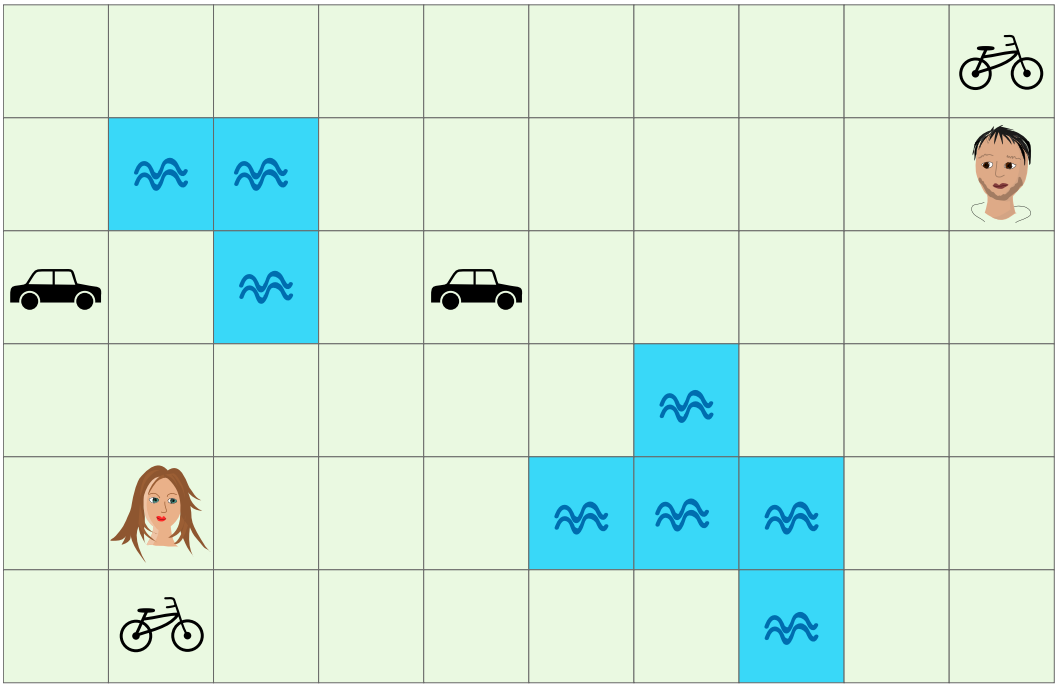
- Relation d'ordre : `https://fr.wikipedia.org/wiki/Relation_d'ordre`
- Tri topologique : `https://fr.wikipedia.org/wiki/Tri_topologique`

22. Un, deux, trois, partez, feu !

Deux amis veulent se voir le plus vite possible. Ils peuvent passer d'une case à la case voisine de droite, de gauche, du haut ou du bas.

La traversée d'une case à une autre prend 1 minute à pied. S'ils arrivent sur une case avec un véhicule, il peuvent l'utiliser. Ils peuvent alors avancer de 2 cases en 1 minute à vélo et de 5 cases en 1 minute en voiture.

Les changements de direction sont toujours possible. Ils ne peuvent pas traverser les étendues d'eau.

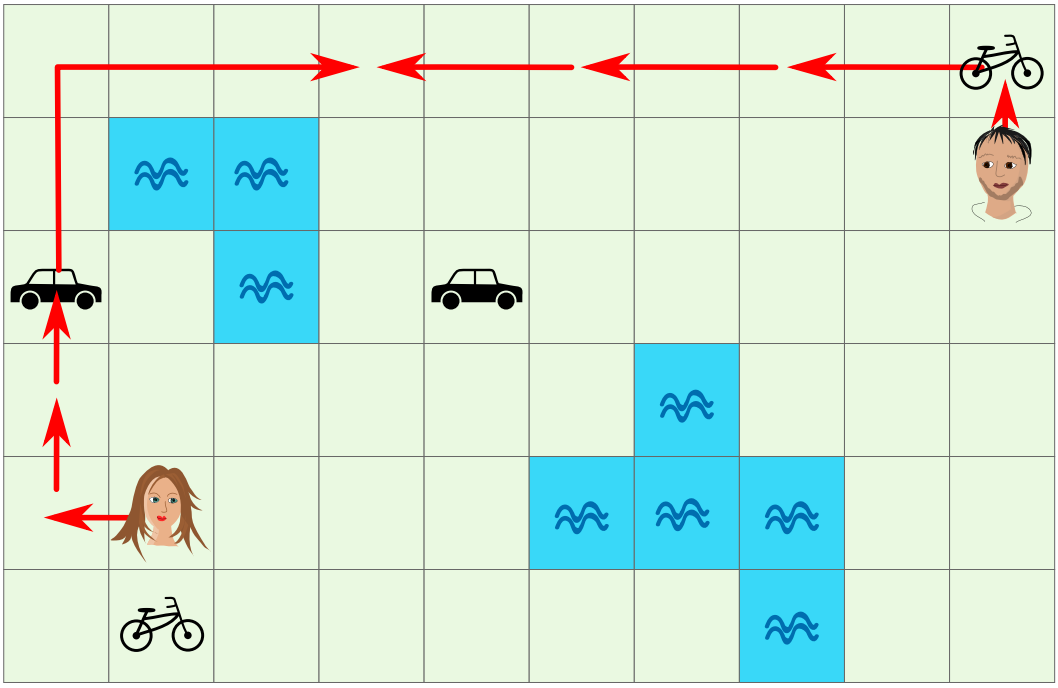


De combien de minutes au minimum les deux amis ont-ils besoin pour se retrouver sur la même case ?

A) 1 minute
B) 2 minutes
C) 3 minutes
D) 4 minutes
E) 5 minutes
F) 6 minutes

Solution

La bonne réponse est 4. L'image montre un chemin permettant aux deux amis de se retrouver sur la même case en 4 minutes.



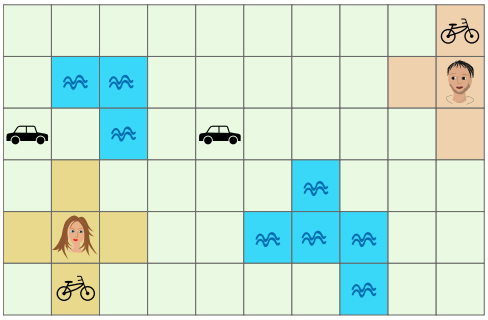
Maintenant, il faut encore prouver qu'ils ne peuvent pas se retrouver en 3 minutes. Les deux amis sont à 11 cases de distance l'un de l'autre. En trois minutes, s'ils se déplacent à pied, il ne peuvent se rapprocher que de 6 cases en tout. Si l'un des deux a atteint un vélo et que l'autre va à pied, ils peuvent se rapprocher de 9 cases, ce qui ne suffit pas non plus. Même s'ils se déplacent les deux à vélo, ils n'y arrivent pas : ils pourraient se rapprocher de 12 cases en 3 minutes, mais les deux vélos sont à 13 cases l'un de l'autre.

Il ne leur reste donc que la possibilité d'utiliser une voiture. En 3 minutes, la fille peut atteindre une voiture, mais n'aurait plus le temps de l'utiliser. Le garçon ne peut pas atteindre de voiture en 3 minutes.

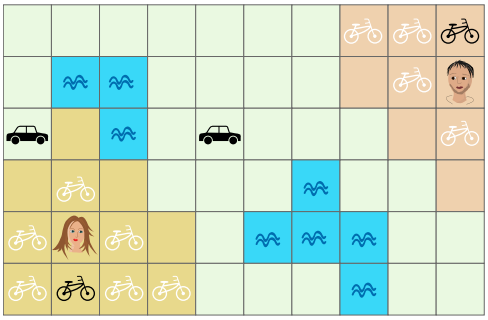
C'est de l'informatique !

Comment as-tu résolu l'exercice ? As-tu trouvé un chemin rapide par hasard et espéré qu'il n'en existe pas de plus rapide ? Ou as-tu essayé beaucoup de chemins différents en te rappelant du plus rapide ?

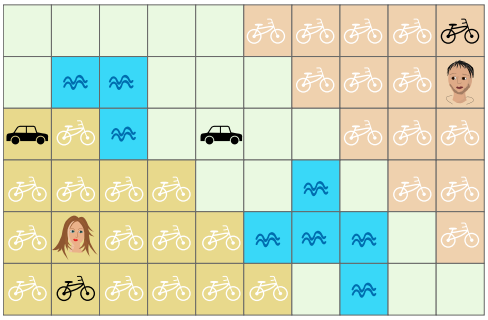
Les programmes informatiques qui ont été développés pour résoudre ce genre de problème travaillent souvent avec une méthode appelée *parcours en largeur*. Dans cet exercice, le parcours en largeur se passe comme cela :



1. Sélectionne toutes les cases que chacun des deux amis peut atteindre en 1 minute.

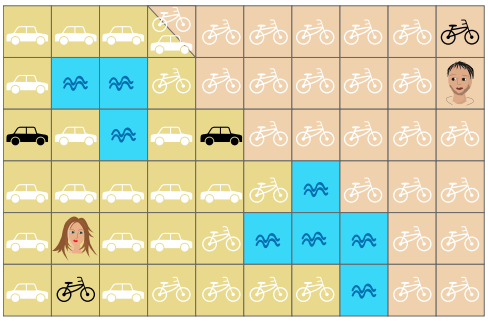


2. Sélectionne toutes les cases qui peuvent être atteinte en (maximum) 1 minute depuis les cases sélectionnées à l'étape 1. Note quel moyen de transport a été utilisé.



3. Sélectionne toutes les cases qui peuvent être atteinte en (maximum) 1 minute depuis les cases sélectionnées à l'étape 2.

Comme les deux régions que tu as sélectionnées ne se chevauchent pas encore, les deux amis ne peuvent pas encore se voir après 3 minutes.



4. Sélectionne toutes les cases qui peuvent être atteinte en (maximum) 1 minute depuis les cases sélectionnées à l'étape 3.

Maintenant, les deux régions se chevauchent sur une case. Elle peut être atteinte en 4 minutes par la fille en voiture et par le garçon en vélo.

Les systèmes de navigation trouvent le chemin le plus rapide entre deux points. Pour cela, ils font attention à ce que le chemin passe par des routes adaptées, et non pas à travers champs ou par des rivières. Cet exercice ressemble à un problème de navigation, mais ici, ce n'est pas une personne qui doit arriver à un but, mais deux personnes qui doivent arriver à un but commun inconnu au départ.

Comme un ordinateur procède de manière systématique lors d'un parcours en largeur, il peut aussi trouver des solutions qui ne sont pas évidentes. Parfois, un détour par une route avec moins de feux

peut être plus rapide que le chemin le plus court entre le départ et l'arrivée. Un voyage en train avec changement peut être plus rapide qu'un voyage direct en bus.

Il existe en informatique plusieurs méthodes pour résoudre des problèmes de ce type. En plus de la méthode de parcours en largeur discutée ici, il existe une méthode appelée *Branch and Bound* (*séparation et évaluation* en français). Le parcours en largeur tient compte de chaque solution partielle obtenue après un certain nombre d'étapes. Avec *Branch and Bound*, les solutions partielles ne menant pas à la solution optimale ne sont plus considérées dans les étapes suivantes.

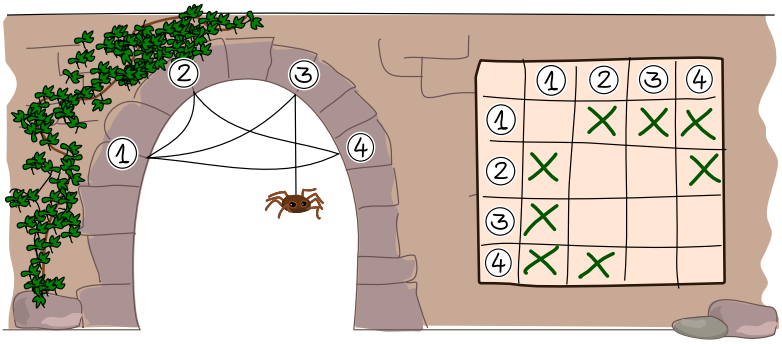
Lorsqu'un problème devient trop complexe, cela peut durer trop longtemps d'essayer toutes les possibilités pour trouver la meilleure solution, même avec l'ordinateur le plus rapide du monde. En pratique, il suffit à un système de navigation de trouver un très bon chemin, même si ce n'est pas le meilleur chemin possible — si tu peux atteindre ton but en 78 minutes, ça ne te fait probablement rien qu'on puisse théoriquement aussi l'atteindre en 77 minutes.

Mots clés et sites web

- Parcours en largeur :
  `https://fr.wikipedia.org/wiki/Algorithme_de_parcours_en_largeur`
- Séparation et évaluation : `https://fr.wikipedia.org/wiki/Séparation_et_évaluation`

23. Toiles d'araignée

Thekla l'araignée aimerait construire autant de toiles différentes que possible. Pour cela, elle a mis au point une méthode lui permettant de décrire la structure exacte de ses toiles.

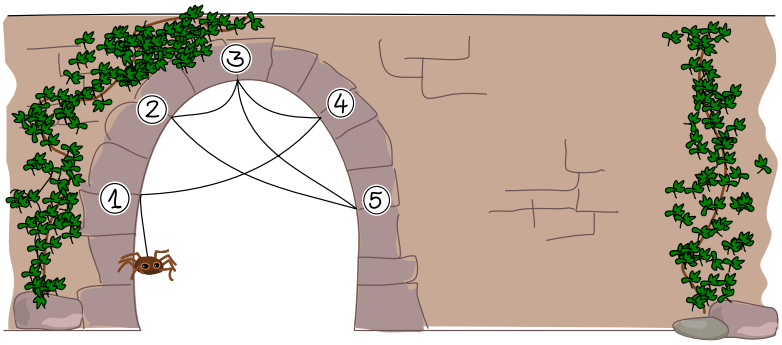


Elle procède de la façon suivante : elle numérote les points d'ancrage de la toile de 1 à N et utilise les case d'une grille comme cela :

- S'il y a un fil qui relie le point d'ancrage x au point d'ancrage y, elle met un « X » dans la case située dans la colonne x et dans la ligne y.

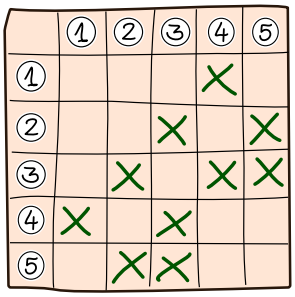
Un fil qui relie le point d'ancrage x au point d'ancrage y relie également le point d'ancrage y au point d'ancrage x.

Thekla construit à présent cette toile :

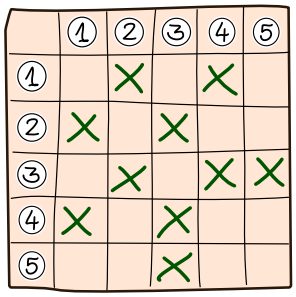


Comment Thekla décrit-elle la structure de cette toile ?

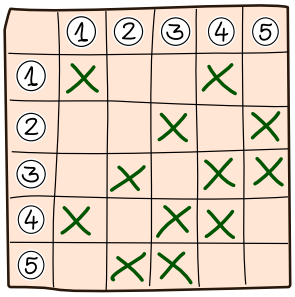
A)



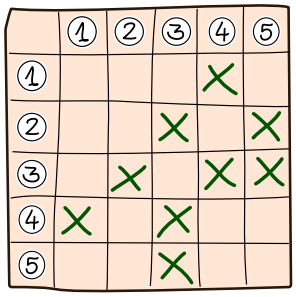
B)



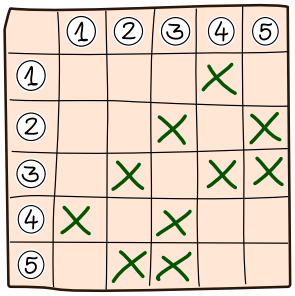
C)



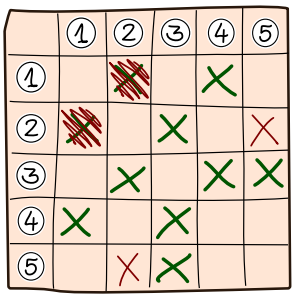
D)



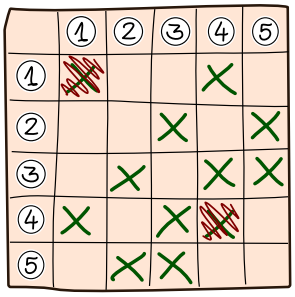
Solution



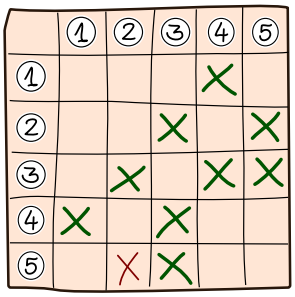
La réponse A est correcte, car toutes les cases sont remplies correctement d'après la règle.



La réponse B comporte une connection supplémentaire erroné (entre le point d'ancrage 1 et le point d'ancrage 2 dans les deux directions) et une connection a été oubliée (entre le point d'ancrage 2 et le point d'ancrage 5 dans les deux directions).



Concernant la réponse C : d'après la règle, aucun « X » ne peut être présent dans la diagonale allant du haut à gauche au bas à droite. ils représenteraient en effet des connections entre d'un point d'ancrage et lui-même. Ceux-ci peuvent exister dans certains réseaux, mais pas dans notre toile d'araignée. La réponse C comporte cependant deux telles connections (au points d'ancrage 1 et 4).



Le réponse D n'est pas symétrique par rapport à la diagonale allant du haut à gauche au bas à droite, alors que toutes les descriptions de toiles devraient l'être. Dans cette réponse, le point d'ancrage 2 et le point d'ancrage 5 sont connectés, mais la connection correspondante dans l'autre direction manque.

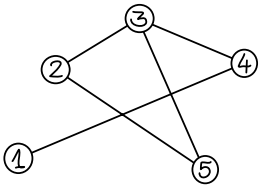
C'est de l'informatique !

La toile d'araignée peut être considérée comme un *graphe* ; un concept qui apparait fréquemment en informatique.

Un graphe est composé de *nœuds* (les points d'ancrage de la toile) et d'*arêtes* (les fils entre les points d'ancrage). Les graphes peuvent aussi être utilisés pour représenter des objets et les relations entre ces objets. Par exemple, un graphe pourrait représenter la manière dont des personnes sont connectées sur un réseau social ou des vols d'avion entre différents pays.

Cet exercice montre comment la structure d'une toile d'araignée peut être enregistrée dans une grille. Certaines propriétés de la toiles sont perdues dans cette représentation, comme par exemple l'apparence exacte de la toile. Souvent, on ne s'intéresse pas aux propriétés géométriques exactes

d'une toile, mais seulement à sa structure. Les informations essentielles sont conservées : combien y a-t-il de nœuds ? Quelles paires de nœuds sont-elles reliées par une arête ?



	1	2	3	4	5
1	0	0	0	1	0
2	0	0	1	0	1
3	0	1	0	1	1
4	1	0	1	0	0
5	0	1	1	0	0

La représentation utilisée ici n'est qu'une possibilité parmi beaucoup de représenter la structure d'un réseau. Cette méthode n'est pas très économe, car toutes les connections sont enregistrées dans les deux sens, ce qui ne serait pas nécessaire ; les cases de la diagonale ne seraient pas nécessaire non plus. Cette méthode a par contre l'avantage de mettre en évidence les erreurs de représentation. Les réponses C et D, par exemple, peuvent être éliminées sans même considérer la toile d'araignée.

La méthode de représentation utilisée dans cet exercice s'appelle une *matrice d'adjacence*.

Mots clés et sites web

- Matrice d'adjacence : `https://fr.wikipedia.org/wiki/Matrice_d'adjacence`

24. Pile de fruits

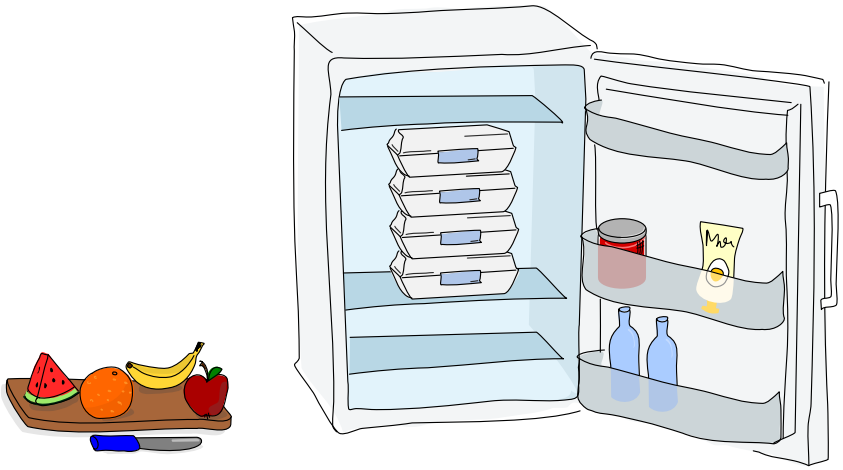
Papa, Maman, Dorie et Ron Castor préparent quatre boîtes avec un fruit différent dans chacune : pomme, banane, orange et pastèque. Les boîtes sont empilées dans le réfrigérateur. Le matin, les castors sont encore très fatigués et prennent simplement la boîte du haut de la pile en quittant le gîte sans la regarder plus en détail.

On ne sait pas exactement dans quel ordre les castors quittent le gîte, mais Maman part dans tous les cas avant Dorie et Papa sort toujours en dernier.

Les membres de la famille aiment des fruits différents. Le tableau suivant indique ce que chaque membre de la famille aime :

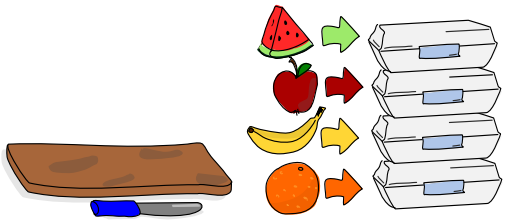
				
Papa	—	—	✓	—
Maman	✓	—	✓	✓
Dorie	✓	✓	✓	—
Ron	✓	✓	—	✓

Mets les fruits dans les boîtes de manière à ce que chaque castor prenne une boîte contenant un fruit qu'il aime.



Solution

Il n'y a qu'une possibilité de répartir les fruits de manière à garantir que chacun ait quelque chose qu'il aime :



Papa n'aime que les oranges et part en dernier. L'orange va donc dans la boîte la plus basse. Ron part en premier, deuxième ou troisième. Comme Maman part avant Dorie, on connait l'ordre de départ exact des castor si l'on sait quand Ron quitte le gîte. Les trois ordres de départ suivants sont possibles :

1.	Maman	Maman	Ron
2.	Dorie	Ron	Maman
3.	Ron	Dorie	Dorie
4.	Papa	Papa	Papa

Maman, Dorie et Ron peuvent chacun partir en deuxième. Il faut donc mettre un fruit que les trois aiment dans la deuxième boîte, et ce n'est le cas que de la pomme. Il ne reste ainsi que la banane et la pastèque pour la boîte du haut. Comme maman n'aime pas les bananes, il faut y mettre la pastèque. La banane va ainsi dans la troisième boîte.

C'est de l'informatique !

Le bon ordre d'une séquence est important dans beaucoup de domaines de l'informatique : beaucoup de calculs doivent utiliser des résultats intermédiaires pour arriver au résultat final. Si différentes étapes de calcul sont effectuées sur différents ordinateurs sans planning consciencieux, des *interblocages* (ou *àtreintes fatales*, *deadlock* en anglais) peuvent avoir lieu. Ce sont des situations dans lesquelles plusieurs ordinateurs s'attendent mutuellement et qui empêchent le programme de se terminer. Un mauvais ordre ne mène cependant souvent qu'à des erreurs (comme il peut amener la mauvaise humeurs chez les castors qui découvrent leur fruit). Par exemple, si l'on doit calculer à l'aide de la formule $Z = (A + B) \times (A - B)$, on peut partager cela en plusieurs étapes d'un programme comme cela :

Entrée A
Entrée B
Calcule $X = A + B$
Calcule $Y = A - B$
Calcule $Z = X \times Y$

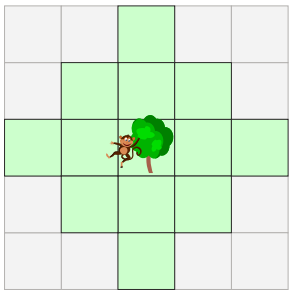
Si l'on essaie maintenant d'effectuer par exemple l'étape $Z = X \times Y$ avant d'avoir calculé X, cela cause une erreur et l'interruption du programme. Alternativement, une valeur par défaut va être

utilisée pour X, ce qui mène à un faux résultat dans la plupart des cas. L'ordre dans lequel les commandes sont effectuées est donc très important en informatique.

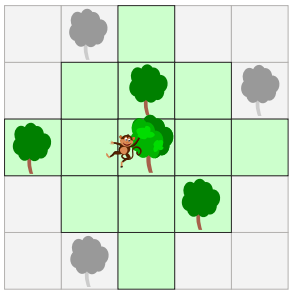
Mots clés et sites web

- Interblocage : `https://fr.wikipedia.org/wiki/Interblocage`

25. Petit singe



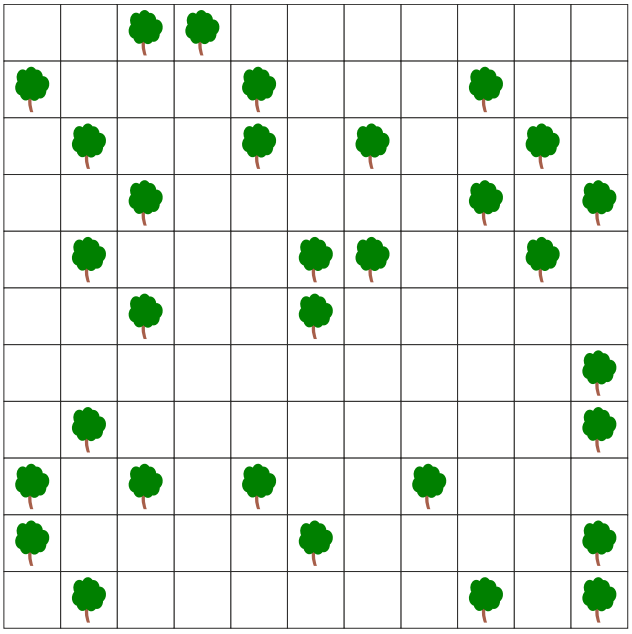
Coco le petit singe peut sauter depuis un arbre aussi loin que le montre le domaine coloré en vert.



Dans l'exemple suivant, Coco peut atteindre les arbres en couleur d'un seul saut. En deux sauts, il peut aussi atteindre les deux arbres gris du haut, mais pas l'arbre gris du bas.

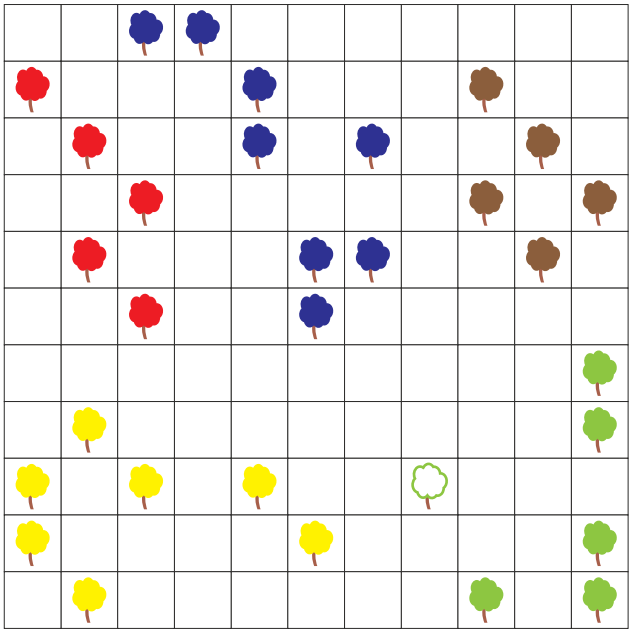
Il existe des groupes d'arbres dans lequels Coco peut se déplacer sans toucher le sol une seule fois.

Sélectionne tous les arbres du plus grand de ces groupes.



Solution

Sur l'image ci-dessous, deux arbres ont la même couleur si Coco peut passer de l'un à l'autre sans toucher le sol.

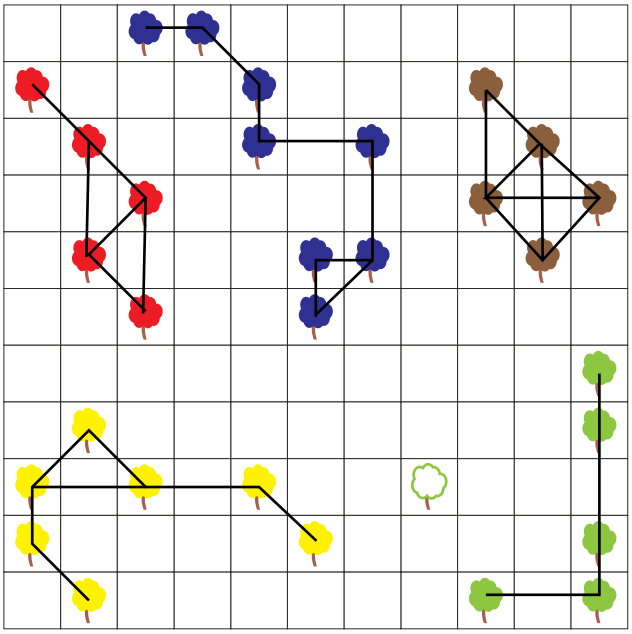


On voit que le groupe d'arbres bleus comptant huit arbres est le plus grand des groupes.

C'est de l'informatique !

Si Coco peut sauter directement d'un arbre à l'autre, ils sont pour ainsi dire connectés l'un à l'autre. On peut représenter cela à l'aide d'un ligne entre les arbres comme montré plus bas. On obtient donc un graphe dont les arbres sont les nœuds avec des arêtes entre les arbres connectés. Coco peut passer d'un arbre à l'autre en sautant seulemnt s'il existe un chemin le long des arêtes menant d'un arbre à l'autre.

On appelle un groupe de nœuds *connexe* si tous les nœuds sont connectés ensemble par des arêtes. Lorsqu'un tel groupe ne peut plus être agrandi sans perdre la connection entre les nœuds, on parle d'une *composante connexe*. Un graphe peut être divisé de manière unique en composantes connexes ; elles sont colorées sur l'image ci-dessous.



On peut faciliement identifier une composante connexe en partant d'un nœud quelconque puis en cherchant tous les nœuds accessibles par des arêtes.

Mots clés et sites web

- Graphe connexe : `https://fr.wikipedia.org/wiki/Graphe_connexe`

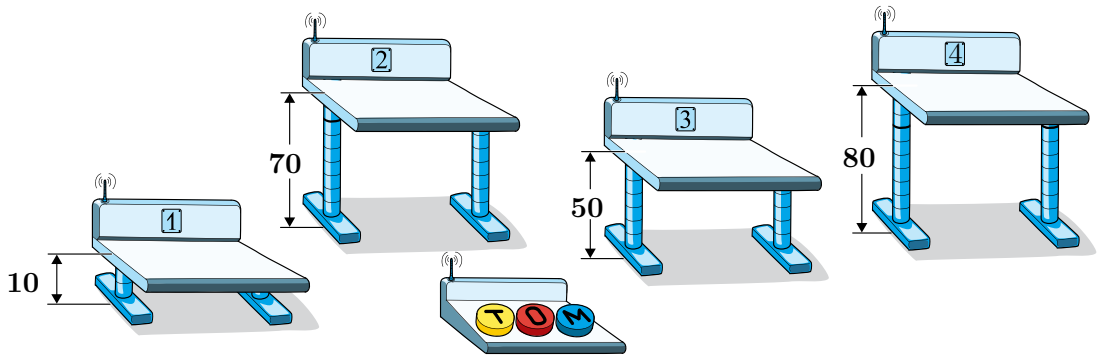
26. Sacrés pupitres

Dans la salle de classe, il y a des pupitres dont la hauteur est réglable électriquement. Tous les pupitres doivent être réglés à 60 cm pour les cours. La hauteur peut être changée à l'aide des touches T, O et M d'une télécommande. Quelqu'un a joué avec la télécommande et l'a reprogrammée. Maintenant, les trois touches fonctionnent comme cela :

- T monte les pupitres 1, 2 et 3 de 10 cm chacun.
- O baisse les pupitres 2, 3 et 4 de 10 cm chacun.
- M monte les pupitres 1, 3 et 4 de 10 cm chacun.

Ces actions sont exécutées chaque fois que l'on appuie sur le bouton.

En ce moment, les hauteurs des pupitres 1, 2, 3 et 4 sont de à 10 cm, 70 cm, 50 cm et 80 cm :



Comment peut-on régler la hauteur des quatre pupitres à 60 cm ?

A) Appuie 4 × sur T, 5 × sur O et 1 × sur M.
B) Appuie 5 × sur T, 1 × sur O et 0 × sur M.
C) Appuie 3 × sur T, 4 × sur O et 2 × sur M.
D) Appuie 2 × sur T, 4 × sur O et 6 × sur M.

Solution

La bonne réponse est C) Appuie 3 × sur T , 4 × sur O et 2 × sur M.

Tu peux constater que les trois touches de la télécommande changent la hauteur de 10 cm, donc toujours de la même distance. Deux des touches font monter les pupitres (T et M) et seulement un touche les fait descendre (O). De plus, chacune des trois touches modifie la hauteur de trois pupitres ; il y a donc toujours un pupitre dont la hauteur ne change pas. La touche O n'a aucun effet sur le pupitre 1, on ne peut donc pas le faire descendre.

Le pupitre 1 est 50 cm trop bas. On peut en déduire que l'on doit appuyer exactement 5 fois sur les touches T ou M (la somme des nombres de fois où les touches T et M sont utilisées doit être égale à 5). On peut exprimer cela par l'équation $T + M = 5$. On peut donc éliminer la solution D), car $T + M = 8$ pour cette solution. D'après la suite de touches de la solution D), la pupitre 1 serait haut de $10 + 20 + 60 = 90$ cm, c'est à dire la hauteur de départ 10 cm plus $2 * 10$ cm pour T plus $6 * 10$ cm pour M.

le pupitre 2 est 10 cm trop haut. M n'a pas d'effet sur le pupitre 2. La bonne solution doit donc satisfaire l'équation $T - O = -1$. On peut donc éliminer la solution B), car en l'utilisant, le pupitre 2 aurait à la fin une hauteur de $70 + 50 - 10 = 110$ cm.

Le pupitre 3 est 10 cm trop bas, ce qui donne l'équation $T - O + M = 1$. Les solutions A) et B) peuvent donc être éliminées. Avec la solution A), le pupitre aurait la même hauteur à la fin qu'au début : $50 + 40 - 50 + 10 = 50$ cm ; avec la solution B, la hauteur du puptire serait $50 + 50 - 10 = 90$ cm. Toutes les solutions excepté la solution C) ont maintenant été éliminées.

Il faut encore vérifier que la solution C) met aussi le pupitre 4 à la bonne hauteur. Le pupitre 4 est 20 cm trop haut et la touche T n'a aucun effet sur sa hauteur. Il faut donc appuyer deux fois sur O et une fois de plus pour chaque fois que la touche M est utilisée. La suite de touches de la solution C) donne la hauteur $80 - 40 + 20 = 60$ cm.

Comme on a déjà constaté plus haut que la solution C) permettait de mettre les pupitres 1, 2 et 3 à la bonne hauteur, on est maintenant sûr que cette solution fonctionne.

Un autre moyen de trouver la solution est de résoudre quatre équations linéaires. Pour chaque pupitre, on écrit une équation décrivant quelles touches changent sa hauteur et quelle est le changement de hauteur désiré. Par exemple, la hauteur du pupitre 1 ne change qu'avec les touches T et M et le changement désiré est de 50 cm, ce que l'on peut obtenir en appuyant sur 5 touches (étant donné que pression sur une touche change la hauteur de 10 cm).

Comme il y a quatre pupitres et trois touches, on obtient quatre équations linéaires avec trois inconnues :

$$\begin{aligned} T + M &= 5 \\ T - O &= -1 \\ T - O + M &= 1 \\ -O + M &= -2 \end{aligned}$$

En soustrayant la troisième équation de la première, on obtient $O = 4$. en substituant dans la deuxièeme équation, on obtient $T = 3$. Toutes les équations sont résolues juste en prenant $M = 2$. C'est donc la seule bonne solution.

C'est de l'informatique !

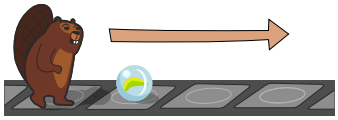
Cet exercice est un problème typique du domaine de l'*optimisation linéaire en nombres entiers*, aussi appelée *programmation linéaire en nombres entiers*. Le problème est donné par un certain nombre de contraintes. Dans ce cas particulier, on peut toutes les formuler sous la forme d'équations linéaires. L'optimisation est un but typique en informatique : on cherche une suite d'action permettant d'arriver à un but prédéfini. On pourrait même décrire tout l'exercice par la recherche d'un chemin dans un espace quadridimensionnel avec trois déplacments possibles pour passer du point $(10, 70, 50, 80)$ au point $(60, 60, 60, 60)$. Cet exercice n'a qu'une solution, mais de tels problèmes ont souvent de multiples solutions, rendant l'optimisation possible. On cherche alors le minimum de la fonction linéaire $T + M + O$.

Mots clés et sites web

- Optimisation linéaire en nombres entiers :
  `https://fr.wikipedia.org/wiki/Optimisation_linéaire_en_nombres_entiers`

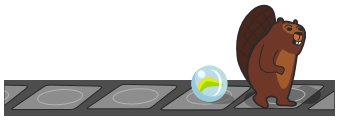
27. Ruban de billes

Sur un ruban, un castor se déplace d'une case à la suivante de gauche à droite. Sur chaque case du ruban peut se trouver une bille.



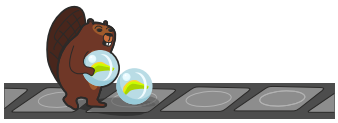
Si le castor arrive sur une case avec une bille et a les mains libres, il la soulève et la prend avec en la portant dans ses bras.

Il dépose la bille sur la prochaine case libre.



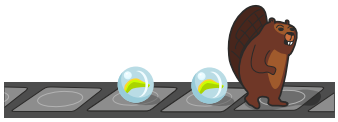
Le castor ne peut porter qu'une seul bille à la fois, et il n'y a la place que pour une bille sur chaque case.

Si le castor porte déjà une bille en arrivant sur une case avec une autre bille...



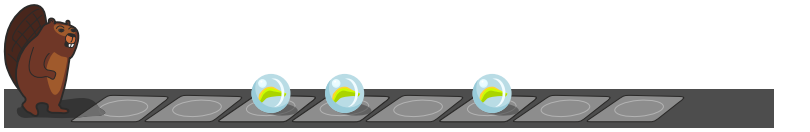
... il dépasse celle-ci...

... et pose sa bille sur la prochaine case vide.



Il peut ensuite à nouveau soulever la bille suivante.

Le castor se trouve devant une partie du ruban sur laquelle sont posées trois billes. Sur quelles cases les billes se trouvent-elles une fois que le castor a traversé cette partie du ruban ?

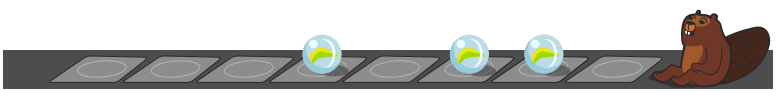


A) 

B) 

C) 

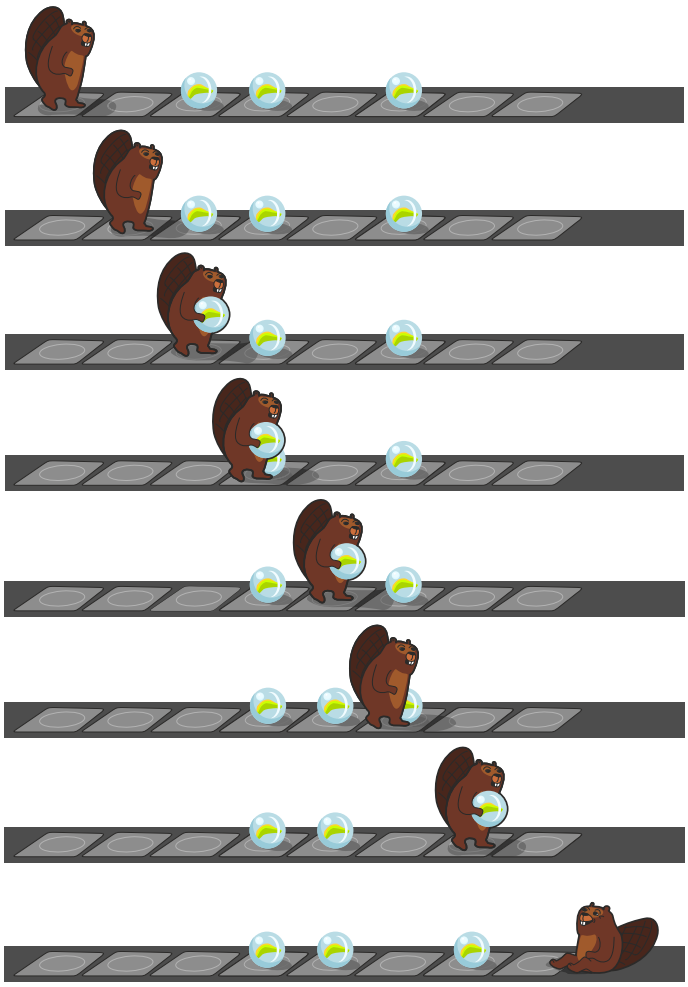
D) 

E) 

Solution

La bonne réponse est C :

L'image suivante montre le déroulement :



C'est de l'informatique !

En informatique, des opérations relativement simples donnent souvent des résultats intéressants. Cet exercice en est un bon exemple. La démarche du castor est un *algorithme*. Il se base sur le fait que le castor peut adopter deux états différents (portant une bille ou n'en portant pas) et qu'il peut trouver deux sortes de cases sur son chemin (occupées ou libres).

Aussi simple que soit l'exemple dans cet exercice, il contient plusieurs des éléments essentiels d'une *machine de Turing*. Une machine de Turing (qui doit son nom au mathématicien Alan Turing) est un ordinateur particulier qui a une structure très simple. En principe, une machine de Turing peut exécuter tous les algorithmes qu'un ordinateur traditionnel peut exécuter. En pratique, les machines

de Turing ne sont pas utilisées comme ordinateur, car nous pouvons construire des ordinateurs qui sont bien plus efficaces, même s'ils sont plus compliqués. Les machines de Turing sont surtout utiles pour la théorie. Leur structure simple permet de prouver des affirmations simples les concernant ; et si l'on peut prouver qu'un exercice ne peut pas être résolu par une machine de Turing, cela veut dire qu'aucun de nos ordinateurs ne peut le résoudre non plus.

Une machine de Turing est composée :

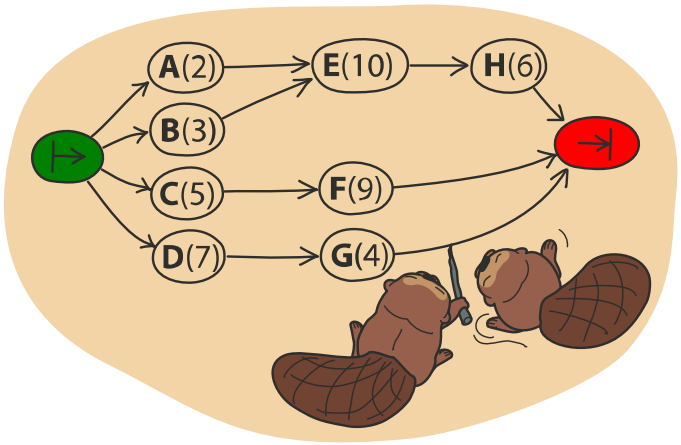
- D'un *ruban* de longueur infinie divisé en *cases*. Chaque case peut contenir un *symbole*. Dans notre cas, il s'agit des cases sur lesquelles le castor se déplace.
- D'une quantité finie de *symboles*. Souvent, on n'utilise que `0` et `1` comme symboles. Dans notre exemple, une bille représente le `1` et une position libre le `0`.
- D'une tête de lecture/écriture qui peut se déplacer sur le ruban dans les deux directions tout en lisant les symboles écrits et en écrivant de nouveaux symboles sur le ruban. Dans notre exemple, le castor joue le rôle de la tête de lecture/écriture.
- D'un registre d'*états* de taille finie. Le comportement de la tête de lecture/écriture peut changer en fonction de l'état. Dans notre cas, il n'y a que deux états : « portant une bille » ou « ne portant pas de bille ».
- D'un ensemble de règles, ou *table d'actions* : que ce passe-t-il, en fonction de l'état, lorsqu'un symbole précis est lu sur le ruban ? Les actions possibles sont : un changement d'état, l'écriture d'un nouveau symbole et le déplacement de la tête de lecture d'une case vers la gauche ou vers la droite.

Mots clés et sites web

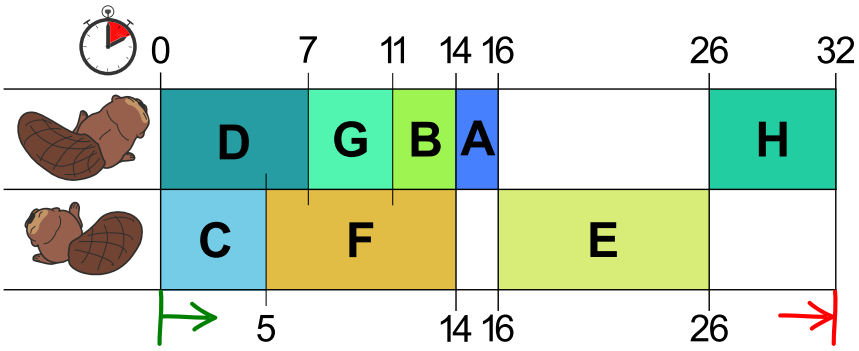
- Machine de Turing : `https://fr.wikipedia.org/wiki/Machine_de_Turing`

28. Plan de travail

La construction d'un barrage par des castors peut être divisée en plusieurs tâches (abattre des arbres, enlever les branches, transporter les troncs jusqu'à la rivière, etc.). L'image de droite montre les huit tâches A, B, C, D, E, F, G et H et le nombre d'heures nécessaires à chacune pour construire un barrage. Les tâches ne sont pas complètement indépendantes les unes des autres : un flèche de X à Y signifie que la tâche X doit être terminée avec de pouvoir commencer la tâche Y.

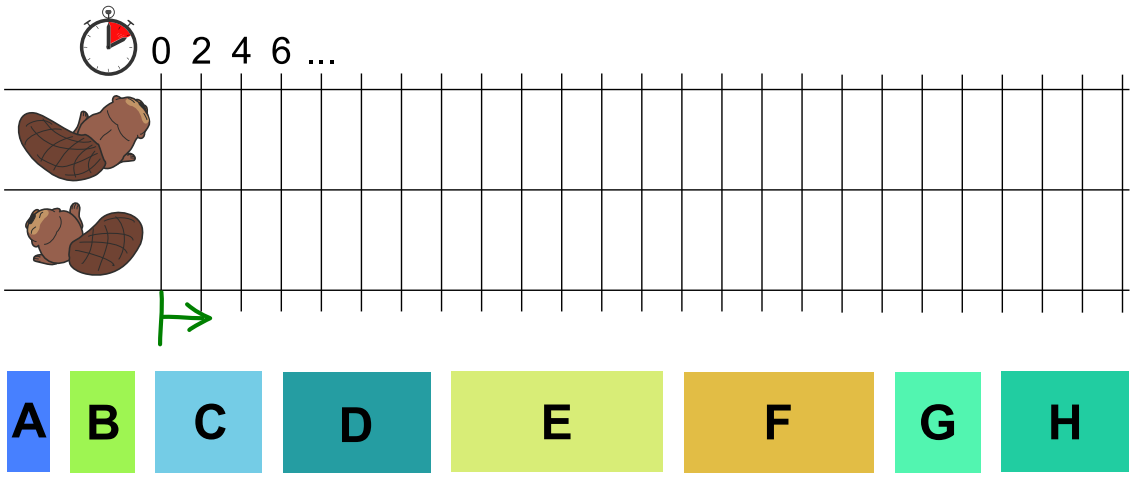


La castore Ulla veut aider le castor Otso à construire le barrage plus vite. Ils se répartissent les tâches et font le plan de travail suivant en tenant compte des dépendances indiquées plus haut.



Le barrage serait ainsi terminé en 32 heures. Mais c'est possible de le faire plus vite !

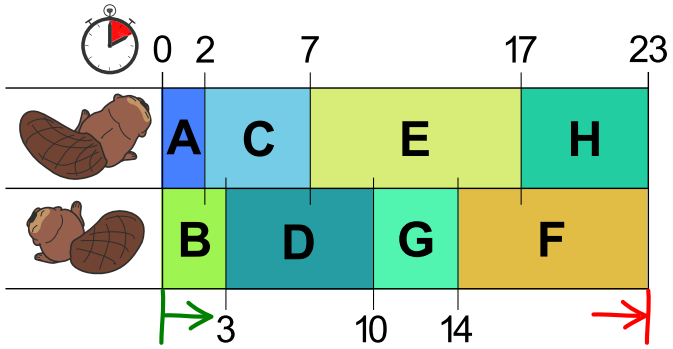
Fais un plan de travail qui permet de finir le barrage le plus vite possible.



Solution

Dans le plan de travail de l'exercice, le premier castor a une longue pause (10 heures), et le deuxième castor a en tout 8 heures de libre. Ils auraient terminé plus vite s'ils travaillaient en permanence.

On obtient un plan de travail plus rapide si l'on fait attention à ce que les deux tâches les plus longues E(10) et F(9) ne soient pas effectuées par le même castor. Voici un plan de travail qui ne dure que 23 heures. Ce n'est pas possible de terminer le barrage plus vite, car les deux castors travaillent ici déjà sans pause.



C'est de l'informatique !

L'être humain est impatient de nature, c'est pourquoi il est souvent souhaitable de finir des travaux aussi rapidement que possible. Lorsqu'un travail peut être réparti (que ce soit entre différentes personnes, machines ou castors), la répartition des différentes tâches entre les travailleurs joue un rôle important dans la durée du travail. Similairement, les ordinateurs doivent souvent répartir leur tâches entre différents processeurs de manière optimale.

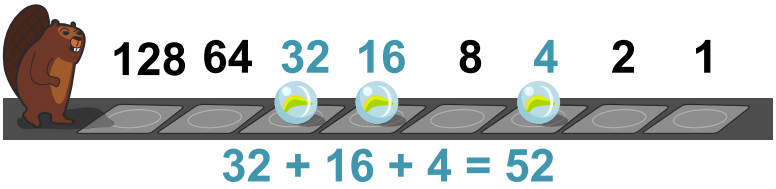
Il existe plusieurs stratégies bien étudiées pour de tels problèmes d'*ordonnancement* en informatique. Pour le premier plan de travail de cet exercice, les plus longues tâches parmi toutes celles à réaliser ont été assignées au castor qui ne travaillait pas à ce moment-là – une mauvaise stratégie dans ce cas. Cela fonctionne souvent mieux lorsque les tâches les plus courtes sont effectuées en premier : la stratégie *Shortest job first* (« le plus court processus en premier » en anglais) minimise aussi le temps d'attente moyen pour chaque tâche.

Mots clés et sites web

- Ordonnancement : `https://fr.wikipedia.org/wiki/Ordonnancement_de_travaux_informatiques`
- Shortest job fext : `https://fr.wikipedia.org/wiki/Shortest_job_first`

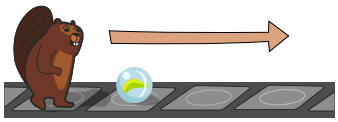
29. Nombres en billes

Les castors ont une manière particulière de représenter les nombres.



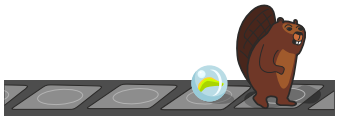
Les différentes cases ont des poids différents et la présence d'une bille sur la case détermine que la valeur est prise en compte. Le nombre 52 est représenté dans l'exemple ci-dessus.

Le castor se déplace sur un ruban d'une case à la suivante de gauche à droite. Sur chaque case du ruban peut se trouver une bille.



Si le castor arrive sur une case avec une bille et a les mains libres, il la soulève et la prend avec en la portant dans ses bras.

Il dépose la bille sur la prochaine case libre.

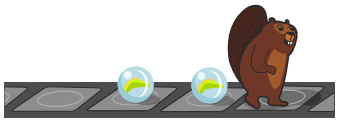


Le castor ne peut porter qu'une seul bille à la fois, et il n'y a la place que pour une bille sur chaque case.

Si le castor porte déjà une bille en arrivant sur une case avec une autre bille...

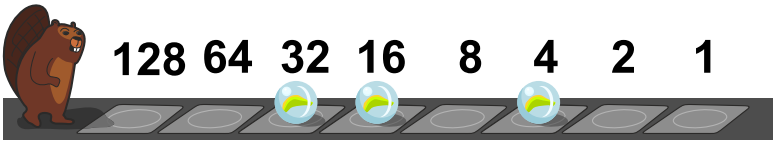
... il dépasse celle-ci...

... et pose sa bille sur la prochaine case vide.



Il peut ensuite à nouveau soulever la bille suivante.

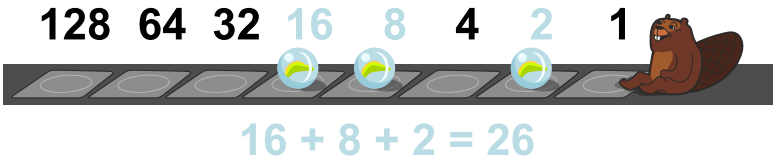
Quel nombre les billes représentent-elles une fois que le castor a traversé cette partie du ruban ?



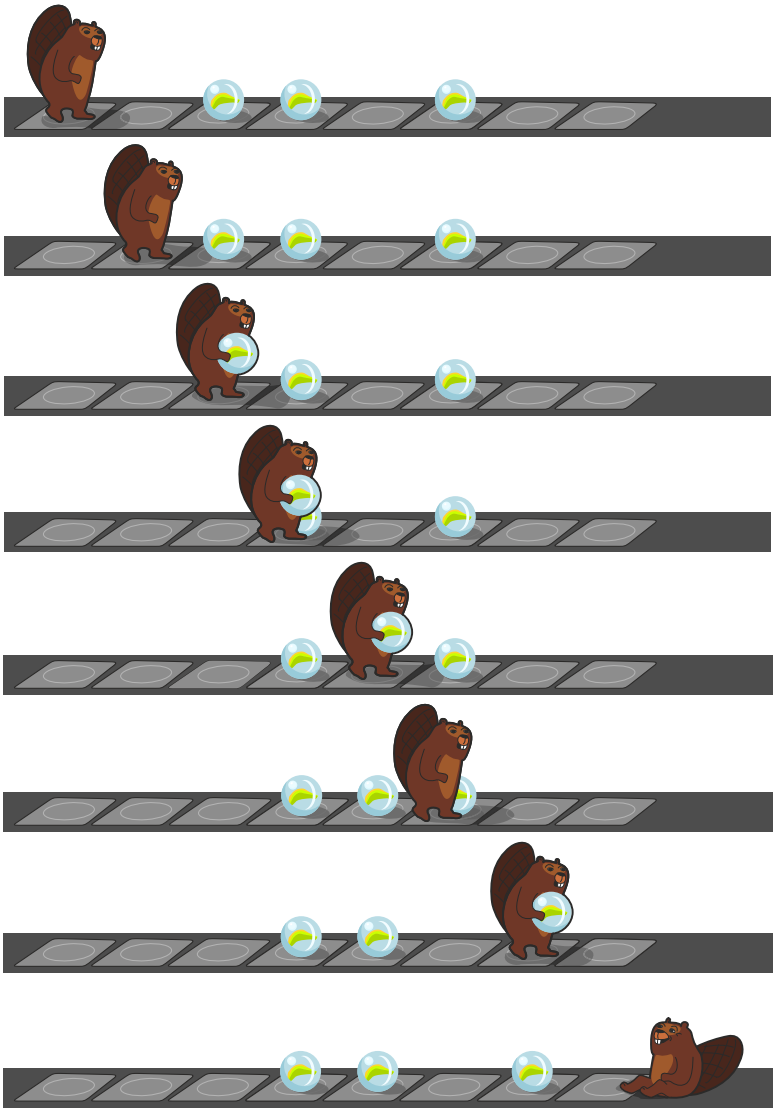
A) 10
B) 26
C) 28
D) 104

Solution

La bonne réponse est B) 26.



L'image suivante montre le déroulement :



C'est de l'informatique !

En informatique, des opérations relativement simples donnent souvent des résultats intéressants. Cet exercice en est un bon exemple. La démarche du castor est un *algorithme*. Il se base sur le fait que le castor peut adopter deux états différents (portant une bille ou n'en portant pas) et qu'il peut trouver deux sortes de cases sur son chemin (occupées ou libres).

Le résultat final de l'algorithme est le même que si l'on avait déplacé chaque bille d'une case vers la droite. Cela repésente une division par deux dans le système numérique des castors. Si le castor se déplaçait de droite à gauche sur le ruban, le nombre serait multiplié par deux. Lorsque toute une rangée de uns et de zéros est décalée vers la gauche ou vers la droite, on appelle cela un *décalage de bits* en informatique.

Aussi simple que soit l'exemple dans cet exercice, il contient plusieurs des éléments essentiels d'une *machine de Turing*. Une machine de Turing (qui doit son nom au mathématicien Alan Turing) est un ordinateur particulier qui a une structure très simple. En principe, une machine de Turing peut exécuter tous les algorithmes qu'un ordinateur traditionnel peut exécuter. En pratique, les machines de Turing ne sont pas utilisées comme ordinateur, car nous pouvons construire des ordinateurs qui sont bien plus efficaces, même s'ils sont plus compliqués. Les machines de Turing sont surtout utiles pour la théorie. Leur structure simple permet de prouver des affirmations simples les concernant ; et si l'on peut prouver qu'un exercice ne peut pas être résolu par une machine de Turing, cela veut dire qu'aucun de nos ordinateurs ne peut le résoudre non plus.

Une machine de Turing est composée :

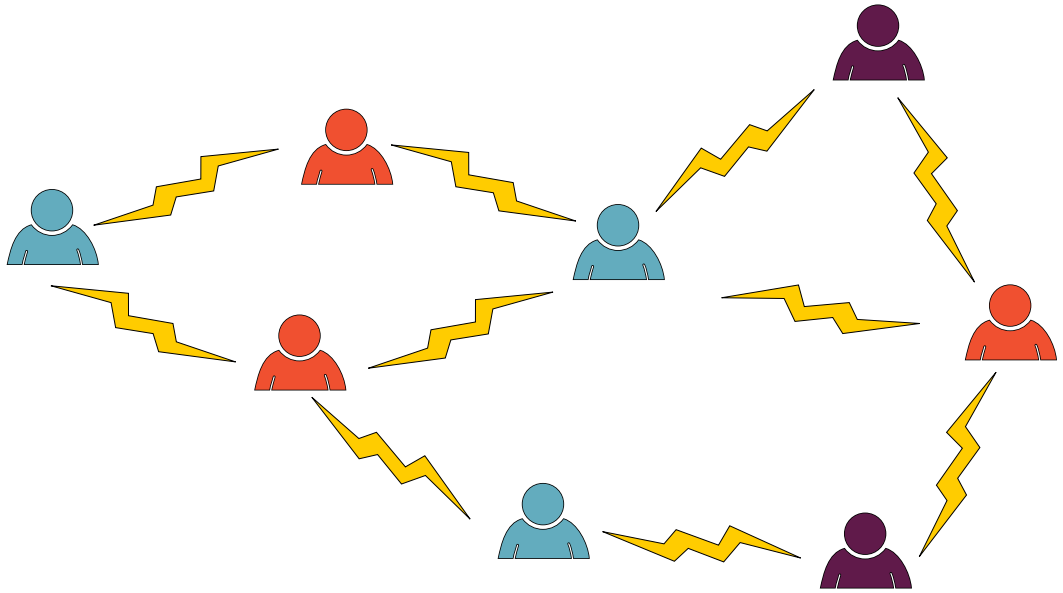
- D'un *ruban* de longueur infinie divisé en *cases*. Chaque case peut contenir un *symbole*. Dans notre cas, il s'agit des cases sur lesquelles le castor se déplace.
- D'une quantité finie de *symboles*. Souvent, on n'utilise que `0` et `1` comme symboles. Dans notre exemple, une bille représente le `1` et une position libre le `0`.
- D'une tête de lecture/écriture qui peut se déplacer sur le ruban dans les deux directions tout en lisant les symboles écrits et en écrivant de nouveaux symboles sur le ruban. Dans notre exemple, le castor joue le rôle de la tête de lecture/écriture.
- D'un registre d'*états* de taille finie. Le comportement de la tête de lecture/écriture peut changer en fonction de l'état. Dans notre cas, il n'y a que deux états : « portant une bille » ou « ne portant pas de bille ».
- D'un ensemble de règles, ou *table d'actions* : que ce passe-t-il, en fonction de l'état, lorsqu'un symbole précis est lu sur le ruban ? Les actions possibles sont : un changement d'état, l'écriture d'un nouveau symbole et le déplacement de la tête de lecture d'une case vers la gauche ou vers la droite.

Mots clés et sites web

- Machine de Turing : `https://fr.wikipedia.org/wiki/Machine_de_Turing`
- Opération bit à bit, décalage de bit : `https://fr.wikipedia.org/wiki/Opération_bit_à_bit#Décalages_de_bit`

30. Travail d'équipe

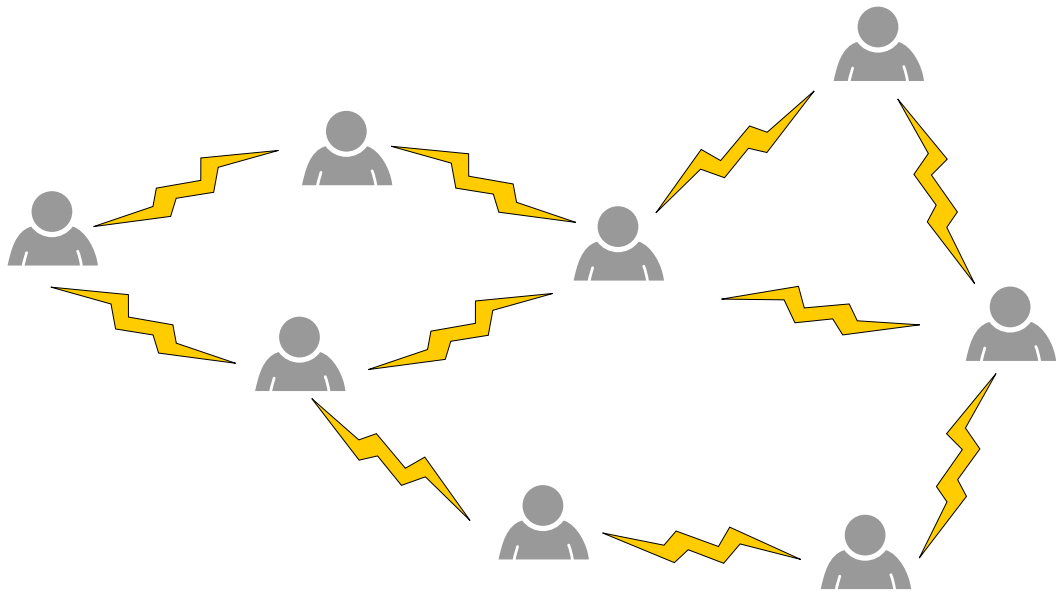
Tu dois répartir huit personnes en groupes de travail pour un projet. Un éclair est dessiné entre deux personnes qui ne veulent pas travailler ensemble. Dans ce cas, tu ne veux pas les mettre dans le même groupe de travail.



Dans l'exemple du haut, une répartition en trois groupes (rouge, bleu, violet) est possible en tenant compte des inimitiés. Il n'y a donc jamais d'éclair entre deux personnes de la même couleur.

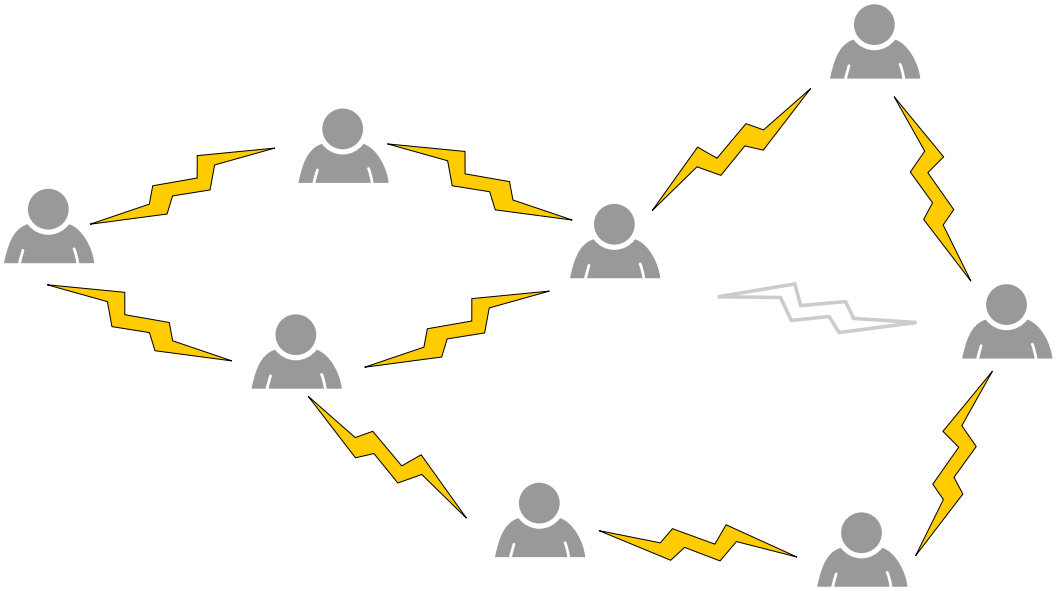
Si tu arrives à convaincre les deux bonnes personnes de travailler ensemble, une répartition en seulement deux groupes (deux couleurs) est possible.

Enlève le bon éclair.

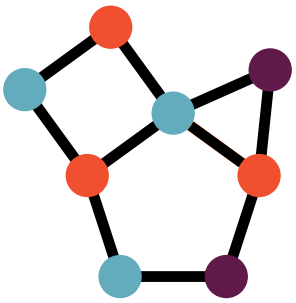


Solution

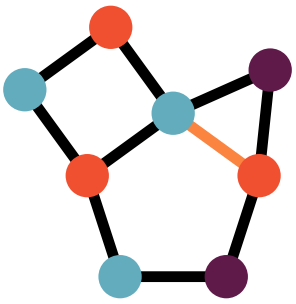
La bonne réponse est :



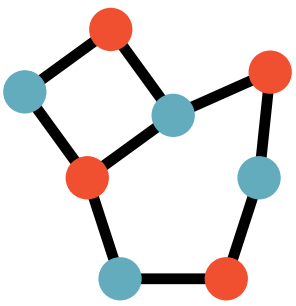
Nous représentons la situation de manière plus abstraite sous forme de *graphe* dans lequel les personnes sont les *nœuds* (cercles) et les éclairs les *arêtes* (lignes).



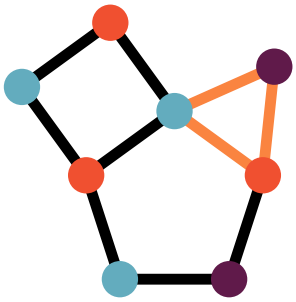
La seule option possible est d'enlever l'arête orange.



Après avoir enlevé cette arête, nous pouvons colorer les nœuds de deux couleurs.

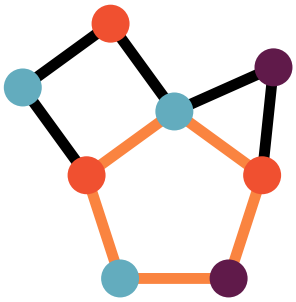


Chaque couleur représente un groupe. On peut voir qu'il n'y a jamais deux personnes du même groupe qui refusent de travailler ensemble : les nœuds voisins ont toujours des couleurs différentes.



Pour déterminer que la seule option est d'enlever cette ligne-là, nous commençons par considérer le triangle orange.

Si une ligne en dehors de ce triangle est éliminée, nous avons toujours besoin de trois couleurs simplement pour le triangle en question.



Considérons maintenant le pentagone orange :

Si une arête en dehors de ce pentagone est éliminée, il reste intact et c'est donc impossible de le colorer avec deux couleurs : si l'on parcours le pentagone dans le sens des aiguilles d'une montre, on doit alterner entre les deux couleurs. Lorsque l'on arrive au dernier nœud, celui-ci a alors la même couleur que le premier nœud juste à côté, étant donné que le nombre de nœuds est impair, comme c'ést le cas pour le triangle.

La seule solution est donc d'enlever l'arête commune au triangle et au pentagone.

C'est de l'informatique !

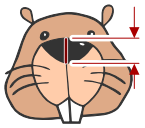
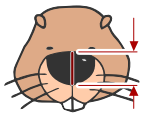
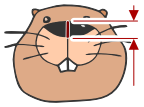
Beaucoup de problèmes du quotidien peuvent être formulés comme des problèmes de *coloration de graphe*. Dans cet exercice du castor, les *nœuds* d'un *graphe* représentent les personnes et une *arête* entre deux personnes montre qu'elles refusent de travailler dans le même groupe. Si nous colorons les nœuds avec k couleurs, cela représente l'assignation des personnes à l'un des k groupes de travail. Un telle coloration est valide si deux nœuds reliés par une arête ont toujours deux couleurs différentes. Dans notre cas, une coloration est donc valide lorsque toutes les personnes de chaque groupe travaillent ensemble. Une arête est appelée *critique* quand une coloration valide avec une couleur de moins devient possible en enlevant cette arête (on peut pour cela changer la couleur de tous le nœuds du graphe). Ici, une arête est donc critique quand on peut réduire le nombre de groupes en convaiquant les deux personnes correspondantes de travailler ensemble.

Mots clés et sites web

- Théorie de graphe : `https://fr.wikipedia.org/wiki/Théorie_des_graphes`
- Coloration de graphe : `https://fr.wikipedia.org/wiki/Coloration_de_graphe`

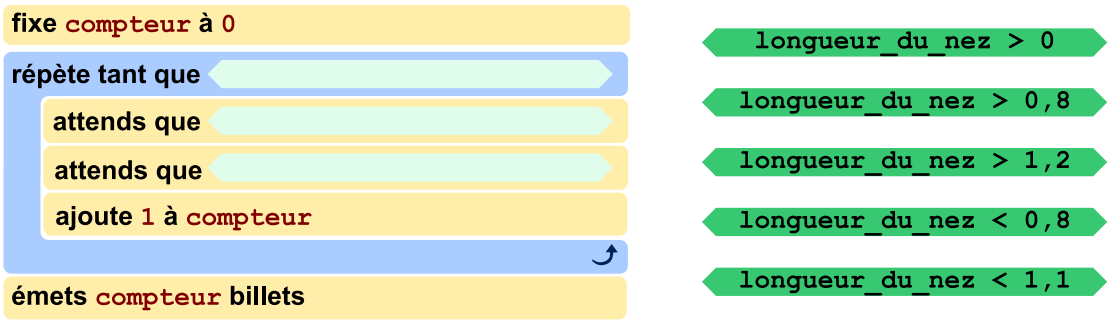
31. Compter avec les muscles

Un nouveau distributeur de billets doit fonctionner de la manière suivante : un client hoche la tête (c'est à dire qu'il baisse la tête puis la remet droite) aussi souvent qu'il veut de billets. Le client lève ensuite la tête, et le distributeur émet les billets. Pour cela, le distributeur a une caméra intégrée. Elle peut reconnaître le nez du client et mesure constamment sa longueur. Le programme de commande du distributeur enregistre le résultat de la mesure actuelle sous le nom `longueur_du_nez` et détermine la position de la tête du client à l'aide du tableau suivant :

Mesure de la caméra	Valeur `longueur_du_nez`	Position de la tête
	1	Le client a la tête droite.
	1,3	Le client a la tête baissée.
	0,7	Le client a la tête levée.

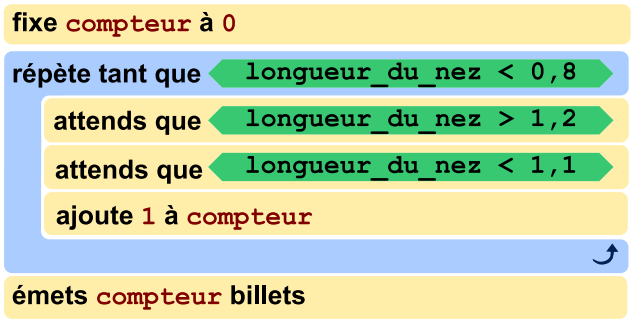
Le programme de commande est presque fini — regarde plus bas.

Complète le programme de commande.



Solution

La seule bonne réponse est :



La structure du programme est donnée : il y a un bloc de commande principal qui se répète, souvent appelé *boucle*. La dernière instruction de cette boucle d'instructions répétées augmente le compteur du nombre de billets à émettre. Les deux instructions commençant par `attends que` doivent donc détecter un hochement de tête du client ; c'est à dire détecter que le client a d'abord baissé la tête puis l'a remise droite. La valeur enregistrée dans longueur_du_nez doit donc d'abord être environ 1,3 puis à nouveau 1. Cela correspond à l'instruction `longueur_du_nez > 1,2` suivie de `longueur_du_nez < 1,1`.

Les instructions de la boucle sont répétées jusqu'à ce que le client lève la tête ; donc jusqu'à ce qu'une valeur clairement inférieure à 1 soit mesurée. La seule condition correspondant à cela est `longueur_du_nez < 0,8`.

Tu as peut-être remarqué que le programme n'utilise pas exactement les valeurs du tableau. En pratique, on ne peut en effet pas mesurer de manière continue, mais seulement à une certaine fréquence (par exemple 25 fois par seconde). Il peut donc arriver, par exemple, que la valeur 1 qui correspond à la tête droite ne soit jamais mesurée, parce que la valeur 1,03 a été mesurée juste après 0,95.

C'est de l'informatique !

La *vision par ordinateur* (*machine vision* ou *computer vision* en anglais) est un domaine de l'informatique très étudié actuellement. Les considérations théoriques et les applications pratiques sont les deux d'une grande importance.

Une application notable de la vision par ordinateur est de permettre aux personnes handicapées de mieux intéragir avec leur environement de manière autonome. Suivant le degré du handicap, il peut arriver qu'une personne ne puisse plus contrôler que très peu de ses muscles, par exemple. Stephen Hawking (1942−2018), physicien mondialement connu, utilisait les mouvements des muscles de ses joues pour contrôler un programme de synthèse vocal après qu'il avait perdu le contrôle de la majeure partie du reste de sa musculature.

Cet exemple concret pourrait aussi être utilisé par les musiciens : ils utilisent en général leurs deux mains pour jouer de leur instrument et ne peuvent donc pas tourner les pages d'une partition. Les appareils usuels disposent pour cette raison d'une pédale. Certains musiciens, comme les organistes,

doivent cependant aussi utiliser leurs pieds pour jouer et pourraient utiliser un simple hochement de tête pour tourner les pages d'une partition automatiquement, par exemple.

Contrairement à l'exemple de cet exercice, dans lequel les valeurs sont concrètes et préprogrammées, la vision par ordinateur est souvent combinée à l'*apprentissage automatique* (*machine learning* en anglais). Le programme s'entraîne alors à reconnaître certains gestes lorsque de nombreux exemples et contre-exemples de chaque geste lui sont présentés. Le programme peut ainsi élaborer un modèle statistique de comment les exemples peuvent être interprétés.

Mots clés et sites web

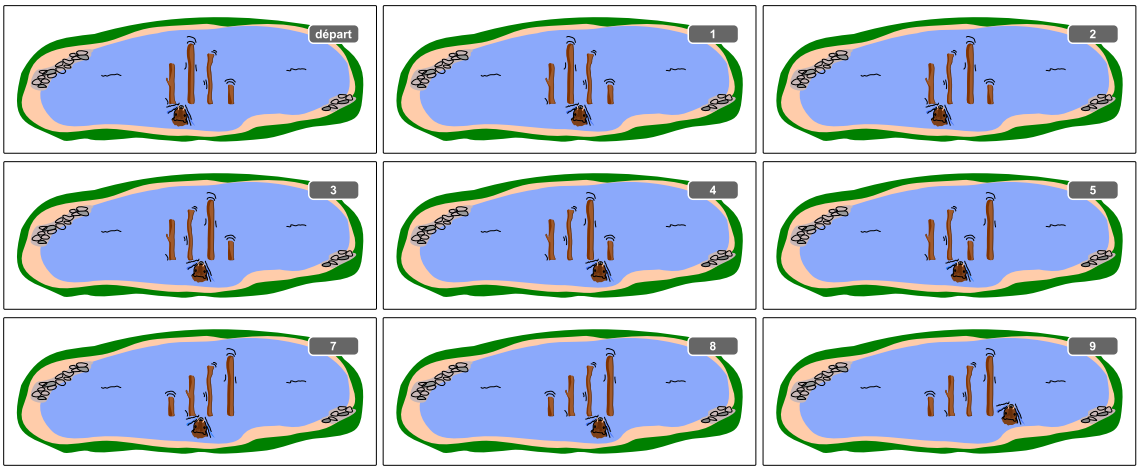
- Vision par ordinateur, machine vision, computer vision :
  `https://fr.wikipedia.org/wiki/Vision_par_ordinateur`
- Apprentissage automatique, machine learning :
  `https://fr.wikipedia.org/wiki/Apprentissage_automatique`
- `https://fr.wikipedia.org/wiki/Stephen_Hawking#Maladie_et_poursuite_de_ses_-travaux`
- `https://fr.wikipedia.org/wiki/Tourneur_de_pages`

32. Beaver Sort

Hamid le castor trie les troncs d'arbre dans le lac. Ils doivent aller du plus petit au plus grand de gauche à droite.

- Hamid commence avec les deux troncs tout à gauche.
- Il compare toujours deux troncs voisins :
  - Si le tronc de droite est plus long que le tronc de gauche, il nage d'un tronc vers la droite.
  - Par contre, si le tronc de gauche est le plus long, il échange les deux troncs. Ensuite, il nage d'un tronc vers la droite s'il se trouve à sa position de départ et d'un tronc vers la gauche sinon.
- Hamid continue comme cela jusqu'à ce qu'il se trouve à la droite des troncs. Les troncs sont alors triés correctement.

L'exemple montre comment Hamid trie 4 troncs. Pour cela, il fait 9 comparaisons en tout.



Le nombre de comparaisons dépend de la configuation de départ des troncs. Pour trier 4 troncs, Hamid doit faire au moins 3 comparaisons (dans le cas où les troncs sont déjà triés correctement) et au maximum 9 comparaisons (dans le cas où les troncs sont dans l'ordre inverse). Hamid doit donc prévoir de faire entre 3 et 9 comparaisons pour trier 4 troncs.

Hamid doit maintenant trier 40 troncs de longueurs différentes. Combien de comparaisons doit-il faire dans le meilleur et le pire des cas ?

A) de 0 à 20 comparaisons
B) de 3 à 40 comparaisons
C) de 39 à 120 comparaisons
D) de 39 à 1521 comparaisons

Solution

Dans le meilleur des cas, celui où tous les troncs sont déjà triés et où Hamid nage toujours vers la droite, Hamid ne doit faire que 39 comparaisons pour trier 40 troncs. Les réponses A) et B) ne peuvent donc pas être justes.

Dans le pire des cas, les troncs sont exactement dans l'ordre inverse. C'est plus difficile de déterminer le nombre de comparaisons nécessaires ici. On constate d'abord que les troncs à la gauche de Hamid sont toujours bien triés. Si Hamid arrive à un tronc qui est plus petit que les k troncs à sa gauche, il l'échange jusqu'à arriver tout à gauche, faisant ainsi déjà k comparaisons. Il revient et nage encore d'un tronc vers la droite, faisant $k-1$ comparaisons mais aucun échange. Il fait donc $k+(k-1)=2k-1$ comparaisons. Ceci a lieu pour chacune des $n-1$ positions entre deux troncs et Hamid a en moyenne $\frac{n}{2}$ troncs à sa gauche. Hamid fait donc $(n-1)\cdot(2\cdot\frac{n}{2}-1)=(n-1)^2$, dans notre cas $39^2=1521$, comparaisons. Cela signifie que la réponse C) avec 120 comparaisons au maximum est fausse, et la bonne réponse est D).

C'est de l'informatique !

Le tri d'élements est une tâche classique en informatique. Une méthode de tri efficace peut permettre de gagner beaucoup de temps.

L'algorithme de tri présenté ici s'appelle *Gnome Sort* et a été introduit par l'informaticien iranien Hamid Sarbazi-Azad sous le nom de *Stupid Sort*. Plus tard, l'informaticien néerlandais Dick Grune a nommé cette méthode de tri Gnome Sort en imaginant qu'un nain de jardin (*garden gnome* en anglais) pouvait trier des pots de fleurs de cette façon.

L'analyse de la durée d'exécution d'un algorithme (c'est à dire du temps dont l'algorithme peut avoir besoin réaliser sa tâche) est très importante en informatique. On veut souvent savoir quelle est la durée dans le meilleur des cas, en moyenne et dans le pire des cas, et cela en fonction de la taille n de l'entrée (dans notre cas du nombre de troncs). Pour faciliter la comparaison de différentes durées d'exécution, on se contente la plupart du temps de donner un ordre de grandeur. La durée du Gnome Sort dans le meilleur des cas en fonction de n est linéaire — on écrit cela $\mathcal{O}(n)$ —, la durée moyenne est de degré deux — on écrit cela $\mathcal{O}(n^2)$, et la durée dans le pire des cas est également de degré 2 — donc également $\mathcal{O}(n^2)$.

Mots clés et sites web

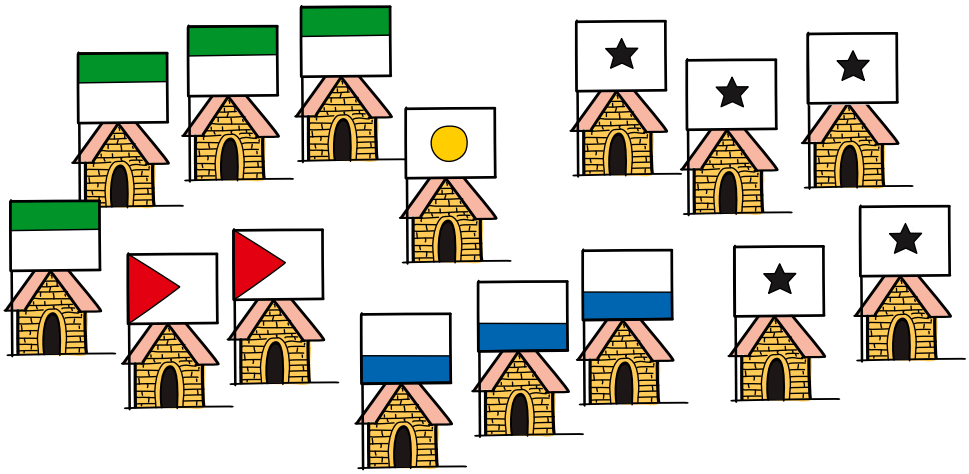
- Gnome Sort : `https://de.wikipedia.org/wiki/Gnomesort`
- Algorithme de tri : `https://fr.wikipedia.org/wiki/Algorithme_de_tri`
- Théorie de la complexité : `https://fr.wikipedia.org/wiki/Théorie_de_la_complexité_-(informatique_théorique)#Relation_au_coût_énergétique`
- O-Notation, Landau-Symbole : `https://fr.wikipedia.org/wiki/Comparaison_asymptotique`

33. Les clans de Castorland

À Castorland vivent cinq clans, autrefois ennemis, qui possèdent chacun un certain nombre de maisons, comme on peut le voir dans l'image : les Mac Intosh, les Applondt, les Mac Rosoft, les Androidier et les Libredosset. Comme ils vivent en paix depuis longtemps, ils décident d'accomplir le rituel d'union. Les règles de ce rituel sont les suivantes :

- Seuls deux clans peuvent s'unir en même temps.
- Une fête d'une semaine a lieu dans chacune des maisons appartenant aux clans qui s'unissent afin de sceller le pacte. La durée du rituel de l'union, en semaines, est donc égale au nombre de maisons des deux clans.
- Après ce rituel, les deux clans ne forment plus qu'un clan. L'union des clans peut alors continuer.

Les clans décident de procéder à l'union en le moins de temps possible. Pour cela, il faut bien planifier l'ordre dans lequel les unions se feront.



Combien de semaines au minimum l'union de tous les clans dure-t-elle ?

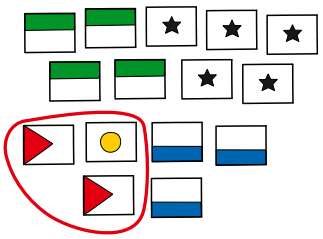
A) 15 semaines
B) 33 semaines
C) 35 semaines
D) 50 semaines
E) 120 semaines

Solution

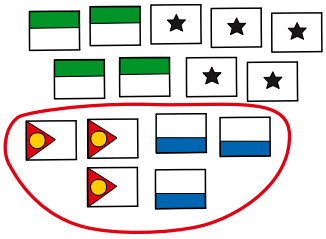
La bonne réponse est B) 33 semaines.

La stratégie optimale pour minimiser le nombre total de semaines nécessaire à l'union de tous les clans consiste à minimiser le nombre de maisons y prenant part. Les maisons des clans unis au début doivent aussi être prises en compte dans les unions suivantes, c'est donc raisonnable de commencer avec les petits clans afin que les grands clans ne doivent pas participer au rituel trop souvent. Pour cela, à chaque étape, les deux clans avec le moins de maisons devraient être unis.

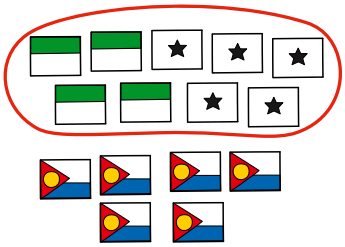
Les étapes du processus sont représentées dans le tableau suivant ; pour avoir une meilleure vue d'ensemble, nous laissons les noms des clans de côté et mentionnons seulement la taille des clans entre parenthèses :



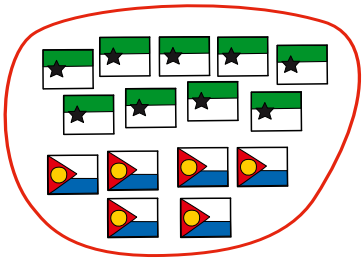
(1) et (2) deviennent un seul clan (3) ; cela dure trois semaines.



(3) et (3) sont réunis en (6) en six semaines.



Maintenant, c'est le tour de (4) et (5) de former (9) en neuf semaines.



Finalement, les deux clans (6) et (9) restant sont réunis. Cela dure quinze semaines.

Le processus est donc terminé après $15 + 9 + 6 + 3 = 33$ semaines, ce qui correspond à la réponse B).

C'est de l'informatique !

Cet exercice est un *problème d'optimisation* — une tâche consistant à minimiser (dans ce cas, la durée du rituel d'union) ou maximiser une valeur. Souvent, des algorithmes sont utilisés pour cela, dont on ne peut ajourd'hui plus se passer pour des méthodes de production écologiques qui économisent les ressources, par exemple

Dans notre cas, le problème peut être résolu de manière optimale à l'aide d'un simple *algorithme glouton*. Celui-ci fait à chaque étape ce qui semble être le mieux d'après un critère simple. Dans notre cas, le critère est que les clans doivent être les plus petits parce que cela prend moins de temps.

Mots clés et sites web

- Optimisation - `https://fr.wikipedia.org/wiki/Optimisation_(mathématiques)`
- Algorithme glouton - `https://fr.wikipedia.org/wiki/Algorithme_glouton`

A. Auteur·e·s des exercices

Daumilas Ardickas
Sarah Atkins
Michael Barot
Liam Baumann
Wilfried Baumann
Linda Björk Bergsveinsdóttir
Javier Bilbao
Lucia Budinská
Carmel Carroll
Sarah Chan
Anton Chukhnov
Kris Coolsaet
Valentina Dagienė
Christian Datzko
Susanne Datzko
Janez Demšar
Nora A. Escherle
Lidia Feklistova
Margarita Flores-Sicich
Fabian Frei
Gerald Futschek
Jens Gallenbacher
Thomas Galler
Mark Edward M. Gonzales
Martin Guggisberg
Yasemin Gülbahar
Ezgi Arzu Güneş
Mathias Hiron
Benjamin Hirsch
Juraj Hromkovič
Andrea Hrušecká
Alisher Ikramov
Tiberiu Iorgulescu
Svetlana Jaksic
YongJu Jeon
Soojin Jun
Ungyeol Jung
Filiz Kalelioğlu
Martin Kandlhofer
Ulrich Kiesmüller
Dong Yoon Kim
Jihye Kim
Vaidotas Kinčius
Víctor Koleszar
Regula Lacher
Taina Lehtimäki
Marielle Léonard
Angélica Herrera Loyo
Tom Naughton
Mochammad Irfan Noviana
Graciela Oyhenard
Gabriela Lourdes Rodríguez Parada

Elsa Pellet

Jean-Philippe Pellet

Hannah Piper

Jonatan Pipping

Zsuzsa Pluhár

Wolfgang Pohl

Peter Rossmanith

Rodrigo Santamaría

Eljakim Schrijvers

Rosario Schunk

Tomas Šiaulys

Timur Sitdikov

Bernadette Spieler

Cuttle.org Team

Ezra Templonuevo

Ahto Truu

Troy Vasiga

Florentina Voboril

Eslam Wageed

Michael Weigend

Kyra Willekes

Hongjin Yeh

Mija Zaļūksne

B. Sponsoring : Concours 2021

http://www.haslerstiftung.ch/

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

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

Kanton Zürich
Volkswirtschaftsdirektion
Amt für Wirtschaft und Arbeit

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.phzh.ch/`
Pädagogische Hochschule Zürich

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.