INFORMATIK-BIBER SCHWEIZ
CASTOR INFORMATIQUE SUISSE
CASTORO INFORMATICO SVIZZERA

Exercices et solutions 2020

Années HarmoS 9/10

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

Éditeurs :

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

0101001101010110010010010000001001011010101001100101100101001001010000101

SSIE

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

Ont collaboré au Castor Informatique 2020

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

Cheffe de projet : Nora A. Escherle

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

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

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

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

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

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

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

INFORMATIK-BIBER SCHWEIZ
CASTOR INFORMATIQUE SUISSE
CASTORO INFORMATICO SVIZZERA

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

HASLERSTIFTUNG

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

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

Préambule

Très bien établi dans différents pays européens et plus largement à l'échelle mondiale depuis plusieurs années, le concours « Castor Informatique » a pour but d'éveiller l'intérêt des enfants et des jeunes pour l'informatique. En Suisse, le concours est organisé en allemand, en français et en italien par la SSIE, la Société Suisse pour l'Informatique dans l'Enseignement, et soutenu par la Fondation Hasler dans le cadre du programme d'encouragement « FIT in IT ».

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

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

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

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

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

Les élèves des années HarmoS 5 et 6 avaient 9 exercices à résoudre : 3 faciles, 3 moyens, 3 difficiles. Les élèves des années HarmoS 7 et 8 avaient, quant à eux, 12 exercices à résoudre (4 de chaque niveau de difficulté). Finalement, chaque autre tranche d'âge devait résoudre 15 exercices (5 de chaque niveau de difficulté).

Chaque réponse correcte donnait des points, chaque réponse fausse réduisait le total des points. Ne pas répondre à une question n'avait aucune incidence sur le nombre de points. Le nombre de points de chaque exercice était fixé en fonction du degré de difficulté :

	Facile	**Moyen**	**Difficile**
Réponse correcte	6 points	9 points	12 points
Réponse fausse	−2 points	−3 points	−4 points

Utilisé au niveau international, ce système de distribution des points est conçu pour limiter le succès en cas de réponses données au hasard.

Chaque participant·e obtenait initialement 45 points (ou 27 pour la tranche d'âge « Petit Castor », et 36 pour les années HarmoS 7 et 8).

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

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

Pour de plus amples informations :

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

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

Table des matières

Ont collaboré au Castor Informatique 2020 i

Préambule iii

Table des matières v

1. Sudoku boisé 3×3 1

2. Prochain arrêt, gare ! 5

3. Quartier coloré 7

4. Abeille assidue 11

5. Serpents et échelles 15

6. Lourdes comparaisons 19

7. Bracelet céleste 23

8. Appareils ménagers 27

9. Excursion de groupe 31

10. Réseau ferroviaire 35

11. Séquence ADN 39

12. Fred le têtu 41

13. Heure de pointe 45

14. L'archipel des castors 49

15. Chauffage au sol 53

A. Auteur·e·s des exercices 56

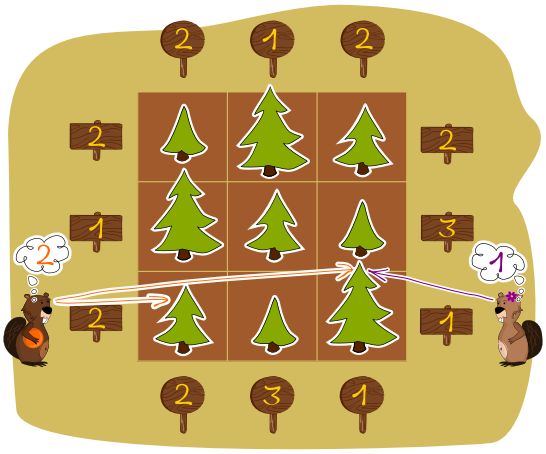
B. Sponsoring : Concours 2020 58

C. Offres ultérieures 60

1. Sudoku boisé 3×3

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

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



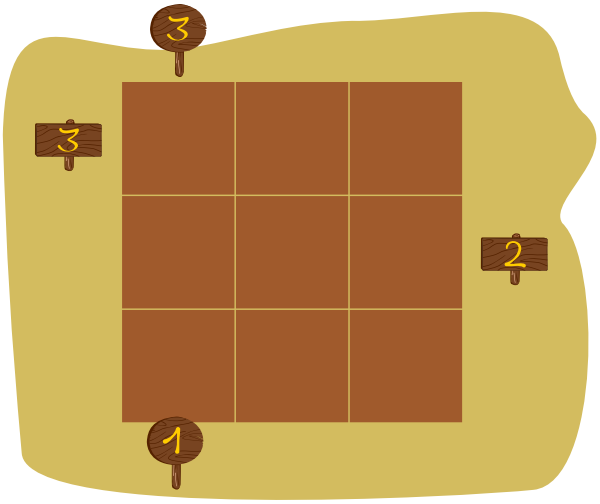
C'est écrit sur un panneau au bout de chaque rangée combien de sapins l'on peut voir depuis cet endroit-là.

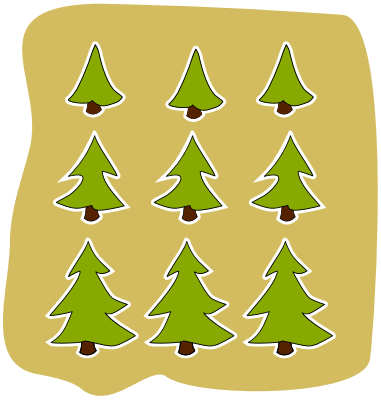
Les castors plantent à présent neuf sapins sur un champ de 3×3 cases, comme dans l'exemple à droite.

Pour cela, les règles suivantes s'appliquent :

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

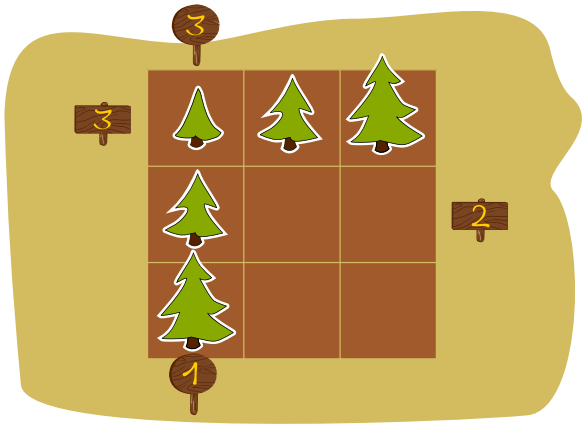
Écris dans chaque case la hauteur du sapin qui s'y trouve.





Solution

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

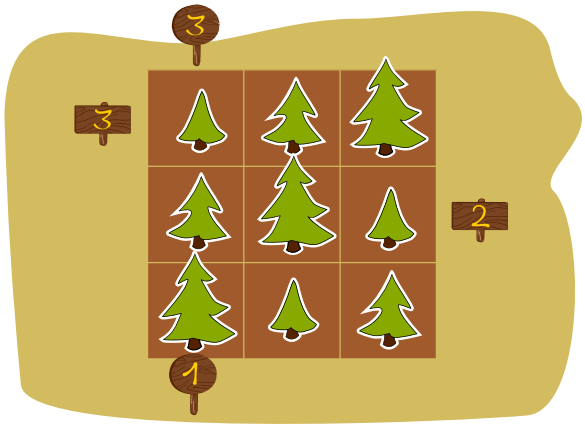


Le panneau avec le 2 à droite indique que l'on peut voir deux sapins depuis là, il doit donc y avoir un sapin de hauteur 3 au milieu et la ligne centrale est ainsi 2 (), 3 (), 1 ().

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

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

Voici la solution complète :



C'est de l'informatique !

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

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

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

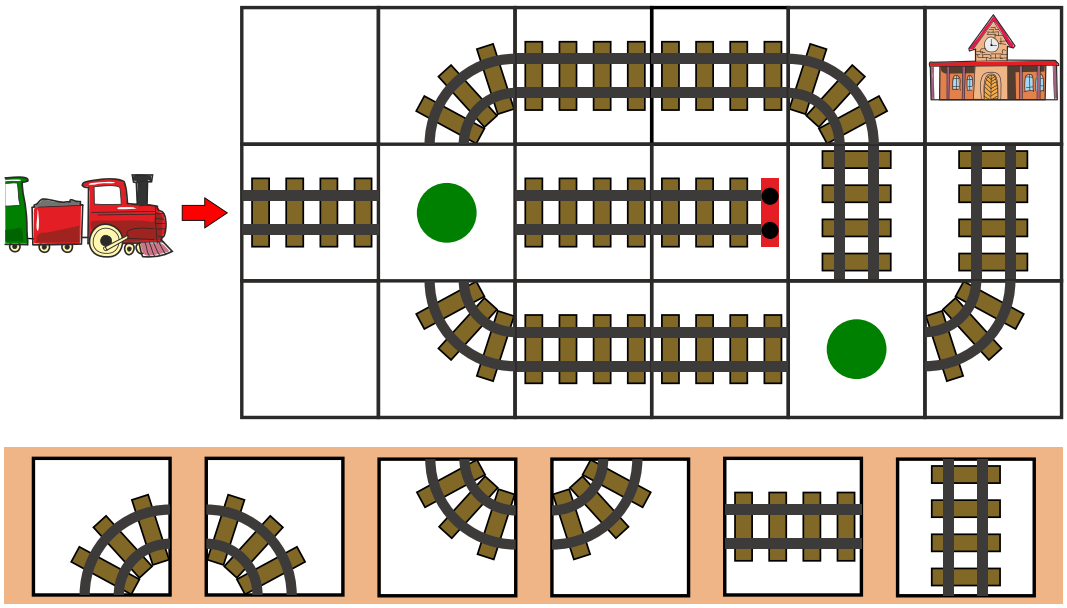
Troisièmement, on peut utiliser de tels champs d'arbres avec des panneaux pour créer des codes correcteurs. Des erreurs arrivant lors de l'entrée des données ou du transfert d'information peuvent ainsi être automatiquement reconnues ou même corrigées.

Mots clés et sites web

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

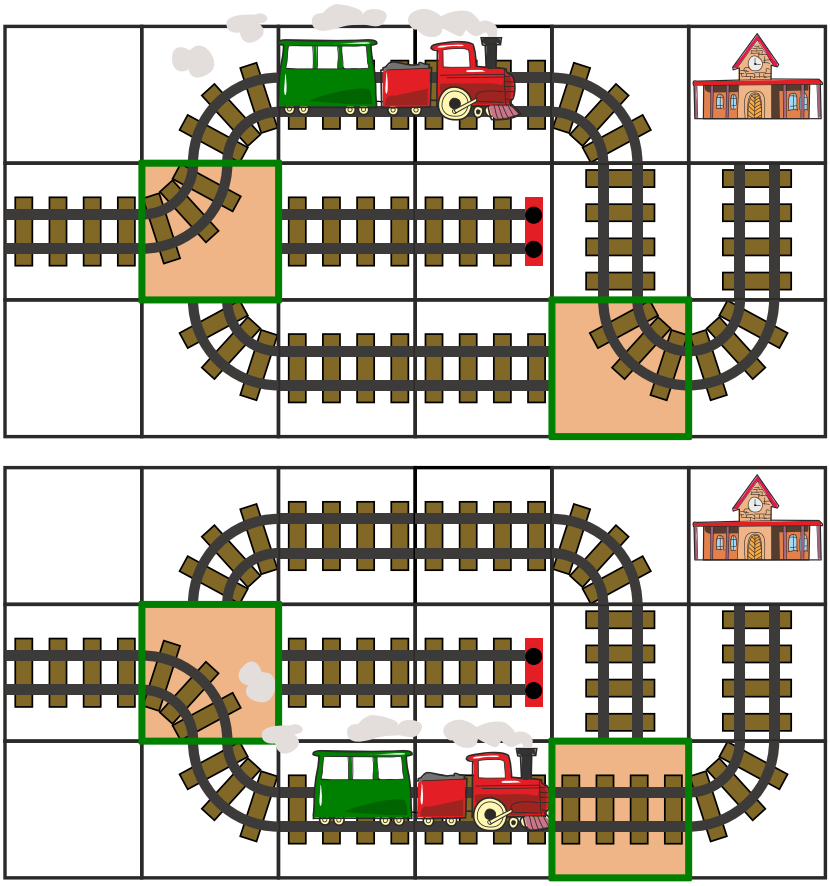
2. Prochain arrêt, gare !

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



Solution

Ce problème a les deux solutions suivantes :



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

C'est de l'informatique !

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

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

Mots clés et sites web

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

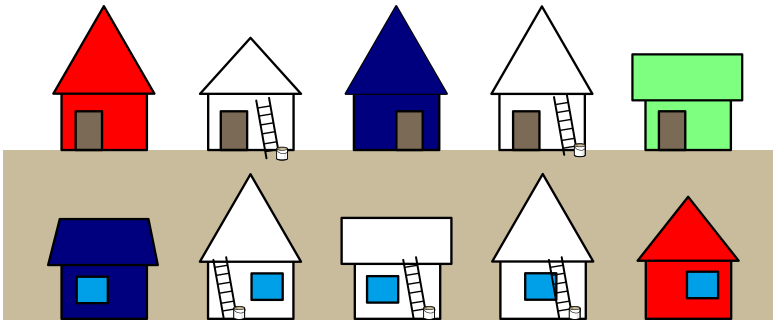
3. Quartier coloré

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

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

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

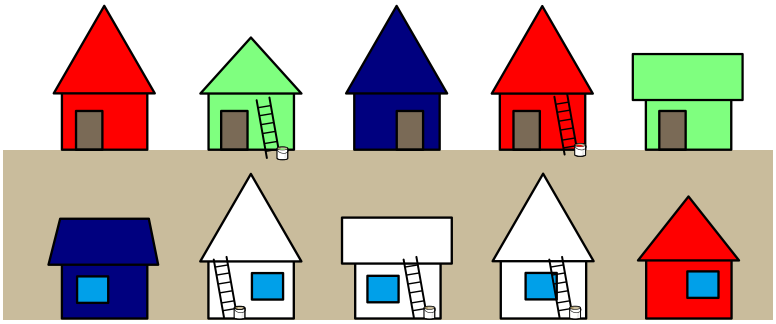
Trouve les couleurs appropriées pour les habitants.



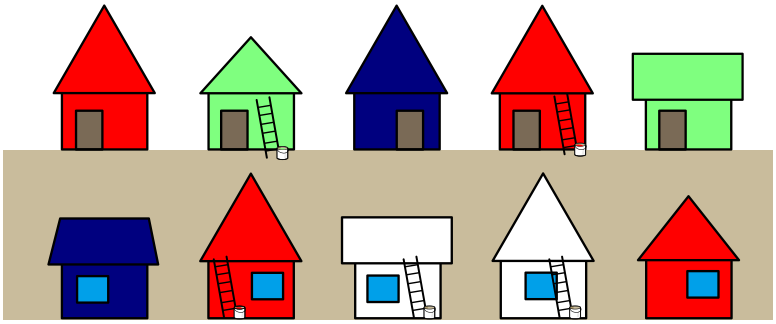
Solution

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

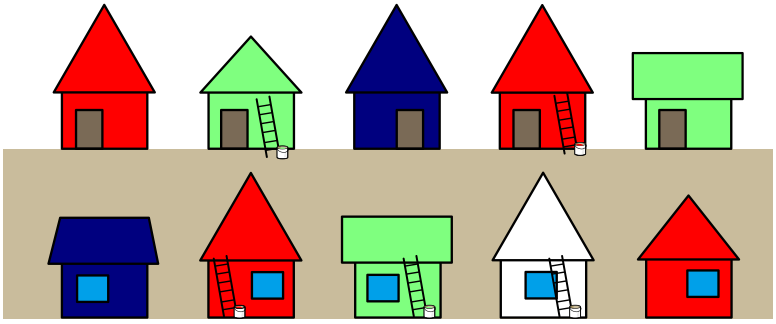
Les deux maisons blanches du côté supérieur de la route sont chacune entourées de deux maisons de couleurs différentes à gauche et à droite. On ne peut donc les peindre qu'en une seule couleur si l'on suit les règles : La maison blanche en haut à gauche en vert clair et la maison blanche en haut à droite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



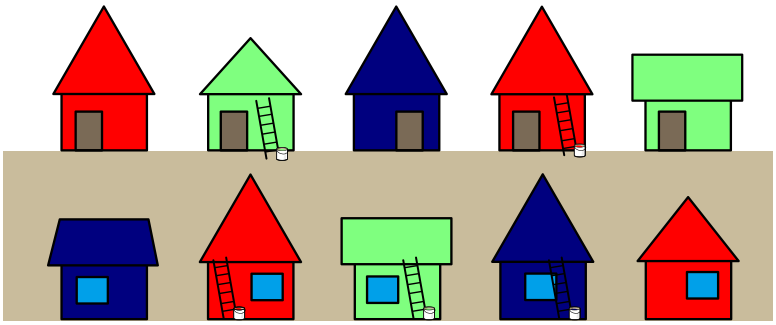
Ensuite, on peut voir que la maison blanche en bas à gauche doit être peinte en rouge, car la maison directement à sa gauche est bleu foncé et la maison directement en face est vert clair :



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



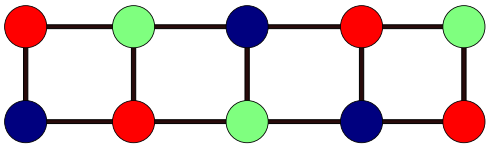
Finalement, on peut aussi trouver la couleur appropriée pour la maison blanche en bas à droite : la maison directement à sa droite et celle directement en face sont les deux rouges, mais comme la maison directement à sa gauche est maintenant vert clair, il ne reste plus que la possibilité de peindre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



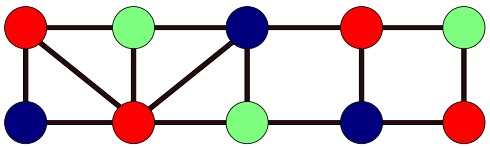
C'est de l'informatique !

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

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



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



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

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

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

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

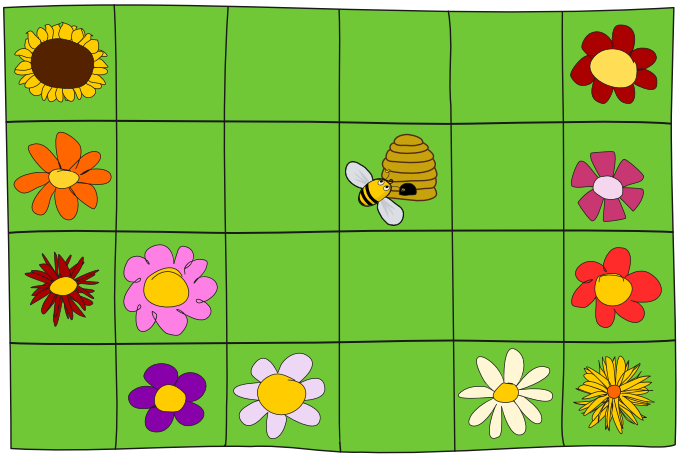
Mots clés et sites web

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

4. Abeille assidue

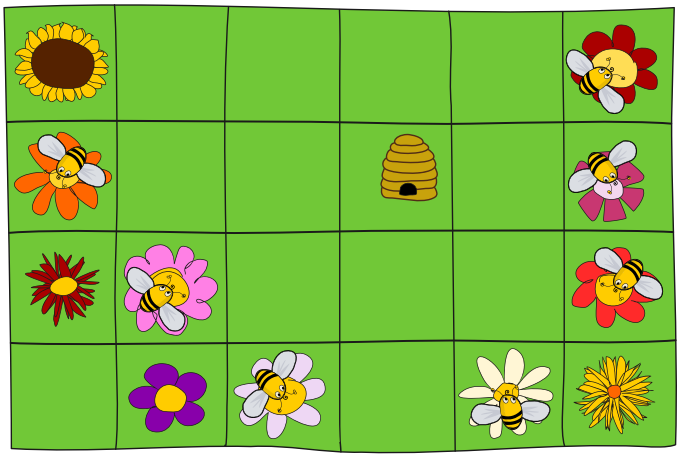
Une abeille met 10 minutes pour voler d'une case vers le haut, le bas, la gauche ou la droite. Après être partie de la ruche, elle vole pendant 30 minutes au maximum avant de revenir en arrière.

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



Solution

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



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

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

C'est de l'informatique !

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

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

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

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

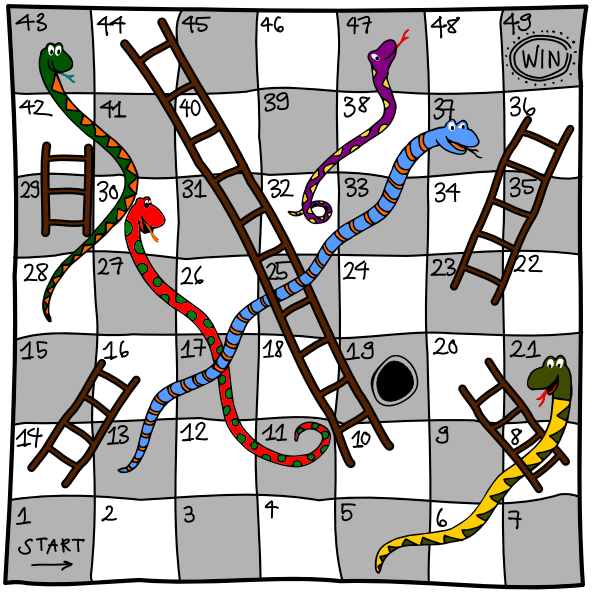
La mesure de distance habituelle qui génère un cercle s'appelle *distance euclidienne*. La mesure de distance utilisée dans l'exercice avec laquelle on ne peut se déplacer que verticalement ou horizontalement sur des carrés s'appelle la *distance de Manhattan* (le nom vient de l'arrangement quadrillé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

Mots clés et sites web

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

5. Serpents et échelles

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



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

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

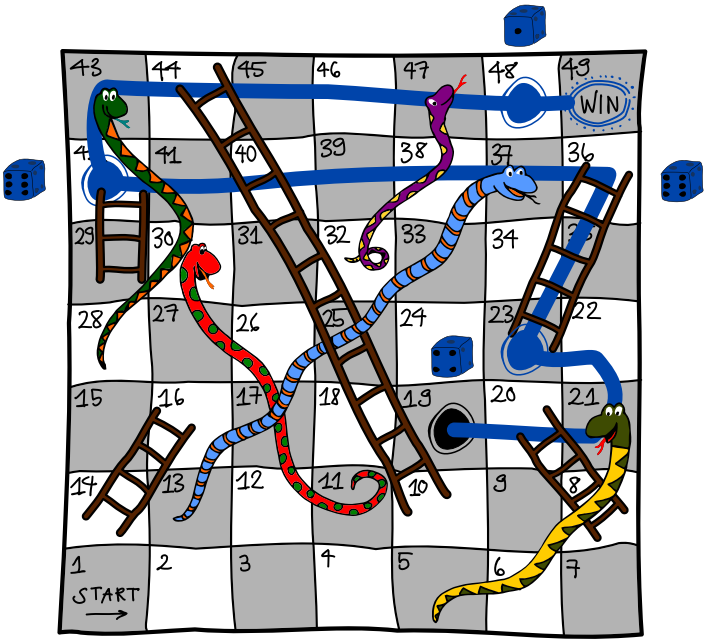
Ton pion est sur la case 19. De combien de tours au minimum as-tu besoin pour atteindre la case 49 ?

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

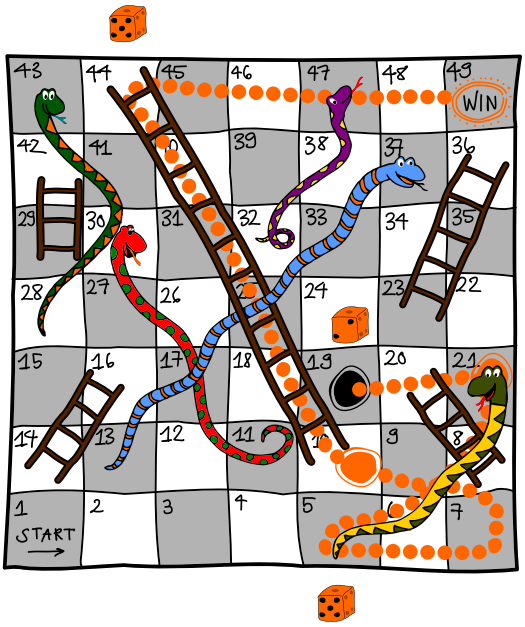
Solution

La bonne réponse est B) 3 tours.

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



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



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

C'est de l'informatique !

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

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

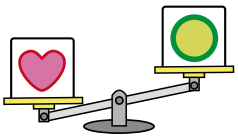
Mots clés et sites web

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

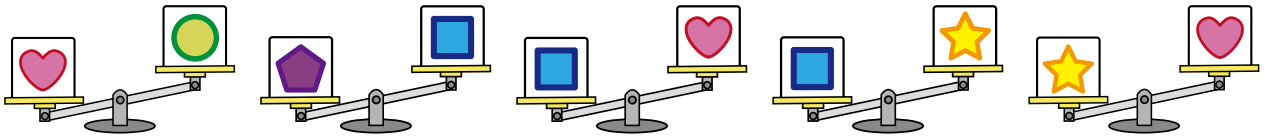
6. Lourdes comparaisons

Cinq boîtes sont marquées de cinq symboles différents :  .

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



En tout, cinq comparaisons ont lieu :



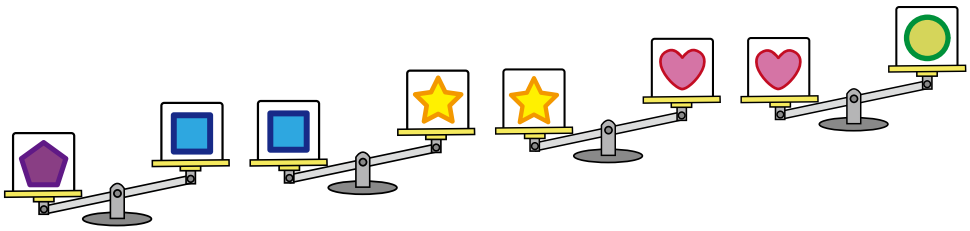
Quelle est la boîte la plus lourde ?

A) 
B) 
C) 
D) 
E) 

Solution

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

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



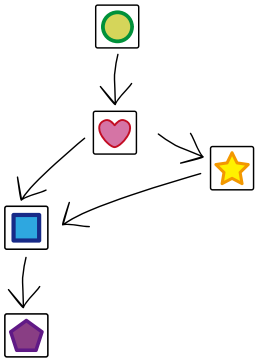
Comme cela, on voit tout de suite que : la boîte avec le pentagone est plus lourde que la boîte avec le carré. La boîte avec le carré est plus lourde que la boîte avec l'étoile. La boîte avec l'étoile est plus lourde que la boîte avec le cœur. Et finalement, la boîte avec le cœur est plus lourde que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

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

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

C'est de l'informatique !

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



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

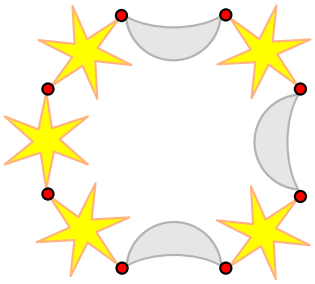
facilement en enlevant étape par étape un object sur lequel n'arrive aucune flèche, puis en mettant les objects ainsi enlevés les uns après les autres dans le même ordre.

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

Mots clés et sites web

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

7. Bracelet céleste



Marie aimerait le bracelet à droite.

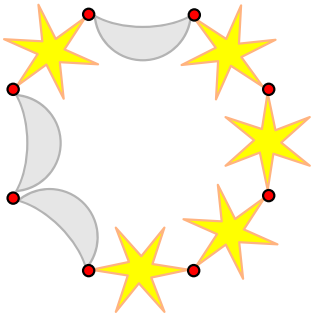
Elle donne donc les instructions suivantes à Jonas :

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

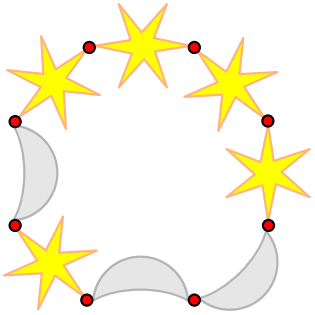
Jonas n'a pas d'image du bracelet désiré. C'est possible que Jonas obtienne un bracelet complètement différent, même s'il suit exactement les instructions de Marie.

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

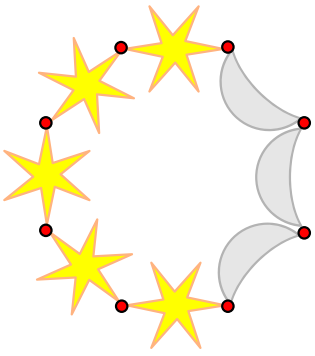
A)



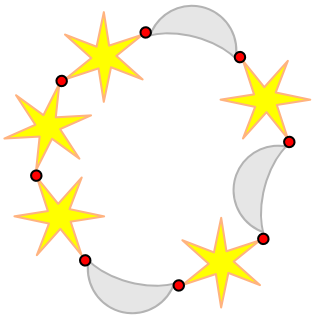
B)



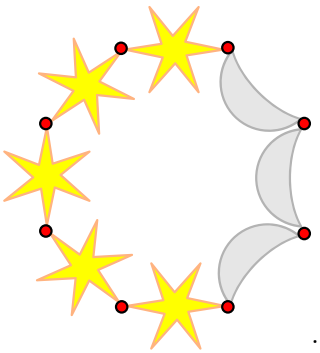
C)



D)

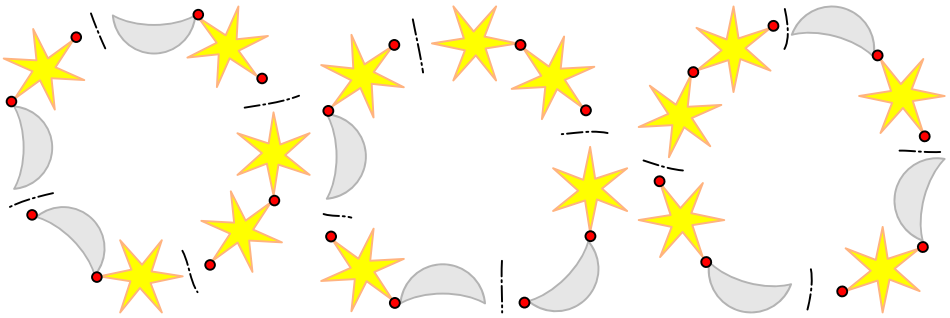


Solution

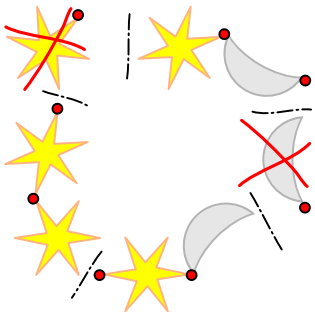
La bonne réponse est C)  .

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

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



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



C'est de l'informatique !

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

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

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

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

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

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

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

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

Mots clés et sites web

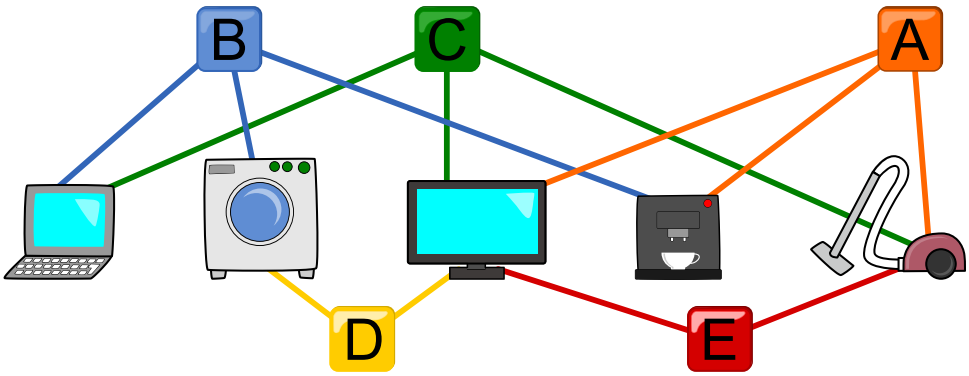
- Grammaire formelle : `https://fr.wikipedia.org/wiki/Grammaire_formelle`

8. Appareils ménagers

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

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

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



Solution

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

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

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

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

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

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

C'est de l'informatique !

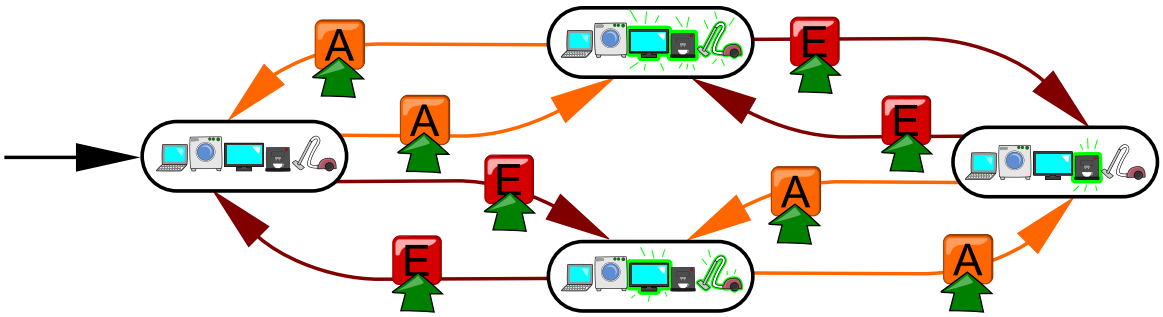
Le sytème composé des appareils et des interrupteurs peut être modélisé à l'aide d'un *automate fini* de la façon suivante.

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

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

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

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



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

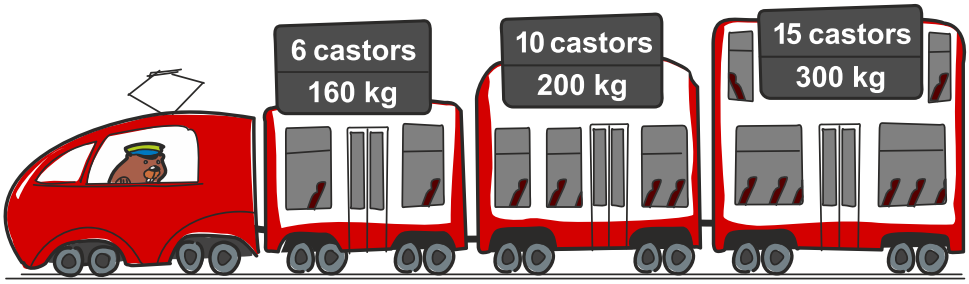
Mots clés et sites web

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

9. Excursion de groupe

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

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



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

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

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

Solution

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

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

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

C'est de l'informatique !

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

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

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

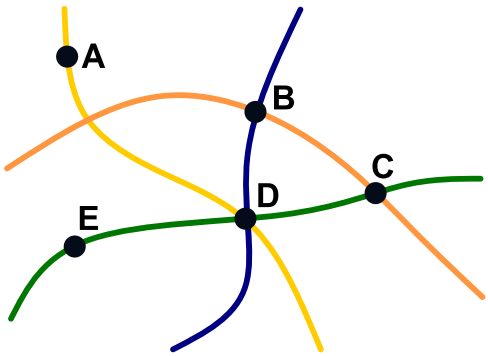
Mots clés et sites web

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

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

10. Réseau ferroviaire

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

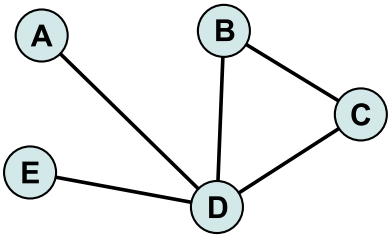


Un diagramme doit représenter cette carte de manière à ce que :

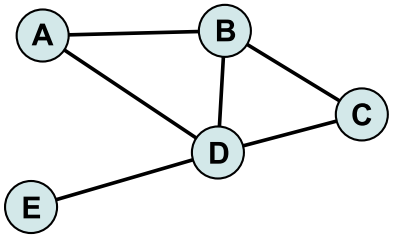
- les villes soient représentées par des cercles ;
- deux villes soient reliées d'un trait si elles sont situées sur la même ligne de train.

Quel diagramme représente la carte correctement ?

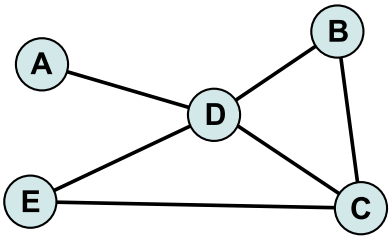
A)



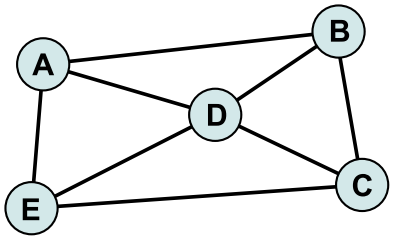
B)



C)

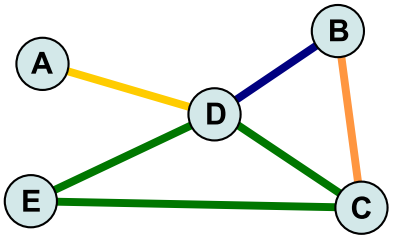


D)



Solution

La bonne réponse est C).



En observant bien la carte, on voit que :

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

Toutes les autres réponses sont fausses :

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

Il faut porter attention aux deux points suivants :

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

C'est de l'informatique !

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

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

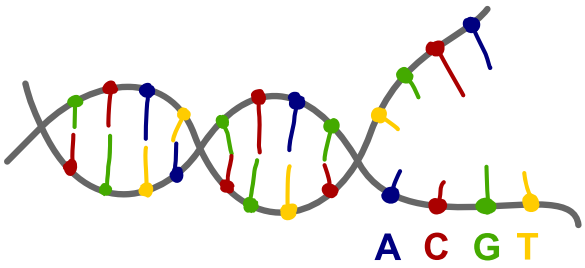
de train. On perd par contre l'information présente sur la carte que l'on passe par la ville D en allant de C à E avec cette ligne de train.

Mots clés et sites web

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

11. Séquence ADN

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



Nous considérons les trois sortes de mutations suivantes :

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

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

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

Solution

La bonne réponse est D) GGTAAAC.

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

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

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

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

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

C'est de l'informatique !

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

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

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

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

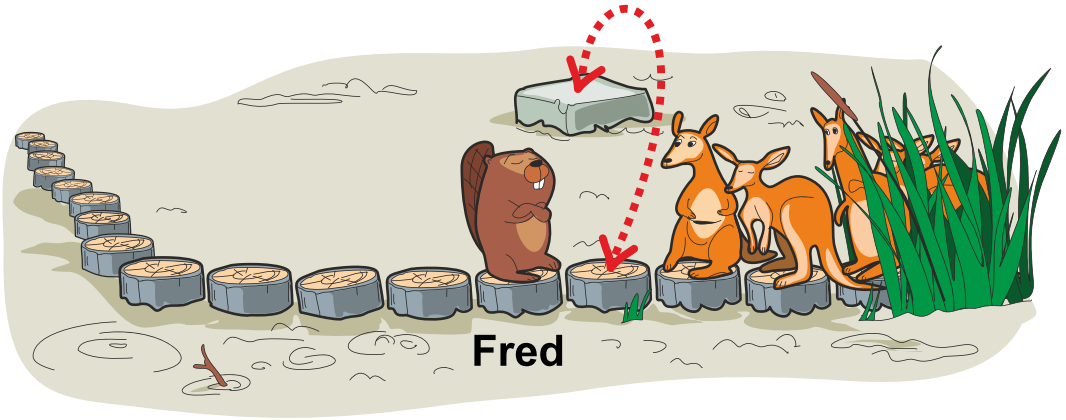
Mots clés et sites web

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

12. Fred le têtu

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

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



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

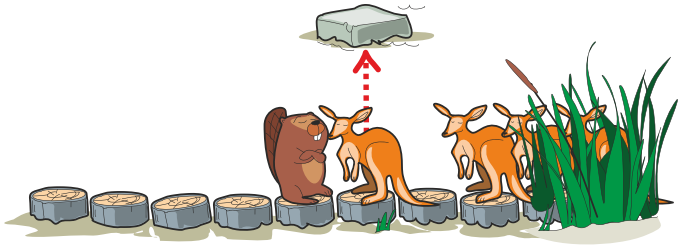
A) Plus de 10 kangourous
B) Exactement 10 kangourous
C) Exactement 6 kangourous
D) Exactement 4 kangourous
E) Moins de 4 kangourous
F) On ne peut pas savoir exactement

Solution

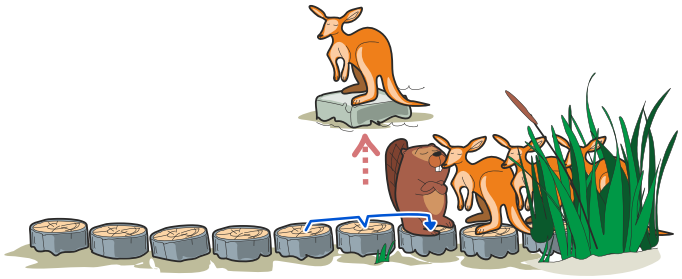
Fred peut laisser passer au maximum 6 kangourous.

Un kangourou dépasse Fred de la manière suivante :

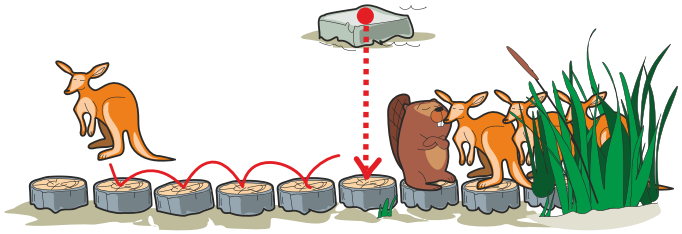
Le kangourou saute sur la pierre.



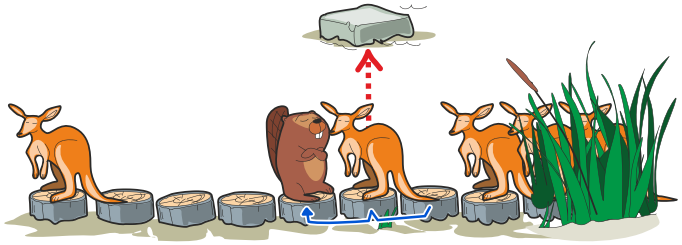
Fred avance de deux rondins.



Le kangourou revient sur le chemin en sautant et continue.



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



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

C'est de l'informatique !

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

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

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

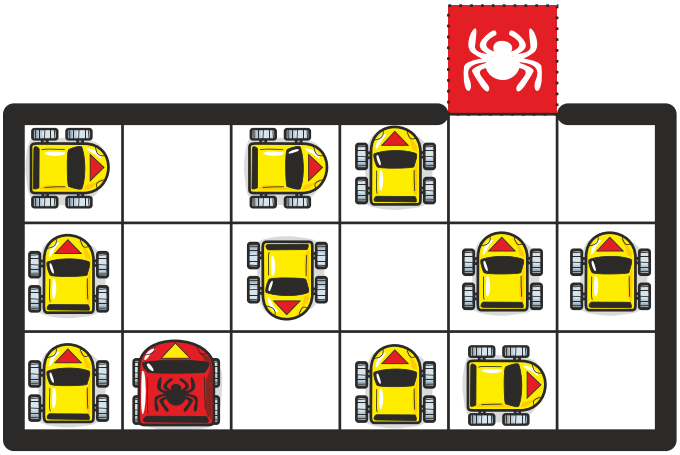
Mots clés et sites web

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

13. Heure de pointe

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

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



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

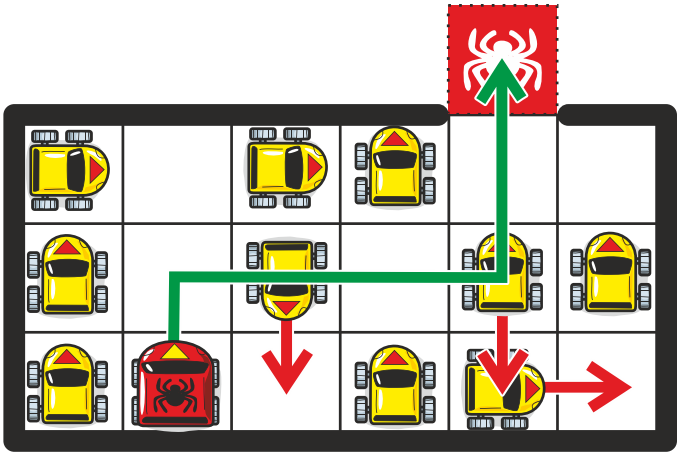
Combien de déplacements de voitures sont nécessaires pour amener la voiture rouge marquée d'une araignée sur la case rouge araignée ?

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

Solution

La bonne réponse est B) 11 déplacements.

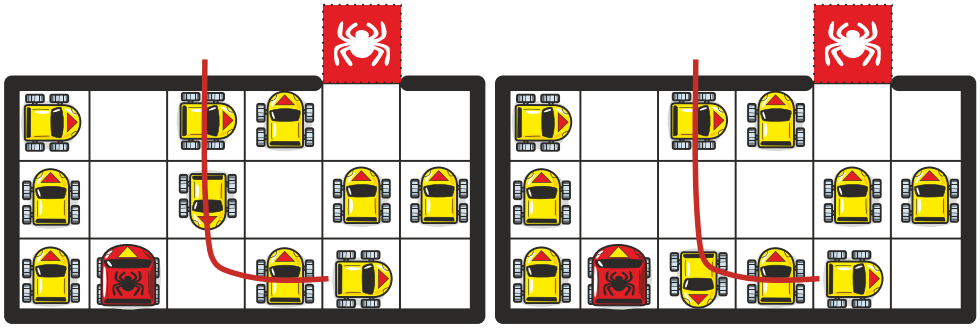
L'image montre les 11 déplacements nécessaires pour amener la voiture rouge à la case araignée :



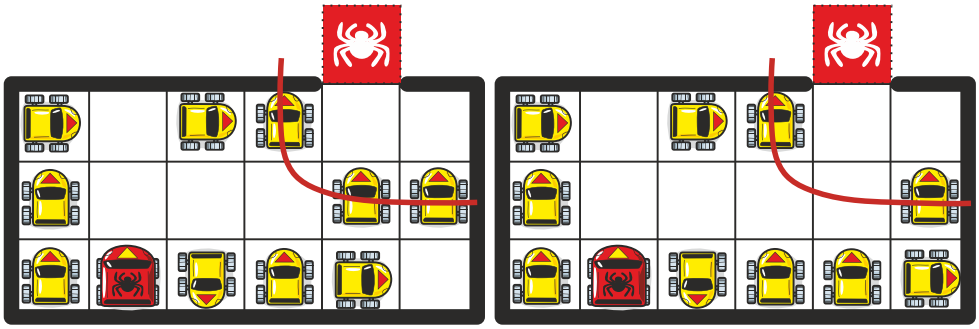
Il faut encore montrer que 11 est le nombre de déplacements minimal.

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

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



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



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

C'est de l'informatique !

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

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

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

Mots clés et sites web

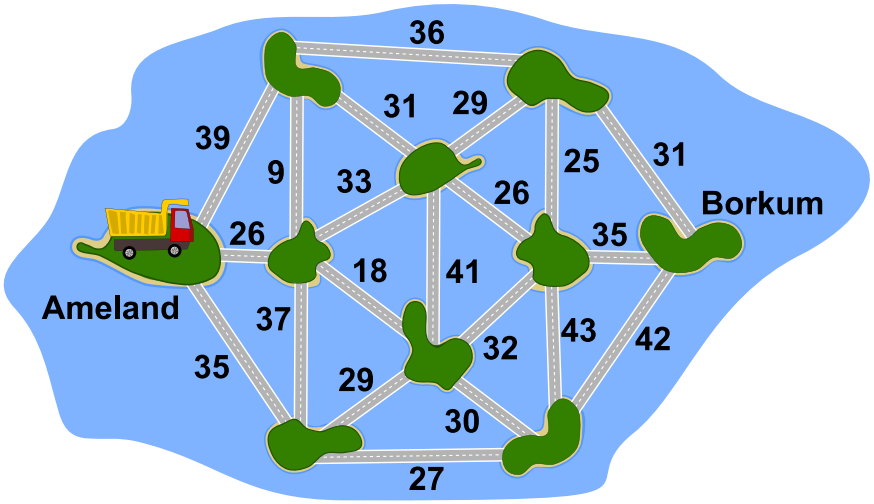
- Force brute : `https://fr.wikipedia.org/wiki/Recherche_exhaustive`
- Séparation et évaluation : `https://fr.wikipedia.org/wiki/Séparation_et_évaluation`
- Algorithme glouton : `https://fr.wikipedia.org/wiki/Algorithme_glouton`
- Rush Hour : `https://fr.wikipedia.org/wiki/Rush_hour_(casse-tête)`

14. L'archipel des castors

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

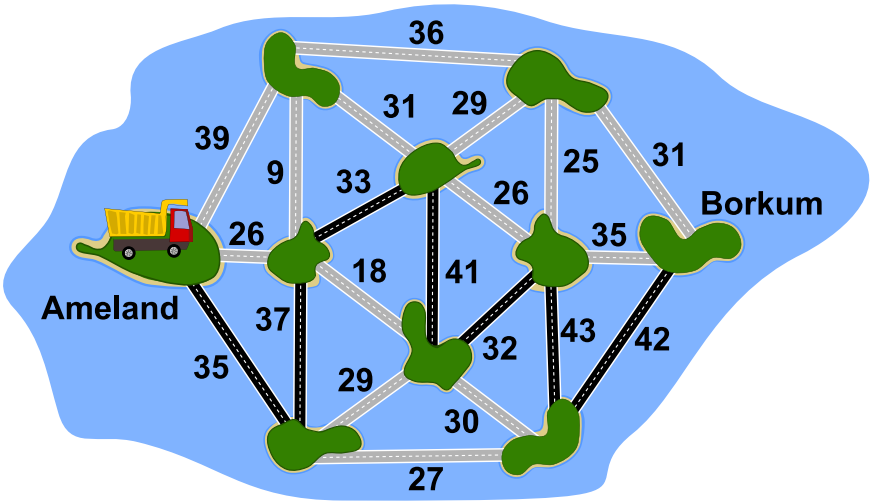
Le castor Knuth aimerait amener du sable sur une plage de l'île de Borkum. Il veut donc transporter autant de sable que possible de l'île d'Ameland à l'île de Borkum en un seul voyage. La longueur de la route à parcourir lui est égale, mais il ne veut prendre aucun pont plus d'une fois.

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

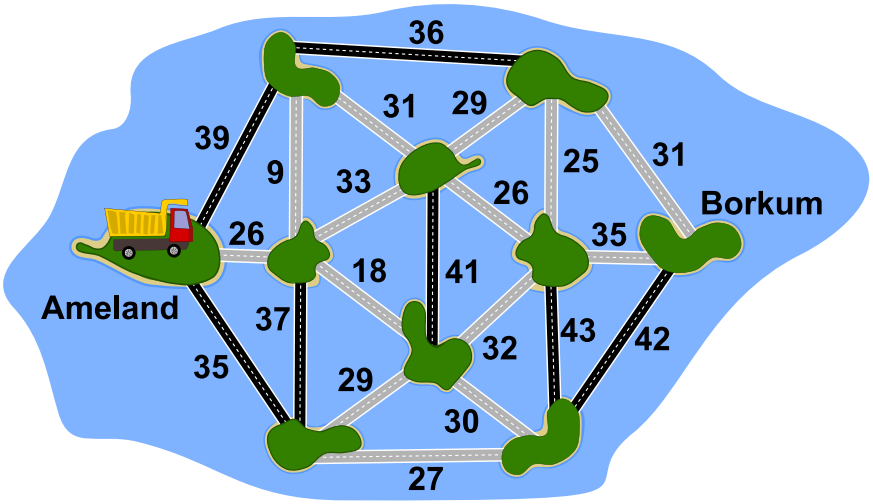


Solution

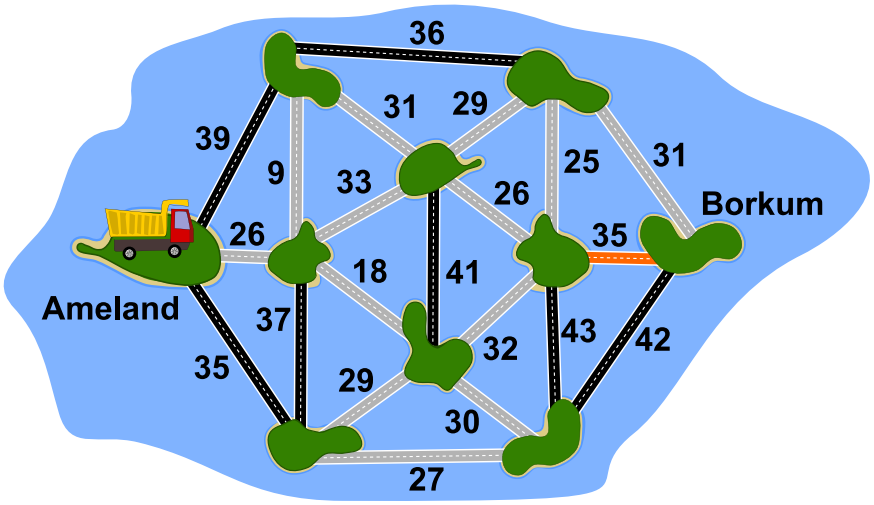
Le poids maximal d'un camion pour ce voyage est de 32 tonnes. Il suit par exemple la route suivante :



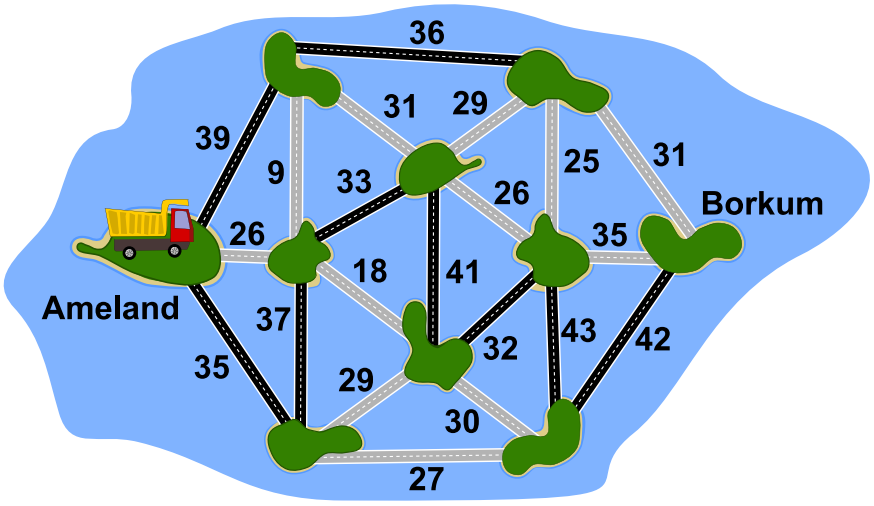
Afin de trouver ce chemin, nous pouvons par exemple commencer par virtuellement enlever tous les ponts de la carte. Nous les trions par capacité de charge. Nous commençons par ajouter à la carte les ponts ayant la plus grande capacité, puis ceux ayant une capacité de charge moindre, et ainsi de suite.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



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



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



C'est de l'informatique !

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

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

Mots clés et sites web

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

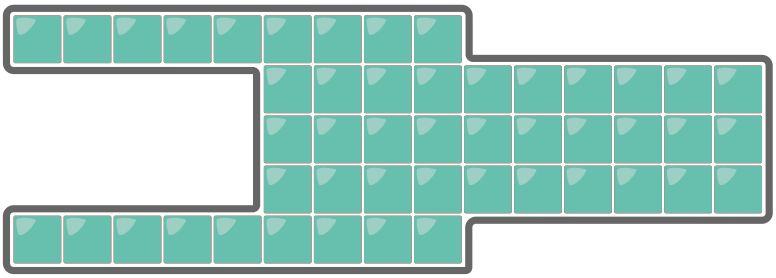
15. Chauffage au sol

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

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

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

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

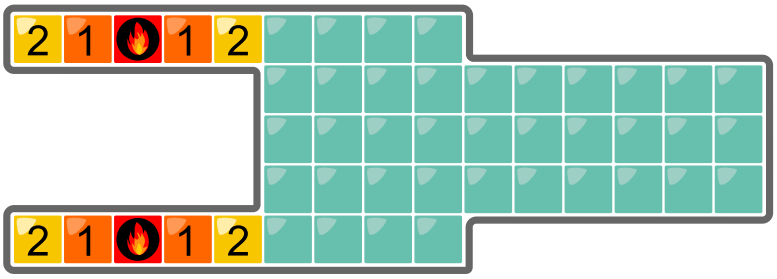


Solution

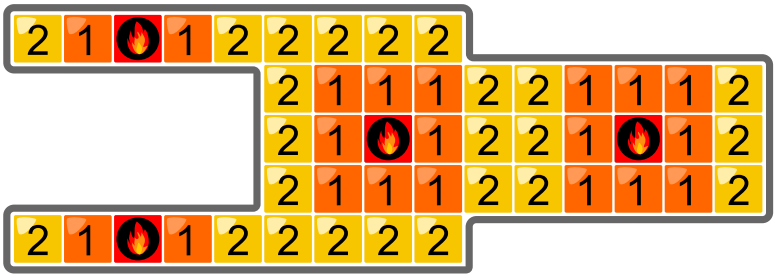
Lorsque les quatre hotspots sont installés comme dans l'image ci-dessous, toutes les catelles de la salle de bain sont chaudes deux minutes après avoir allumé le chauffage.

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

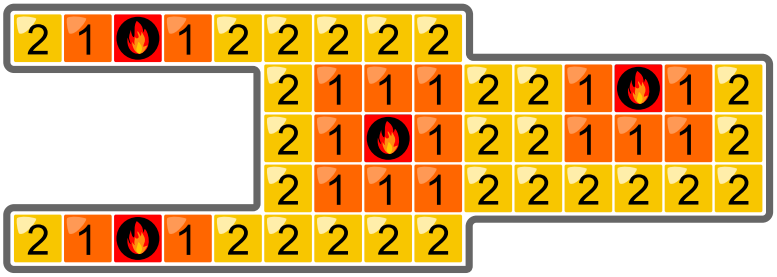
Il serait maintenant logique de commencer à répartir les hotspots dans les deux couloirs à gauche. En mettant un hotspot au milieu de chaque couloir, toutes les catelles des couloirs sont chauffées au bout de deux minutes :

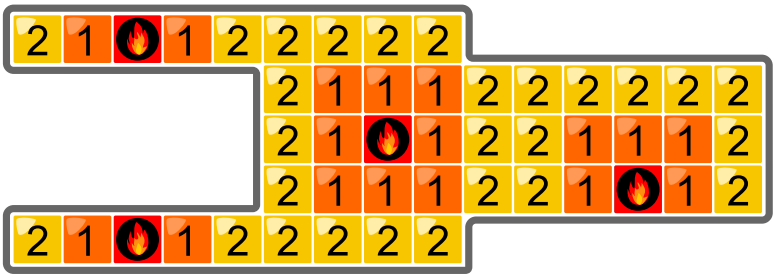


On peut placer les deux autres hotspots comme cela :



Les deux placements suivants sont également possibles :





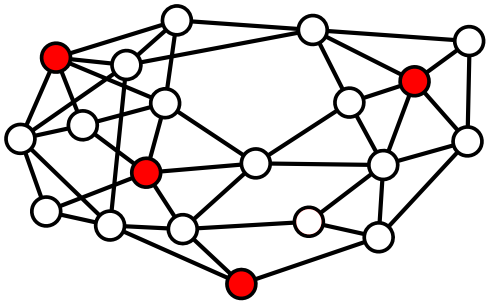
Si la salle de bain avait une forme différente, deux hotspots pourraient suffire à chauffer la même surface en deux minutes.

C'est de l'informatique !

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

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

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



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

Mots clés et sites web

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

A. Auteur·e·s des exercices

Faisal Al-Sudani
Michael Barot
Carlo Bellettini
Linda Björk Bergsveinsdóttir
Maksim Bolonkin
Andrey Brodnik
Lucia Budinská
Špela Cerar
Sarah Chan
Marios O. Choudary
Kris Coolsaet
Valentina Dagienė
Christian Datzko
Susanne Datzko
Hanspeter Erni
Fabian Frei
Gerald Futschek
Jens Gallenbacher
Yasemin Gulbahar
Mathias Hiron
Juraj Hromkovič
Tiberiu Iorgulescu
Takeharu Ishizuka
Mile Jovanov
Ungyeol Jung
Vaidotas Kinčius
Ritambhra Korpal
Regula Lacher
Marielle Léonard
Hiroki Manabe
Pedro Marcelino
Kwangsik Moon
Anna Morpurgo
Xavier Muñoz
Hiroyuki Nagataki
Vania Natali
Rana R. Natawigena
Andrei Nicolicioiu
Dejan Ozebek
Gabriel Parriaux
Elsa Pellet
Jean-Philippe Pellet
Melinda Phelps
Margot Phillipps
Hannah Piper
Wolfgang Pohl
Prathyush Ponnekanti
Raymond Chandra Putra
Susannah Quidilla
Pedro Ribeiro
Chris Roffey
Peter Rossmanith

 Eljakim Schrijvers

 Vipul Shah

 Maiko Shimabuku

 Timur Sitdikov

 Emil Stankov

 Preethi Sudharsha

 Maciej M. Sysło

 Peter Tomcsányi

 Monika Tomcsányiová

 Troy Vasiga

 Michael Weigend

 Khairul Anwar Mohamad Zaki

B. Sponsoring : Concours 2020

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.zhdk.ch/`
Zürcher Hochschule der Künste

C. Offres ultérieures

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