INFORMATIK-BIBER SCHWEIZ
CASTOR INFORMATIQUE SUISSE
CASTORO INFORMATICO SVIZZERA

Exercices et solutions 2023

Années HarmoS 9/10

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

Éditeurs :

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

SSIE

www.svia-ssie-ssii.ch
schweizerischerverein für informatikind erausbildung//société suisse pour l'information matiquedans l'enseignement//società svizzera per l'informatica nell'insegnamento

HASLERSTIFTUNG ETH zürich | ABZ AUSBILDUNGS- UND BERATUNGSZENTRUM FÜR INFORMATIKUNTERRICHT hep/ Scuola universitaria professionale della Svizzera italiana SUPSI

Ont collaboré au Castor Informatique 2023

Masiar Babazadeh, Susanne Datzko-Thut, Jean-Philippe Pellet, Giovanni Serafini, Bernadette Spieler

Cheffe de projet : Nora A. Escherle

Nous adressons nos remerciements pour le travail de développement des exercices du concours à :
Juraj Hromkovič, Angélica Herrera Loyo, Regula Lacher, Manuel Wettstein : ETH Zurich, Ausbildunges- und Beratungszentrum für Informatikunterricht
Tobias Berner : Pädagogische Hochschule Zürich
Christian Datzko : Wirtschaftgymnasium und Wirtschaftsmittelschule, Basel
Fabian Frei : CISPA - Helmholtz-Zentrum für Informationssicherheit
Sebastian Knüsli : Gymnasium Kirschgarten, Basel

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, Vaidotas Kinčius : Bebras.org, Lituanie
Wolfgang Pohl, Jakob Schilke : Bundesweite Informatikwettbewerbe (BWINF), Allemagne
Hannes Endreß : Materna Information & Communications SE, Allemagne
Ulrich Kiesmüller : Simon-Marius-Gymnasium Gunzenhausen, Allemagne
Kirsten Schlüter : Bayerisches Staatsminiserium für Unterricht und Kultus, Allemagne
Margareta Schlüter : Universität Tübingen, Allemagne
Jacqueline Staub : Universität Trier, Allemagne
Michael Weigend : WWU Münster, Allemagne
Wilfried Baumann, Liam Baumann, Josefine Hiebler : Österreichische Computer Gesellschaft, Autriche
Gerald Futschek : Technische Universität Wien, Autriche
Zsuzsa Pluhár : ELTE Informatikai Kar, Hongrie

La version en ligne du concours a été réalisée sur l'infrastructure cuttle.org. Nous remercions pour la bonne collaboration :
Eljakim Schrijvers, Justina Dauksaite, Arjan Huijsers, Dave Oostendorp, 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
Gabriel Thullen : Collège des Colombières, Versoix

Nous remercions les personnes suivantes pour l'organisation et la réalisation de la finale suisse :
Dennis Komm, Hans-Joachim Bückenhauer, Jan Lichensteiger, Moritz Stocker : ETH Zurich, Ausbildunges- und Beratungszentrum für Informatikunterricht
Pour la correction des épreuves :
Fiona Binder, Joel Birrer, Marlene Bötschi, Danny Camenisch, Gianluca Danieletto, Alexander Frey, Sven Grübel, Laure Guerrini, Charlotte Knierim, Richard Královič, Yanik Künzi, Kenli Lao, Sandro

Marchon, Zoé Meier, Dario Näpfer, Kai Zürcher
Pour la traduction française des épreuves :
Jan Schönbächler : Lycée-Collège de l'Abbaye de St-Maurice

Christoph Frei : Chragokyberneticks (Logo Castor Informatique Suisse)
Andrea Leu, Maggie Winter, Lena Frölich : Senarclens Leu + Partner AG

Des remerciements particuliers sont dûs pour leur grand soutien à Juraj Hromkovič, Dennis Komm, Gabriel Parriaux et la Fondation Hasler. Sans eux, ce concours n'existerait pas.

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

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

INFORMATIK-BIBER SCHWEIZ
CASTOR INFORMATIQUE SUISSE
CASTORO INFORMATICO SVIZZERA

Le Castor Informatique 2023 a été réalisé par la Société Suisse pour l'Informatique dans l'Enseignement (SSIE) et soutenu de manière déterminante par la Fondation Hasler. Les sponsors du concours sont l'Office de l'économie et du travail du canton de Zurich et l'UBS.

Cette brochure a été produite le 10 janvier 2024 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 2023.

Préambule

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

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

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

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

Le concours Castor Informatique 2023 a été fait pour cinq tranches d'âge, basées sur ces 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

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

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

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

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

Chaque participant·e obtenait initialement 45 points (ou 27 pour 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 (en étant classés différemment dans les niveaux de difficulté).

Certains exercices sont indiqués comme « bonus » pour certaines catégories d'âge : ils ne comptent pas dans le total des points, mais servent à départager plusieurs scores identiques en cas de qualification pour les éventuels tours suivants.

Pour de plus amples informations :

SVIA-SSIE-SSII Société Suisse pour l'Informatique dans l'Enseignement
Castor Informatique
Jean-Philippe Pellet

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

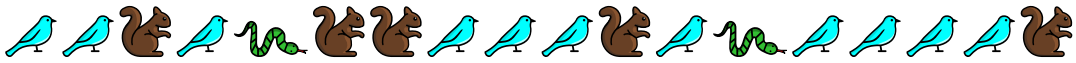
Table des matières

1. L'arbre magique

Ben a un arbre magique dans son jardin :

- Si un oiseau  se pose sur l'arbre, deux pommes y poussent tout de suite.
- Si un écureuil  grimpe sur l'arbre, une pomme en tombe. S'il n'y a pas de pomme sur l'arbre, il ne se passe rien.
- Si un serpent  vient sous l'arbre, toutes les pommes disparaissent tout de suite.

Ce matin, 25 pommes sont sur l'arbre. Plusieurs animaux rendent ensuite visite à l'arbre, un écureuil en dernier. Ben a noté leur ordre exact :



Combien de pommes y a-t-il sur l'arbre après la dernière visite ?

A) 3 pommes

B) 7 pommes

C) 17 pommes

D) 31 pommes

Solution

La bonne réponse est B. Après que le dernier écureuil est monté sur l'arbre, il y reste sept pommes.

On peut calculer combien de pommes se trouvent sur l'arbre à chaque visite d'un animal :

Animal :	**Départ**					
Instruction :	–	+2	+2	–1	+2	reset
Nombre de pommes :	25	27	29	28	30	0

Animal :	**Report**								
Instruction :	–	–	–	+2	+2	+2	–1	+2	reset
Nombre de pommes :	0	0	0	2	4	6	5	7	0

Animal :	**Report**					
Instruction :	–	+2	+2	+2	+2	–1
Nombre de pommes :	0	2	4	6	8	**7**

Comme toutes les pommes disparaissent lorsqu'un serpent vient sous l'arbre, nous pouvons ignorer tout ce qui se passe avant l'arrivée du deuxième (et dernier) serpent. Comme indiqué dans le tableau, quatre oiseaux se posent sur l'arbre après le passage du dernier serpent. Il y a ensuite $4 \times 2 = 8$ pommes sur l'arbre. Ensuite, un écureuil grimpe, faisant tomber une pomme ; il reste donc $8 - 1 = 7$ pommes sur l'arbre.

C'est de l'informatique !

La visite d'un animal modifie l'état de l'arbre magique – mais d'une manière bien définie : seul le nombre de pommes sur l'arbre change. La visite des animaux ne change pas les autres propriétés de l'arbre, comme son nombre de feuilles, la longueur de ses branches ou la forme de son tronc, par exemple. Pour cet exercice, il est donc suffisant de considérer le nombre de pommes.

Un programme informatique a lui aussi un état qui peut être modifié par les instructions du programme. On considère généralement les données stockées par le programme comme son état ; ces données sont stockées par le programme dans des *variables* définies lors de la programmation.

La suite des visites des animaux à l'arbre dans cet exercice est comme un programme informatique : chaque visite est une instruction qui change l'état de l'arbre. Cet état (ici, le nombre de pommes sur l'arbre) peut être stocké à l'aide d'une seule variable.

Tu as peut-être remarqué que tu n'avais pas besoin de considérer le « programme » en entier pour résoudre l'exercice, mais uniquement la partie suivant la dernière visite d'un serpent. En observant attentivement l'effet des différentes instructions sur l'état du programme, on peut découvrir certaines

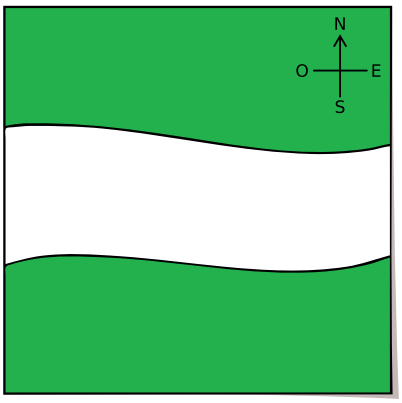
propriétés de ce programme. Une telle analyse de programmes (informatiques) fait partie des tâches des informaticiens et informaticiennes.

Mots clés et sites web

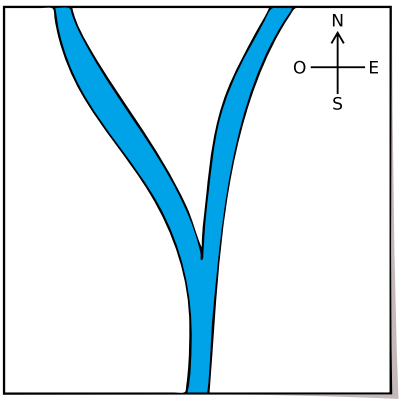
- Variable : `https://fr.wikipedia.org/wiki/Variable_(informatique)`
- État d'un programme : `https://fr.wikipedia.org/wiki/État_(informatique)#Processus`

2. La maison de Karla

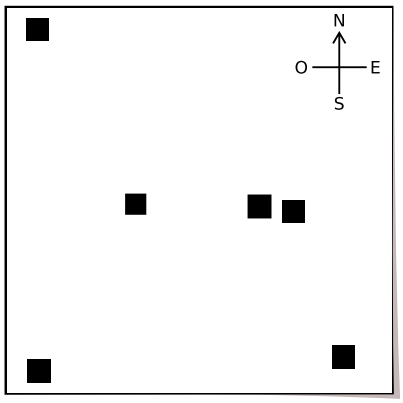
Karla a trois cartes qui montrent exactement la même région. Une carte montre les forêts ; une autre, les rivières, et la troisième, les maisons dans cette région. La maison de rêve de Karla se trouve dans la forêt et près d'une rivière.



Carte des forêts



Carte des rivières

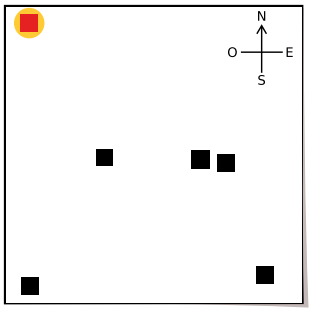


Carte des maisons

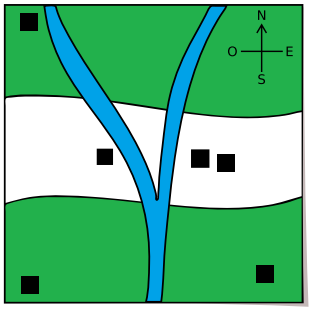
Quelle est la maison de rêve de Karla ?

Solution

La maison en haut à gauche de la carte des maisons est la maison de rêve de Karla :



Pour trouver la maison de rêve de Karla, il faut analyser les informations des trois cartes. La maison de rêve doit se trouver dans une forêt et près d'une rivière. Ce n'est vrai que pour la maison en haut à gauche. C'est facile à voir en superposant les trois cartes :



C'est de l'informatique !

Lorsque les informations sur les forêts, les rivières et les maisons sont représentées sur une seule carte, c'est facile de trouver la maison recherchée.

Un *système d'information géographique* (SIG) assemble une multitude d'informations spatiales (par exemple les forêts, routes, frontières, stations service, maisons, etc.) et les représente sur une carte. Un SIG sert donc à la visualisation et à l'analyse de *données géographiques*. Un SIG permet par exemple à la protection civile de mettre en place des plans d'évacuation.

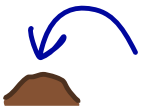
Les programmes graphiques utilisent aussi plusieurs *niveaux* avec des informations graphiques différentes (appelés *calques*). Une question importante est toujours quel niveau est le plus haut et est donc représenté au premier plan. Ici, ce sont par exemple les maisons qui doivent être au premier plan afin qu'elles ne soient pas cachées par les forêts.

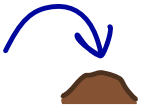
Mots clés et sites web

- SIG : `https://fr.wikipedia.org/wiki/Système_d'information_géographique`
- Calque : `https://fr.wikipedia.org/wiki/Calque_(infographie)`

3. Graines de carottes

Le robot lapin peut exécuter les instructions suivantes :


Saute vers la **gauche** sur la prochaine colline


Saute vers la **droite** sur la prochaine colline


Plante une graine de carotte sur la colline sur laquelle tu es

Le robot lapin a suivi la suite d'instructions suivante :

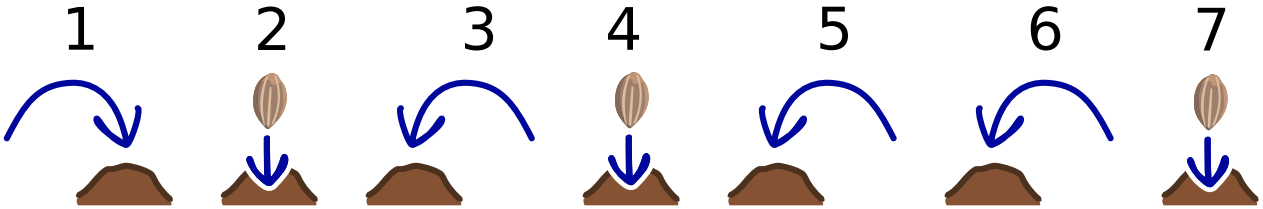
Il a passé sur quatre collines. Nous ne savons pas sur quelle colline il a commencé.

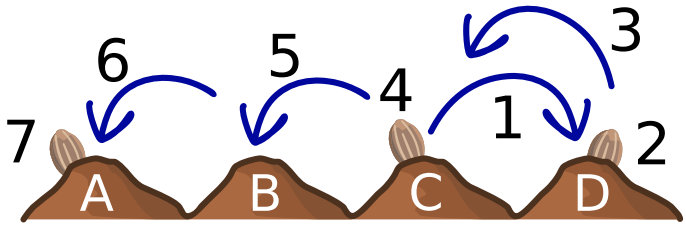
Sur quelles collines le robot a-t-il planté des graines de carottes ?

Solution

Voici la bonne réponse : 

Pour mieux pouvoir expliquer la bonne réponse, nous donnons des lettres aux collines (voir ci-dessus) et des numéros aux instructions :





Nous commençons par déterminer le point de départ du robot : avant de sauter trois fois de suite vers la gauche (instructions 3, 5 et 6), il doit se trouver sur la colline D. Avant cela, il saute une fois vers la droite (instruction 1). Le robot a donc commencé sur la colline C. Les graines de carottes sont donc plantées sur les collines D, puis C et finalement A, d'après les instructions 2, 4 et 7.

C'est de l'informatique !

Les vrais robots ont des ordinateurs intégrés, et ils sont programmés de manière similaire au robot lapin. Un programme informatique est constitué de plusieurs *instructions* individuelles.

Dans notre cas, la suite d'instructions pour l'ordinateur du robot est donnée sous forme d'images. Le résultat (*sortie*, *output* en anglais) du programme ne dépend pas que de la position de départ (*entrée*, *input* en anglais), mais aussi des instructions et de l'ordre dans lequel elles sont données.

Cet exercice du Castor illustre l'utilisation de robot dans l'agriculture. Les robots ne peuvent pas seulement planter, mais aussi arroser, polliniser et répandre des produits de protection de manière ciblée.

Mots clés et sites web

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

- Smart Farming : `https://www.agroscope.admin.ch/agroscope/fr/home/themes/economie-technique/smart-farming.html`
- Les robots et l'agriculture : `https://cordis.europa.eu/article/id/441912-robots-help-farmers-say-goodbye-to-repetitive-tasks/fr`

4. Le trésor de Barbe-de-Castor

Il y a trois coffres au trésor sur une île : un coffre se trouve au pied du volcan, le deuxième sous un palmier et le dernier sur la plage. Tous les coffres sont vides.

Un jour, le pirate Barbe-de-Castor vient sur l'île, remplit un des coffres d'or et le ferme. Le même jour, trois touristes sont sur l'île : Anita, Britta et Carla. Chacune fait une photo : l'une avant que Barbe-de-Castor ait rempli le coffre, les deux autres après.

La photo d'Anita	**La photo de Britta**	**La photo de Carla**
… montre le coffre sur la plage.	… montre les deux coffres sur la plage et sous le palmier.	… montre les deux coffres sous le palmier et au pied du volcan.
		

Tous les coffres sont vides sur les photos. Barbe-de-Castor a eu de la chance qu'aucune des touristes ne trouve son or !

Dans quel coffre au trésor se trouve l'or ?

Solution

Voici la bonne réponse :

L'or est dans le coffre au trésor au pied du volcan.

Nous vérifions pour chaque coffre si l'or peut s'y trouver. Pour cela, nous regardons si les photos correspondent à l'histoire.

1. **Le coffre sous le palmier.** Les photos de Carla et Britta montrent le coffre vide sous le palmier. Si ce coffre contenait l'or, les deux photos devraient avoir été prises avant que le coffre ne soit rempli, mais nous savons que seule une photo a été prise avant que Barbe-de-Castor n'amène son or. L'hypothèse que l'or se trouve dans le coffre sous le palmier mène donc à une contradiction. Nous pouvons en conclure qu'il n'y a pas d'or dans le coffre sous le palmier.
2. **Le coffre sur la plage.** Les photos d'Anita et de Britta montrent le coffre vide sur la plage. Si ce coffre contenait l'or, les deux photos devraient avoir été prises avant que le coffre ne soit rempli, mais nous savons que seule une photo a été prise avant que Barbe-de-Castor n'amène son or. L'hypothèse que l'or se trouve dans le coffre sur la plage mène donc à une contradiction. Nous pouvons en conclure qu'il n'y a pas d'or dans le coffre sur la plage.
3. **Le coffre au pied du volcan** n'est que sur la photo de Carla et y est vide. Si ce coffre contenait l'or, Carla pourrait être la touriste qui a pris sa photo avant que Barbe-de-Castor n'amène son or. Le coffre au pied du volcan n'est pas sur les photos d'Anita ni de Britta, qui peuvent donc être les touristes ayant pris leur photo après la visite de Barbe-de-Castor. L'hypothèse que l'or se trouve dans le coffre au pied du volcan ne mène donc à *aucune* contradiction.

Comme l'or doit se trouver dans l'un des coffres, nous pouvons en déduire qu'il se trouve dans le coffre au pied du volcan.

C'est de l'informatique !

L'*inférence* logique a été utile pour résoudre cet exercice. Nous avons utilisé trois photos et nos connaissances sur la situation sur l'île pour déterminer pourquoi certaines hypothèses pourraient être vraies ou pas. La recherche de contradictions joue un rôle important dans l'inférence logique. Lorsqu'une conclusion est une conséquence logique d'une hypothèse (ou *prémisse*), mais que l'hypothèse et la conclusion ne peuvent pas les deux être vraies en même temps, on peut être sûr que l'hypothèse est fausse.

La logique joue un rôle important dans beaucoup de domaines de l'informatique : les circuits du hardware informatique, que ce soit dans les processeurs ou les dispositifs de stockage, sont des applications d'opérateurs logiques. Des conditions complexes en prgrammation ou des recherches difficiles dans des bases de données peuvent être formulées à l'aide de relations logiques. Le comportement de programmes peut être décrit et vérifié à l'aide de calculs logiques. Les *langages de programmation logiques* travaillent directement avec des déclarations et inférences logiques pour effectuer des calculs.

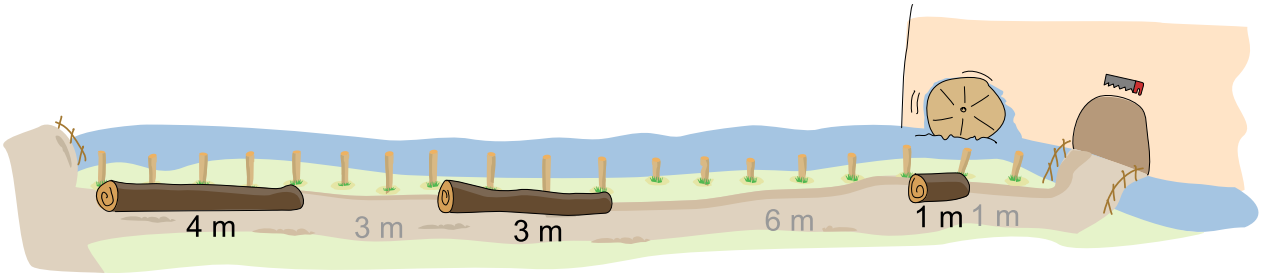
Mots clés et sites web

- Inférence : `https://fr.wikipedia.org/wiki/Inférence_(logique)`
- Déduction : `https://fr.wikipedia.org/wiki/Déduction_logique`
- Programmation logique : `https://fr.wikipedia.org/wiki/Programmation_logique`
- Prolog : `https://fr.wikipedia.org/wiki/Prolog`

5. Les troncs de Timea

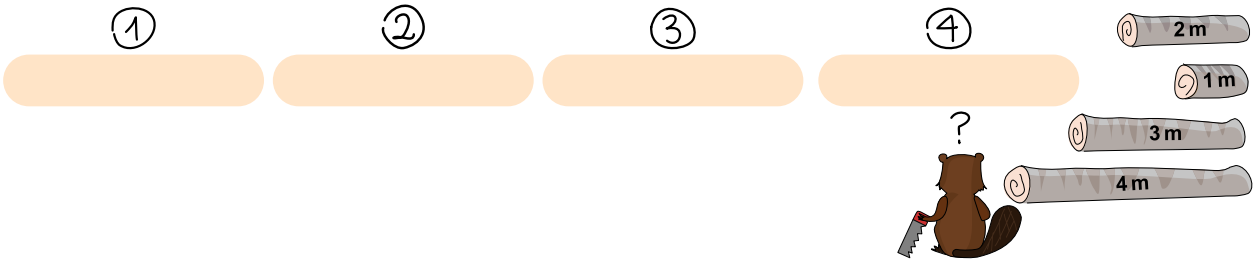
Timea la castor coupe des troncs d'arbre de différentes longueurs, puis les vend. Dès qu'elle a coupé un tronc, elle le pose sur le chemin long de 18 mètres. Timea suit pour cela la règle suivante : en commençant à gauche, elle place le tronc dans le premier espace vide assez grand pour l'y mettre.

Elle vend quelques troncs. Il y a ensuite trois espaces vides sur le chemin :



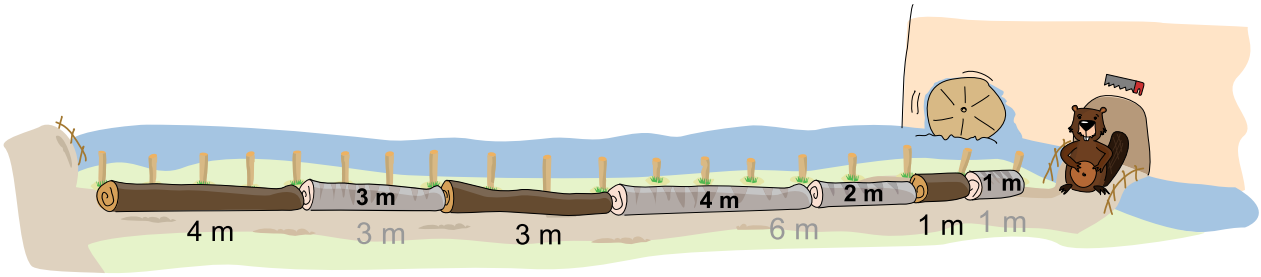
Timea veut maintenant couper quatre troncs longs de 1, 2, 3, et 4 mètres.

Dans quel ordre Timea doit-elle couper les troncs afin de tous pouvoir les placer dans les espaces sur le chemin ?



Solution

La bonne réponse est :



Si Timea coupe les troncs dans l'ordre (3 m, 4 m, 2 m, 1 m), elle peut tous les mettre dans les espaces sur le chemin. Pour le tronc de 3 m, l'espace de 3 m tout à gauche est le premier espace depuis la gauche assez grand pour l'y mettre. Timea y met donc le tronc de 3 m. Le tronc de 4 m va ensuite dans l'espace de 6 m à gauche, laissant un espace de 2 m. Cet espace de 2 m est le premier espace de libre pour le tronc de 2 m, et Timea met le dernier tronc dans l'espace de 1 m restant.

D'autres ordres possibles sont (3 m, 2 m, 4 m, 1 m) et (4 m, 3 m, 2 m, 1 m).

Aucun des autres ordres ne permet à Timea de poser tous les troncs : le tronc de 1 m doit toujours venir en dernier, car c'est le seul à pouvoir occuper le dernier espace. Le tronc de 2 m ne peut pas être coupé avant celui de 3 m, car il serait mis dans l'espace de 3 m et générerait un nouvel espace de 1 m. Seuls les trois ordres ci-dessus remplissent ces conditions.

C'est de l'informatique !

Cet exercice du Castor est un cas particulier du *problème de bin packing*. Dans ce problème, il s'agit de ranger des objets de tailles différentes dans un certain nombre de boîtes, boîtes pouvant elles aussi avoir des tailles différentes. Ici, les objets sont les troncs et les boîtes sont les espaces vides sur le chemin.

Les problèmes de *bin packing* se rencontrent dans des situations très différentes de la vie quotidienne. Quelques exemples : (a) des petits et grands meubles doivent être rangés dans un dépôt de meubles en économisant la place ; (b) une société de transport veut faire des économies et utiliser moins de camions en rangeant les paquets de manière optimale ; (c) le système d'exploitation d'un ordinateur doit enregistrer des données de différentes tailles sur le disque dur. Lorsque les données sont effacées, des espaces vides apparaissent sur le disque dur. Ces espaces doivent être remplis sans que de l'espace de stockage ne soit gaspillé, comme sur le chemin de cet exercice.

En informatique, le problème de *bin packing* est considéré comme l'un des problèmes les plus difficiles. Même les programmes informatiques ne peuvent trouver de solutions garanties optimales que pour les cas ne comptant que peu d'objets et de boîtes. Il existe par contre plusieurs méthodes et stratégies permettant de trouver de bonnes solutions aux problèmex de *bin packing*. Dans cet exercice, la stratégie est imposée par la règle de Timea : elle pose chaque tronc dans le premier espace assez grand depuis la gauche. On appelle cette stratégie *first fit*. On observe dans cet exercice que cette

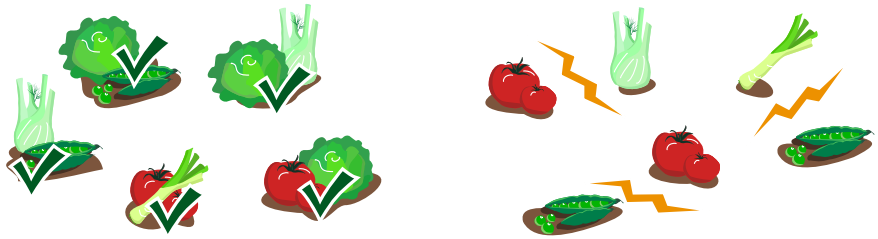
stratégie peut mener à de mauvais résultats : les troncs doivent être placés dans un certain ordre pour pouvoir remplir tous les espaces vides sur le chemin.

Mots clés et sites web

- Problème de *bin packing* : `https://fr.wikipedia.org/wiki/Problème_de_bin_packing`
- Gestion de la mémoire : `https://fr.wikipedia.org/wiki/Gestion_de_la_mémoire`
- Fragmentation : `https://fr.wikipedia.org/wiki/Fragmentation_(informatique)`

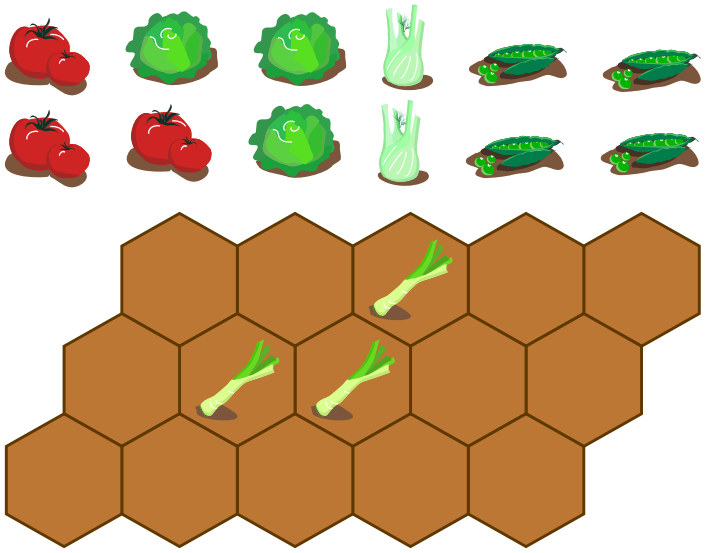
6. Jardin potager

Lisa prépare un jardin potager. Elle veut y cultiver cinq sortes de légumes différentes. Certaines sortes de légumes se supportent bien et sont compatibles ✓, d'autres sont incompatibles ↯ :



Lisa a divisé le jardin en domaines hexagonaux. Elle veut planter exactement une sorte de légumes dans chaque domaine.

Lisa a déjà planté des poireaux  dans trois domaines.

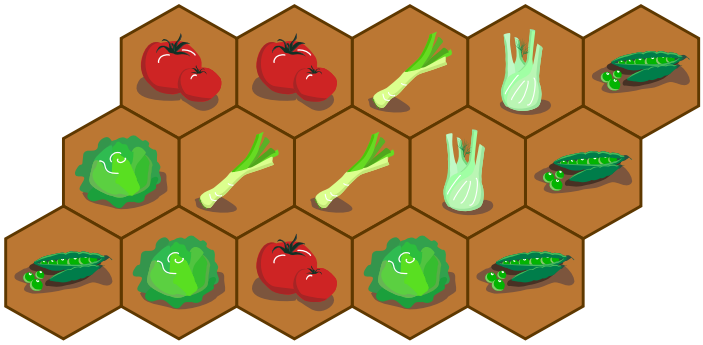


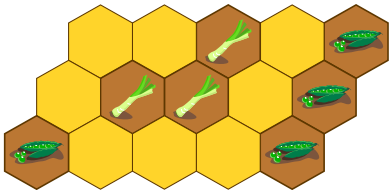
Lisa plante en suivant la règle suivante : les légumes incompatibles ne peuvent pas être plantés dans des domaines qui se touchent.

Plante une sorte de légumes dans chaque domaine encore libre en respectant la règle de Lisa.

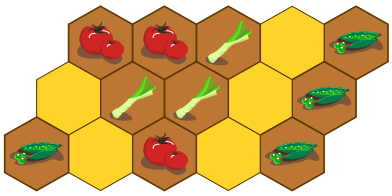
Solution

Voici la bonne réponse :

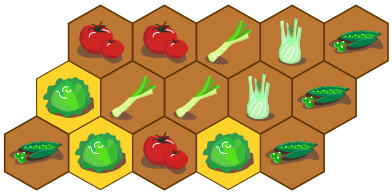




Comme les petits pois ne sont pas compatibles avec les poireaux, Lisa ne plante pas de petits pois dans les domaines jaunes. Il ne reste que les autres domaines pour le petits pois.



Comme les tomates ne sont pas compatibles avec les petits pois, Lisa ne plante pas de tomates dans les domaines jaunes. Elle peut planter des tomates dans les autres domaines, car les tomates et les poireaux sont compatibles.



Comme les fenouils ne sont pas compatibles avec les tomates, Lisa ne plante pas de fenouil dans les domaines jaunes. Elle peut planter du fenouil dans les deux domaines entre les poireaux et les petits poids. Lisa peut planter de la salade dans les domaines jaunes, car la salade est compatible avec tous les autres légumes.

C'est de l'informatique !

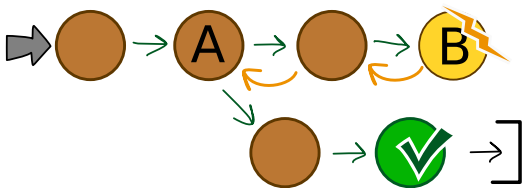
Pour planter des légumes afin d'avoir une récolte aussi grande que possible, il faut respecter beaucoup de *conditions* : chaque sorte a des besoins de place, de nutriments et de lumière différents, par exemple. Dans cet exercice du Castor, nous ne considérons qu'une sorte de condition : la compatiblité des différentes sortes de légumes entre elles.

Pour déterminer quoi planter où dans le jardin de Lisa tout en respectant les conditions de compatiblité, on pourrait procéder de la manière suivante : on essaie toutes les combinaisons de légumes de manière systématique. Une fois que le jardin est rempli, on vérifie si les conditions sont remplies et si la combinaison est une solution au problème de Lisa. En informatique, on appelle un telle manière d'essayer toutes les combinaisons possibles une *recherche exhaustive*. Cette méthode peut prendre beaucoup de temps si elle est appliquée à des problèmes ayant beaucoup de combinaisons possibles et peu de solutions.

C'est souvent mieux de procéder étape par étape et de prendre en compte toutes les conditions à chaque étape. C'est ainsi que nous avons trouvé la solution au problème de Lisa, et aucune « fausse » combinaison ou arrangement du jardin n'était possible.

Heureusement, c'était possible de trouver la solution directement : il y avait toujours des domaines dans lesquels nous pouvions planter certains des légumes restants. Ce n'est pas toujours le cas.

Lorsque l'on essaie d'assembler la réponse étape par étape, il peut y avoir plusieurs possibilités de remplir toutes les conditions à une certaine étape A :



Suivant le choix fait en A, il peut ne plus y avoir de possibilités à une étape suivante B. On revient alors en arrière sur les dernières étapes jusu'à arriver à nouveau à l'étape A offrant plusieurs possibilités. On choisit alors une autre possibilité et essaie de trouver un solution depuis là.

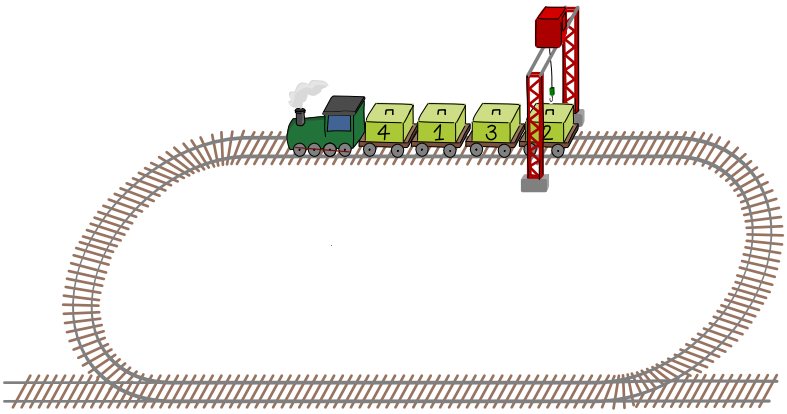
Ce retour en arrière est appelé *retour sur trace* en informatique (*backtracking* en anglais).

Mots clés et sites web

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

7. Train de marchandises

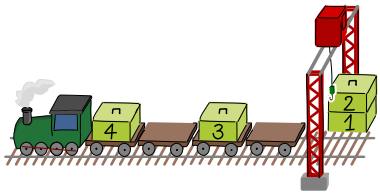
Un train tire des wagons chargés de caisses numérotées. La grue est à une position fixe et décharge les caisses. Pour décharger une caisse, la caisse doit être positionnée directement sous la grue.



La grue doit décharger les caisses dans l'ordre croissant de leurs numéros en commençant par la caisse 1. Le train ne peut rouler qu'en avant. Il doit faire un tour complet pour pouvoir décharger d'autres caisses après avoir dépassé la grue.

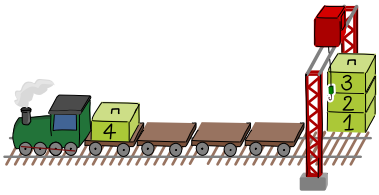
Voilà comment il décharge les caisses 1, 2, 3 et 4 :

Tour 1 :



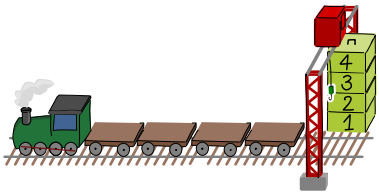
Il saute la caisse 4, décharge la caisse 1, saute la caisse 3 et décharge la caisse 2.

Tour 2 :



Il saute la caisse 4 et décharge la caisse 3.

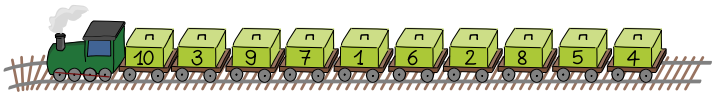
Tour 3 :



Il décharge la caisse 4.

Le train ci-dessus doit donc faire trois tours pour décharger toutes les caisses dans le bon ordre.

Combien de tours faut-il pour décharger le train suivant ?



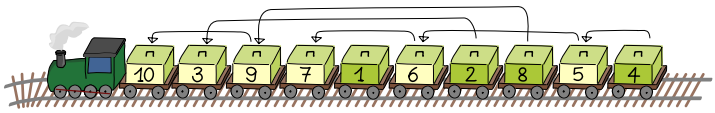
A) 1 tour
B) 2 tours
C) 3 tours
D) 4 tours
E) 5 tours
F) 6 tours
G) 7 tours
H) 8 tours
I) 9 tours
J) 10 tours

Solution

La bonne réponse est sept tours.

L'ordre imposé pour décharger est 1, 2, 3, 4, 5, 6, 7, 8, 9 et 10. Au premier tour, les caisse 1 et 2 sont déchargées ensemble. Au deuxième tour, les caisses 3 et 4 sont déchargées ensemble, puis la caisse 5, puis la 6, puis les caisses 7 et 8 ensemble, puis la 9 et finalement la 10. Cela fait sept tours en tout.

Alternativement, on peut utiliser le fait qu'un tour supplémentaire est nécessaire chaque fois que la caisse suivante se trouve à la gauche de la caisse actuelle.



Par exemple, comme la caisse 3 est à gauche de la caisse 2, elle sera sautée pour décharger la caisse 2 et nécessitera un tour supplémentaire pour la ramener à la hauteur de la grue. Ici, c'est le cas pour les paires de caisses (2,3), (4,5), (5,6), (6,7), (8,9) et (9, 10) ; il faut donc six tours en plus du premier, ce qui fait sept tours en tout.

C'est de l'informatique !

Lorsque, pour n'importe quel numéro de la suite 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, le numéro suivant se trouve plus à gauche dans le train, on appelle cela une *inversion*. Chaque inversion nécessite un tour supplémentaire. On obtient la réponse de l'exercice en comptant le nombre d'inversions.

Il y a beaucoup d'applications liées au nombre d'inversions présentes dans une suite. Pour certains algorithmes de tri, comme le *tri à bulles*, le nombre d'inversions nous renseigne sur le nombre de permutations nécessaire pour obtenir la suite désirée. Si deux clients classent le même ensemble d'articles par préférence, le nombre d'inversions entre leurs classements nous informe sur leurs préférences communes. C'est utilisé par les magasins en ligne pour identifier des clients « similaires » et recommander des produits.

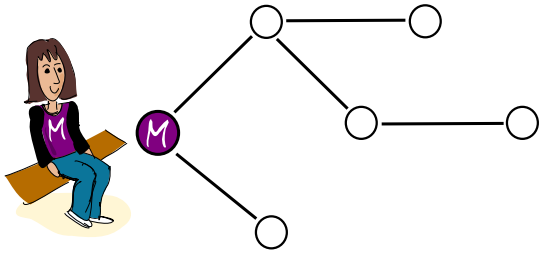
Mots clés et sites web

- Algorithme de tri : `https://fr.wikipedia.org/wiki/Algorithme_de_tri`
- Tri à bulles : `https://fr.wikipedia.org/wiki/Tri_à_bulles`

8. Le village de Martina

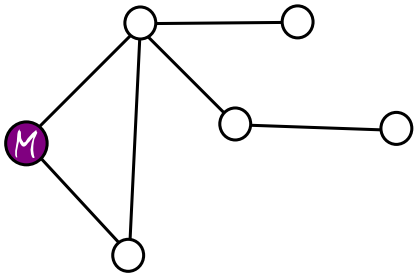
Il y a six maisons dans le village de Martina. Il y a aussi des chemins pour aller d'une maison à la suivante. Martina met le même temps à parcourir chacun de ces chemins.

Martina a dessiné une carte du village spéciale. Elle y a dessiné les chemins qui lui permettent d'aller le plus vite possible jusqu'aux autres maisons.

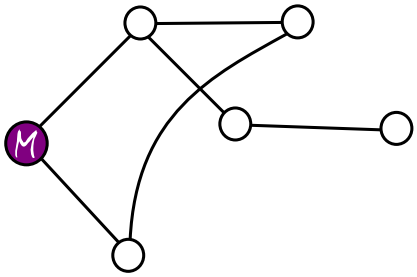


Il existe bien sûr aussi une vraie carte du village avec tous les chemins.

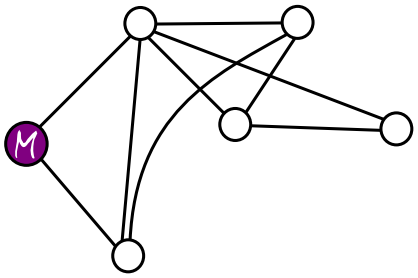
Lequel de ces dessins ne peut-il pas *être la vraie carte ?*



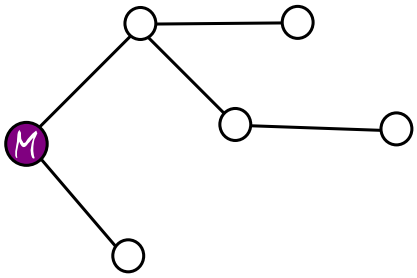
A)



B)



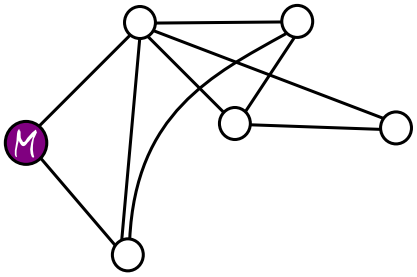
C)



D)

Solution

La bonne réponse est C :



La carte spéciale de Martina montre que le chemin le plus court jusqu'à la maison tout à droite passe par chemins. Si C était la vraie carte du village, Martina pourrait aller plus vite jusqu'à cette maison en ne passant que par deux chemins. C ne peut donc pas être la vraie carte du village.

Les cartes A, B et D ne montrent de chemin plus rapide jusqu'à aucune des maisons que ceux de la carte spéciale de Martina. Ces cartes peuvent donc être les vraies cartes du village.

C'est de l'informatique !

Martina est informaticienne. Elle a dessiné sa carte sous forme de *graphe*. Un graphe est constitué de *nœuds* (ici, les maisons) qui peuvent être reliés par des *arêtes* (ici, les chemins). Dans de nombreux domaines informatiques, les graphes peuvent modéliser la réalité – comme dans cet exercice du Castor.

Martina sait qu'il existe beaucoup d'algorithmes pour les graphes, qui permettent de répondre à des questions telles que « quel est le chemin le plus court jusqu'à une autre maison ? », comme le parcours en largeur. Peut-être qu'elle a élaboré sa carte spéciale à l'aide d'un parcours en largeur d'un graphe plus grand, la vraie carte du village.

En théorie des graphes, qui traite des graphes et algorithmes associés, la carte de Martina correspond à un sous-graphe de la carte complète du village. La carte de Martina a deux particularités :

- Tous les nœuds sont reliés directement (par une arête) ou idirectement (par plusieurs arêtes) les uns aux autres ;
- Il n'y a toujours qu'un seul chemin reliant les deux nœuds de n'importe quelle paire.

En informatique, un graphe avec ces particularité est appelé un *arbre*. La maison de Martina correspond à la *racine* de l'arbre. En partant de la racine, Martina peut atteindre tous les autres nœuds (les autres maisons du village) d'une seule manière. Le graphe de Martina est donc un arbre ; de plus, il contient tous les nœuds du graphe complet (la vraie carte du village), mais pas forcément toutes ses arêtes. Un sous-graphe ayant ces propriétés est appelé un *arbre couvrant* du graphe complet.

En informatique, les algorithmes traitant les graphes ont beaucoup d'applications, surtout celles liées aux réseaux (réseaux de transport, réseaux de communication…), par exemple le calcul d'itinéraires par les systèmes de navigations. Les arbres couvrants peuvent être utilisés pour la construction de réseaux peu coûteux et être utile pour résoudre des problèmes particulièrement difficiles.

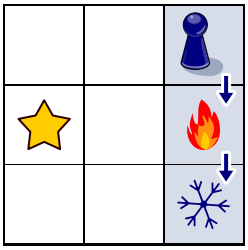
Mots clés et sites web

- Théorie des graphes : `https://fr.wikipedia.org/wiki/Théorie_des_graphes`
- Arbre : `https://fr.wikipedia.org/wiki/Arbre_(théorie_des_graphes)`
- Parcours en largeur : `https://fr.wikipedia.org/wiki/Algorithme_de_parcours_en_largeur`
- Arbre couvrant : `https://fr.wikipedia.org/wiki/Arbre_couvrant`

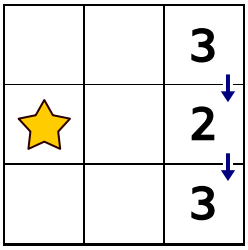
9. Chaud ou froid

Nina et Daniel jouent à la chasse au trésor. Dans sa tête, Nina choisit une case sur une planche de jeu à cases carrées. C'est là que le trésor est caché.

Daniel choisit une case de départ. En partant de là, son pion avance pas à pas d'une case vers la gauche, la droite, le haut ou le bas.



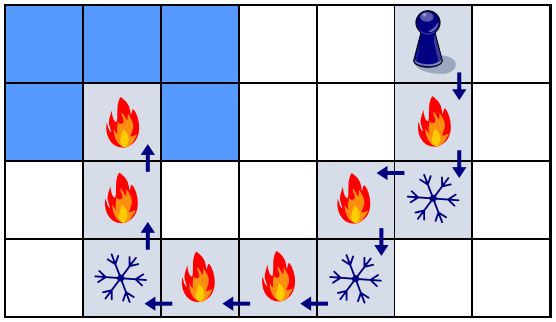
Pour commencer, Nina et Daniel prennent un petit plateau. Nina cache le trésor sur la case avec l'étoile. Daniel commence en haut à droite et fait deux pas en suivant les flèches. Après chaque pas, Nina lui dit s'il est plus près ou plus loin du trésor qu'avant.



Cette image montre la distance entre Daniel et le trésor pour ces trois cases. Cette distance est le nombre minimal de pas qui pourraient amener le pion de Daniel au trésor.

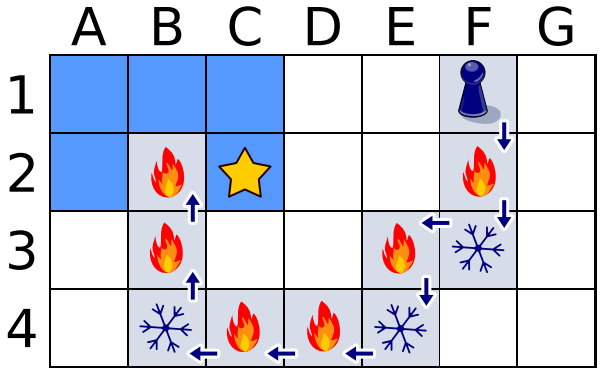
Ils prennent maintenant une plus grande planche de jeu. Nina cache le trésor sur une des cases bleues. L'image montre les pas de Daniel et ce que Nina dit après chaque pas.

Où se cache le trésor ?



Solution

Voici la bonne réponse :



Nous suivons le chemin de Daniel et les indications de Nina. Daniel commence sur la ligne 1 de la planche de jeu. Après le premier pas, il est sur la ligne 2 et est plus près du trésor que sur la ligne 1. Après le pas suivant, il est sur la ligne 3 et à nouveau plus loin du trésor que sur la ligne 2. Comme il est resté sur la même colonne, le trésor doit se trouver sur une case de la ligne 2. En effet, quelle que soit la colonne sur laquelle le trésor est caché, on a le chemin le plus court depuis une autre colonne en partant de la même ligne que le trésor.

Mais sur quelle colonne le trésor est-il caché ? En continuant son chemin jusqu'à la ligne 4, Daniel arrive plus près après quelques pas vers la gauche ; il est plus près du trésor sur la colonne 3 que sur la colonne 4. Mais après le dernier pas sur la ligne 4, Daniel est de nouveau plus loin du trésor sur la colonne 2 que sur la colonne 3. Le trésor doit donc être sur une case de la colonne 3, car ce qui vaut pour les lignes vaut aussi pour les colonnes : le chemin le plus court part de la même colonne que celle où se trouve le trésor.

C'est de l'informatique !

Daniel se déplace (avec son pion) sur la planche de jeu. Nina mesure la distance entre chaque case sur laquelle il se trouve et le trésor et utilise cela pour son feeback. Habituellement, on utilise la longueur de la ligne droite reliant deux points comme mesure de la distance entre eux (*distance euclidienne*). Cependant, les deux cases ne sont pas des points. C'est pour cela que Nina utilise le nombre de pas minimal que Daniel devrait faire pour atteindre le trésor comme distance. Cette *mesure* peut être appliquée aux grilles et est connue sous le nom de *distance de Manhattan* en informatique, d'après la forme de grille du plan de Manhattan, à New York.

Les informaticiennes et informaticiens choisissent le type de mesure de la distance entre deux objets en fonction de la question à laquelle ils veulent répondre. Par exemple, si l'on veut mesurer la distance entre deux mots de même taille d'un langage naturel, on peut compter le nombre de positions auxquelles les mots diffèrent ; il s'agit alors de la *distance de Hamming*. Si les mots sont de tailles différentes, on peut utiliser la *distance de Levenshtein*. En informatique, les distances jouent souvent un rôle dans la recherche de solutions optimales : qu'importe si la solution du problème doit être

la plus rapide, la plus courte ou la moins chère, il suffit souvent de changer la mesure de distance (durée, longueur ou coût) sans rien changer à l'algorithme.

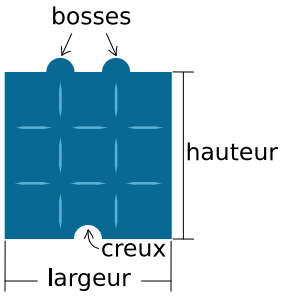
Mots clés et sites web

- Distance de Manhattan : `https://fr.wikipedia.org/wiki/Distance_de_Manhattan`
- Distance de Hamming : `https://fr.wikipedia.org/wiki/Distance_de_Hamming`
- Distance de Levenshtein : `https://fr.wikipedia.org/wiki/Distance_de_Levenshtein`

10. Briques castor

Les briques castor ont quatre propriétés qui les différencient :

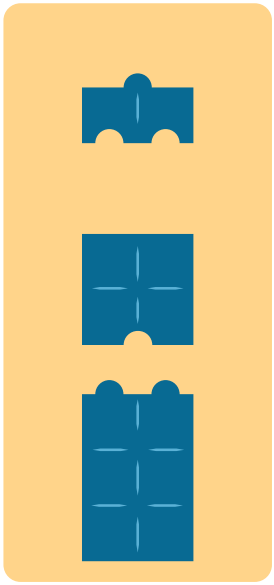
1. Largeur : étroite, moyenne, large
2. Hauteur : petite, moyenne, grande
3. Nombre de bosses en haut : zéro, une, deux
4. Nombre de creux en bas : zéro, un, deux



Otto répartit les briques en groupes de trois. Il le fait de manière à ce que les trois briques de chaque groupe aient, pour chacune des quatre propriétés…

- … trois fois la même valeur…
- … ou trois valeurs différentes.

Voici l'un des groupes d'Otto, à droite.



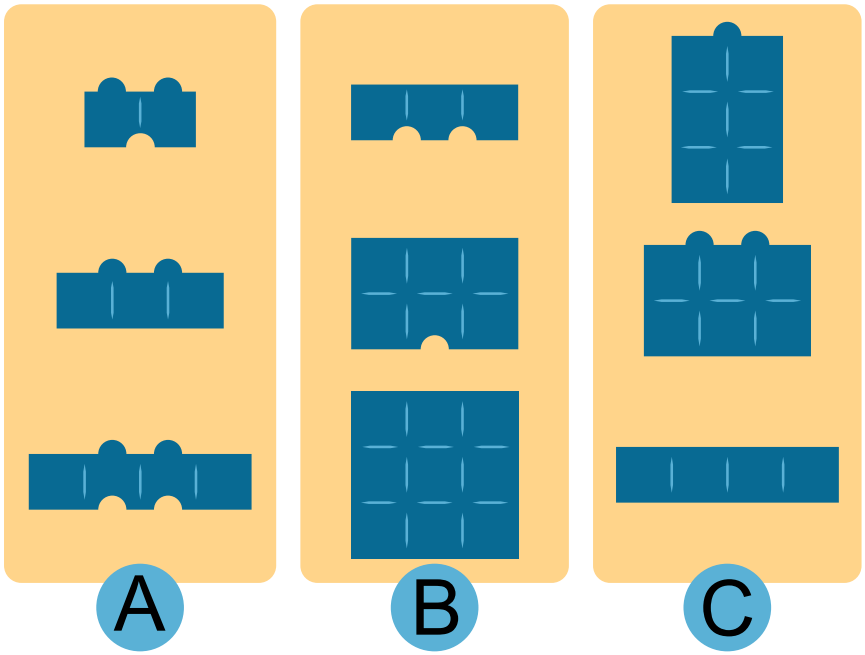
Ces trois briques ont :

- toutes la même largeur,
- trois hauteurs différentes,
- un nombre de bosses différent,
- un nombre de creux différent.

Répartis ces briques en groupes de trois comme le ferait Otto.

Solution

Voici la bonne répartition :



Les briques sont réparties en groupes comme Otto le ferait. La table suivante montre quelles propriétés ont la même valeur ou des valeurs différentes pour chaque groupe.

Propriété	**Groupe A**	**Groupe B**	**Groupe C**
Largeur	différentes	égales	différentes
Hauteur	égales	différentes	différentes
Bosses	égales	égales	différentes
Creux	différentes	différentes	égales

Mais est-ce la seule possibilité de répartir les briques comme Otto le ferait ?

On peut y réfléchir de la manière suivante : si une propriété doit avoir des valeurs différentes dans chaque groupe, chaque valeur doit être présente sur le même nombre de briques qu'il y a de groupes. Si ce n'est pas le cas, il doit y avoir au moins un groupe dans lequel toutes les briques ont la même valeur pour cette propriété.

En observant les briques, on voit que les valeurs de largeur étroite et large ne sont présentes que sur deux briques chacune. Il doit donc y avoir un groupe pour lequel la valeur de la largeur est moyenne pour toutes les briques.

Aucune des cinq briques de largeur moyenne n'a qu'une seule bosse ; il ne peut donc pas y avoir de groupe dans lequel toutes les briques ont un nombre de bosses différent. Par contre, il y a trois briques avec zéro bosse – et elles ont toutes une hauteur différente et un nombre de creux différent. Le groupe B est donc le seul groupe de brique de largeur moyenne possible.

Chacun des deux autres groupes doit contenir des briques ayant toutes une largeur différente.

Parmi les six briques restantes, la valeur de la hauteur n'est grande ou moyenne qu'une seule fois. Il doit donc y avoir un groupe dans lequel toute les briques sont petites. Le groupe A est le seul groupe correspondant aux règles d'Otto avec des briques de petite taille. Les trois briques restantes forment le groupe C, qui correspond également aux règles d'Otto.

C'est de l'informatique !

Dans cet exercice du Castor, les briques castor sont décrites à l'aide de quatre *propriétés* (ou *attributs*). Pour pouvoir répartir les briques dans des groupes de trois comme Otto, il faut connaître la valeur des propriétés de chaque brique.

Pour cela, il suffit à un être humain de jeter un coup d'œil à chaque brique. Un programme informatique devant faire des groupes de trois ne peut généralement pas voir et a besoin d'une description enregistrée dans une *structure de données*.

On peut par exemple décrire les briques dans les lignes d'une *base de données*. Les colonnes de la base de données correspondent aux propriétés, et dans chaque ligne (aussi appelée *enregistrement*) se trouvent les valeurs d'une brique dans les colonnes correspondantes :

Brique	**Largeur**	**Hauteur**	**Bosses**	**Creux**
1	étroite	grande	1	0
2	moyenne	moyenne	2	0
...	...	...	...	...

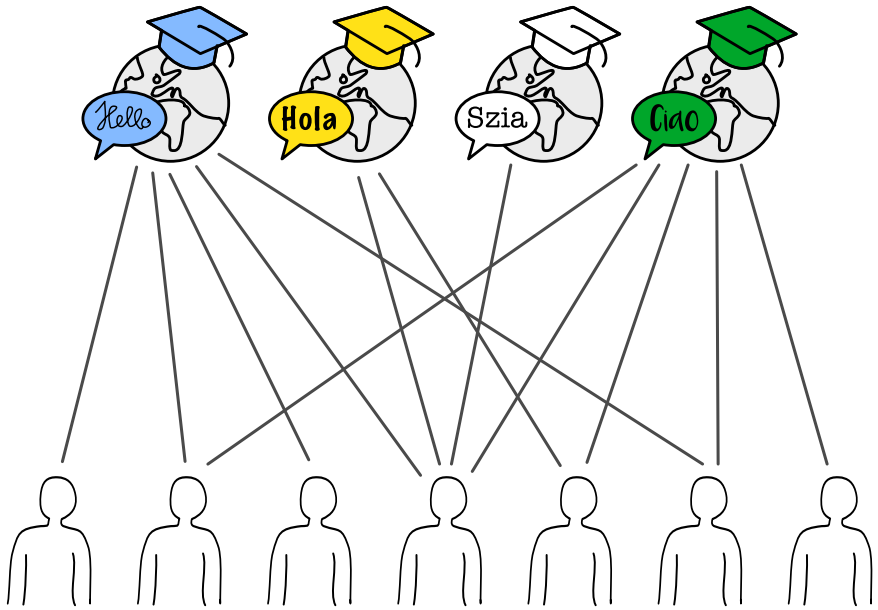
L'élaboration de bases de données est une tâche habituelle pour les informaticiennes et informaticiens. C'est un travail minutieux et il faut bien réfléchir à quels attributs des objets sont importants pour le traitement des données par un programme informatique. Ce n'est pas facile de faire des changements une fois la base de données remplie, surtout quand les données de beaucoup d'objets sont déjà enregistrées.

Mots clés et sites web

- Structure de données : `https://fr.wikipedia.org/wiki/Structure_de_données`
- Base de données : `https://fr.wikipedia.org/wiki/Base_de_données`
- Enregistrement : `https://fr.wikipedia.org/wiki/Enregistrement_(base_de_données)`

11. Répartition des tâches

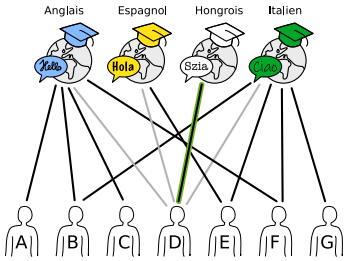
Une école de langue organise quatre cours d'été. Sur l'image ci-dessous, les lignes montrent quel enseignant de l'école est capable de donner quel cours.



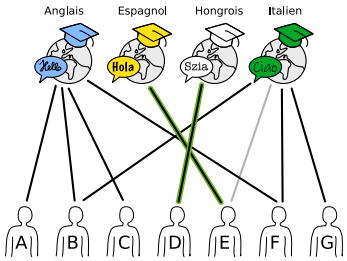
Chaque enseignant ne peut donner qu'un seul cours. Il y a quand même plusieurs possibilités d'assigner un enseignant capable à chaque cours.

Assigne un enseignant à chaque cours. Pour cela, surligne la ligne reliant l'enseignant au cours.

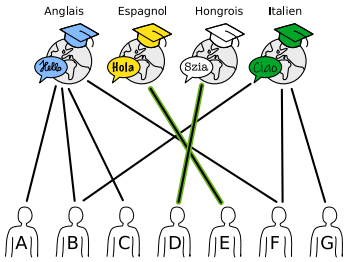
Solution



D est la seule personne capable d'enseigner le hongrois. Elle doit donc être assignée à ce cours et ne peut pas en donner d'autre.



E est maintenant la seule personne capable d'enseigner l'esapgnol. Elle doit donc être assignée à ce cours et ne peut pas en donner d'autre.



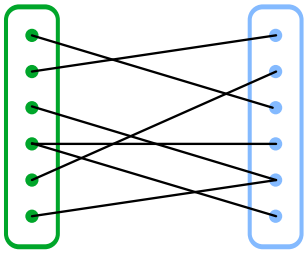
Pour les deux cours restants, l'anglais et l'italien, on a le choix. B et F ne peuvent être assignés qu'à un seul cours chacun, même s'ils sont capables d'enseigner les deux.

Il y a ainsi dix possibilités en tout d'assigner un enseignant capable à chaque cours :

anglais	**italien**	**hongrois**	**espagnol**
A	B	D	E
A	F	D	E
A	G	D	E
B	F	D	E
B	G	D	E
C	B	D	E
C	F	D	E
C	G	D	E
F	B	D	E
F	G	D	E

C'est de l'informatique !

Un *graphe* est constitué de *nœuds* (points) qui sont reliés par des *arêtes* (lignes). Les *graphes bipartis* sont un type de graphe spécial : les nœuds peuvent être séparés en deux sous-ensembles de manière à ce qu'il n'y ait d'arêtes qu'entre les deux sous-ensembles.



La situation de cet exercice du Castor peut être représentée par un graphe biparti : les cours forment l'un des sous-ensembles et les enseignants l'autre sous-ensemble. Les graphes bipartis sont bien adapté à la modélisation et la résolution de problèmes d'affectation. On rencontre fréquemment des problèmes d'affectation au quotidien, par exemple lors de l'élaboration d'horaires ou de la répartition du travail entre des employés ou des machines. Pour le petits problèmes, il est possible de trouver simplement une solution optimale ; cela devient cependant vite très complexe pour les plus grands problèmes. C'est pour cela que différents algorithmes permettant de trouver un maximum de paires rapidement ont été développés en informatique.

Une autre exemple de problème pouvant être représenté par un graphe biparti est le problème des mariages. Un ensemble d'hommes désirant se marier fait face à un ensemble de femmes désirant également se marier. Le but du procédé est de marier chaque homme à une femme (et chaque femme à un homme) en respectant les souhaits de partenaire de chacun. Le mathématicien Philipp Hall a formulé les conditions dans lesquelles une telle affectation est possible dans le lemme des mariages en 1935.

Dans notre cas, il ne s'agit pas de ce type d'affectation complète, mais d'affecter à chaque nœud d'un sous-ensemble (les cours) un nœud d'un autre sous-ensemble (les enseignants).

Mots clés et sites web

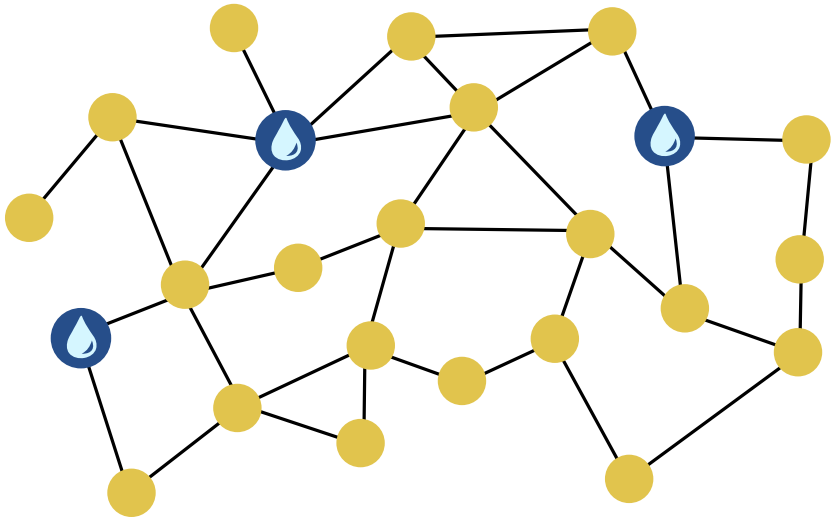
- Graphe biparti : `https://fr.wikipedia.org/wiki/Graphe_biparti`
- Problème d'affectation : `https://fr.wikipedia.org/wiki/Problème_d'affectation`
- Programme pour résoudre l'exercice :
  `https://www.coding4you.at/dachu_2023/ir02/index.html`
- Lemme des mariages : `https://images.math.cnrs.fr/Le-lemme-des-Mariages.html`

12. Fontaine

L'été est chaud dans la ville. La maire fait installer des fontaines d'eau potable.

Les fontaines doivent être installées de manière à ce qu'il ne faille jamais parcourir plus de deux tronçons de rue pour atteindre une fontaine depuis n'importe quel coin de rue. La maire sera alors satisfaite.

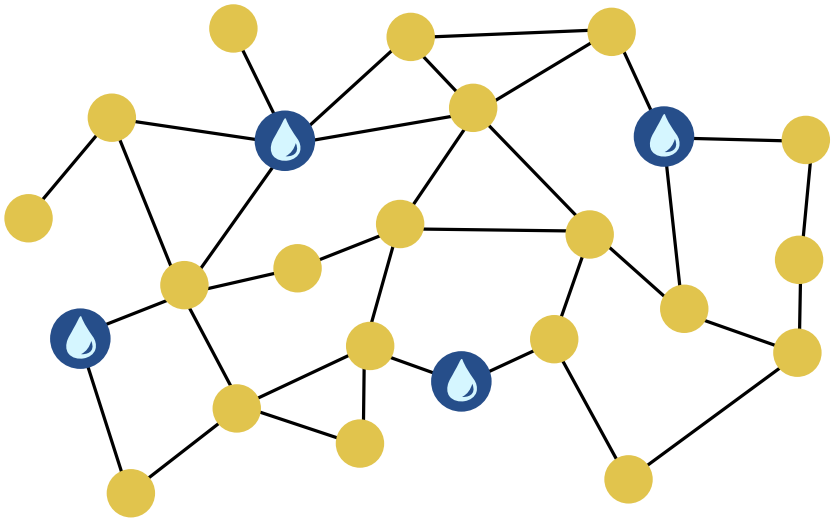
Voici un plan de la ville. Les lignes sont les tronçons de rue et les points les coins de rue. Il y a déjà des fontaines à trois coins de rue.



Installe une fontaine supplémentaire pour satisfaire la maire.

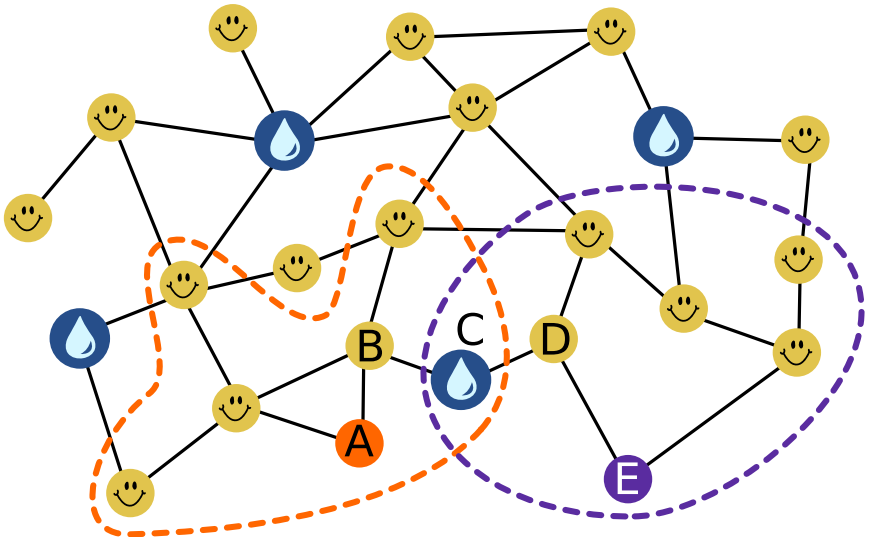
Solution

Voici la bonne réponse :



Si une fontaine supplémentaire est intallée en bas au centre, il faut parcourir au maxium deux tronçons de rue pour atteindre une fontaine. La maire est alors satisfaite.

Comment pouvons-nous déterminer à quel coin de rue installer une fontaine supplémentaire ? Sur le plan, nous indiquons d'un tous les coins de rue se trouvant déjà à deux tronçons de rue d'une fontaine au maximum. La maire peut déjà être satisfaite de ces coins de rue.



Pour les cinq coins de rue A, B, C, D et E restants, nous ajoutons une fontaine au coin de rue C. Comme ça, ces coins-là sont aussi loin de deux tronçons au maximum d'une fontaine.

Le coin C est le seul endroit où installer une nouvelle fontaine permettant de satisfaire la maire : Si l'on considère les coins de rue se trouvant à deux tronçons des coins A et E (entourés d'une ligne sur l'image), seul le coin C remplit cette condition pour les coins A *et* E.

C'est de l'informatique !

Le plan de la ville peut être représenté par un *graphe*. C'est un outil important en informatique qui permet de modéliser les relations entre des objets et de répondre à des questions sur ces relations. Ici, on peut représenter les coins de rue comme des objets et donc des *nœuds* du graphe. Les relations entre deux objets sont modélisées par des *arêtes* qui relient deux nœuds. Ici, une arête entre deux coins de rue veut dire qu'ils sont reliés par un tronçon de rue. On peut appeler cette relation « voisinage ». Les arêtes peuvent aussi représenter d'autres relations, comme l'amitié.

Dans cet exercice du Castor, il faut trouver un sous-ensemble de nœuds (pour installer une fontaine) de manière à ce que chaque nœud à l'extérieur de ce sous-ensemble soit relié à un « nœud fontaine » par un chemin de deux arêtes de long au maximum. En langage technique informatique, on parlerait de la recherche d'un ensemble 2-dominant (*distance−2 dominating set* en anglais). En général (pour toutes les longueurs de chemin $d \geq 1$), cette recherche d'un sous-ensemble de taille minimale fait partie des problèmes les plus difficiles rencontrés en informatique.

Les « ensembles d-dominants » jouent un rôle croissant actuellement, en particulier dans le domaine du *social computing* : pour traiter des données venant de réseaux sociaux de manière automatique (par exemple pour détecter la diffusion de *fake news*), les relations entre les utilisateurs (fan, follower, ami) sont modélisées sous forme de graphes. Ces graphes peuvent être si grands que seule une sélection représentative (aussi petite que possible) peut être prise en considération – par exemple, un set 3-dominant. Comme la sélection la plus petite possible ne peut pas être calculée efficacement, on développe des méthodes informatiques qui permettent de rapidement déterminer de petites sélections (mais pas forcément *la* plus petite).

Mots clés et sites web

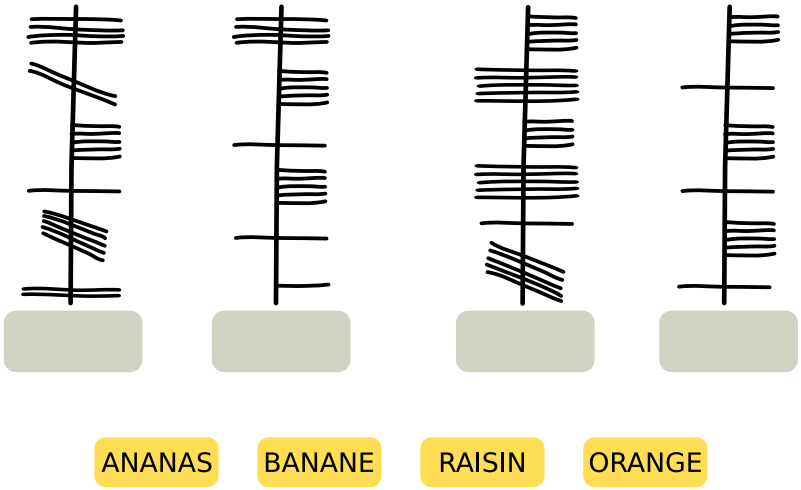
- Ensemble dominant : `https://fr.wikipedia.org/wiki/Ensemble_dominant`
- Théorie des graphes : `https://fr.wikipedia.org/wiki/Théorie_des_graphes`
- Analyse des réseaux sociaux : `https://fr.wikipedia.org/wiki/Analyse_des_réseaux_sociaux`

13. Ogham

Sue connaît le vieil alphabet irlandais utilisé en écriture oghamique. Chaque lettre est composée d'un ou plusieurs traits qui sont arrangés le long d'une longue ligne. Deux lettres qui se suivent sont séparées par un espace le long de la ligne.

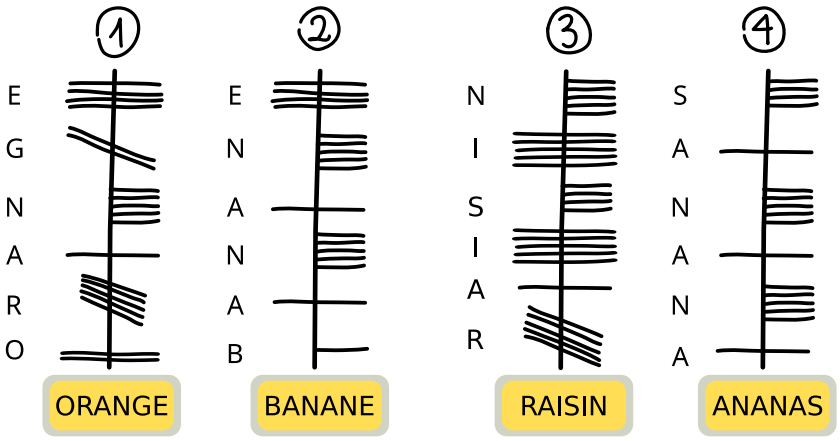
Sue utilise l'écriture oghamique comme code secret. Elle écrit ainsi quatre mots – ses fruits préférés : ANANAS, BANANE, RAISIN et ORANGE.

Quel mot correspond à quel code en Ogham ?



Solution

Voici la bonne réponse :



Il y a plusieurs possibilités de trouver la bonne assignation. Il faut dans tous les cas déterminer dans quel sens les lettres sont écrites le long de la ligne. Pour cela, le mot ANANAS est spécialement utile : il contient trois fois la lettre A, séparée par d'autres lettres.

Il n'y a que dans le code en Ogham 4 que la même lettre apparaît trois fois avec d'autres lettres entre deux. Le code 4 est donc le seul qui peut correspondre au mot ANANAS. On peut en déduire que les mots sont écrits de bas en haut en Ogham et que la lettre A s'écrit avec un trait horizontal traversant le ligne verticale.

La lettre A en Ogham n'est présente deux fois que dans le code 2. De plus, on connaît le symbole Ogham du N grâce au code d'ANANAS (cing traits verticaux à droite de la ligne), et l'ordre des autres lettres indique que seul le mot BANANE correspond à ce code. ORANGE ne va qu'avec le code 1, entre autre parce qu'on n'y trouve la lettre A qu'une seule fois et en troisième position. Il ne reste que le code 3 pour le mot RAISIN, et on y retrouve en effet les lettres Ogham R, S et N connues des autres mots aux bonnes positions.

C'est de l'informatique !

Dans cet exercice du Castor, il faut déchiffrer un texte inconnu. Ici, ce n'est pas très difficile car le *texte clair* est connu. De plus, le texte inconnu est divisé en lettres et en mots comme le texte connu. Lorsque l'on déchiffre un texte secret ou dans un alphabet inconnu sans le connaître en texte clair, c'est souvent utile de réfléchir à la fréquence des mots et des lettres, et d'utiliser cela comme base pour trouver ces mots et lettres dans le texte. C'est de cette manière que plusieurs écritures et alphabets antiques ont été déchiffrés. Cela devient plus compliqué lorsque les symboles du texte inconnu ne sont pas faciles à assigner aux lettres et mots du texte connu comme il le sont en Ogham. Dans ce cas, il est souvent nécessaire de comparer le texte à des textes ou écritures connues, comme dans cet exercice. Les hiéroglyphes égyptiens, par exemple, n'ont pas pu être déchiffrés pendant des siècles, jusqu'a ce qu'une pierre avec des hiéroglyphes et deux textes connus soit trouvée par hasard, la pierre de Rosette. Sur la pierre se trouvait trois fois le même texte écrit dans des langues

différentes, mais contenant les mêmes noms. Ceci permit de déchiffrer des éléments essentiels des hiéroglyphes. Ce n'est cependant pas le cas de tous les alphabets : environ 650 symboles de la culture Maya ne sont toujours pas entièrement déchiffrés, ainsi que les écritures linéaires A et B de la région méditéranéenne.

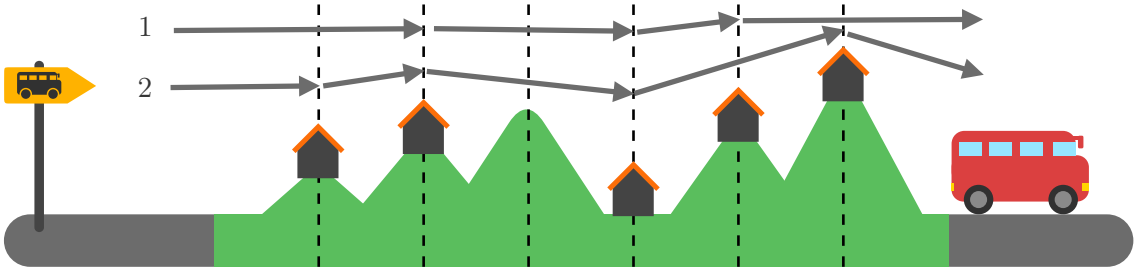
En informatique aussi, il faut déchiffrer des textes et des symboles – après qu'ils ont été encryptés pour le transfert de données sécurisé. Pour cela, des méthodes très différentes de celles utilisées pour coder des mots dans d'autres écritures sont appliquées. De tels chiffres simples sont trop faciles à déchiffrer, surtout avec des ordinateurs, en général à l'aide des analyses de fréquence des mots et lettres mentionnées plus haut.

Mots clés et sites web

- Cryptographie : `https://fr.wikipedia.org/wiki/Cryptographie`
- Cryptoanalyse : `https://fr.wikipedia.org/wiki/Cryptanalyse`
- Ogham : `https://fr.wikipedia.org/wiki/Ogham`

14. Randonnée

Pendant ses vacances, Mia aime faire des randonnées où elle dort chaque nuit à un endroit différent. Mia a une carte de la région de ses prochaines vacances. La carte montre son point de départ, son but et tous les endroits où elle peut passer la nuit.



Mia a divisé la région en sections à l'aide de lignes traitillées. Elle ne peut parcourir qu'une ou deux régions par jour en marchant. Elle a déjà noté deux des randonnées qu'elle peut faire sur la carte :

- La randonnée 1 dure trois nuits,
- La randonnée 2 dure quatre nuits.

Mia peut encore faire d'autres randonnées.

Combien de randonnées différentes Mia peut-elle faire ? Compte aussi les randonnées 1 et 2.

A) 2 randonnées

B) 3 randonnées

C) 4 randonnées

D) 5 randonnées

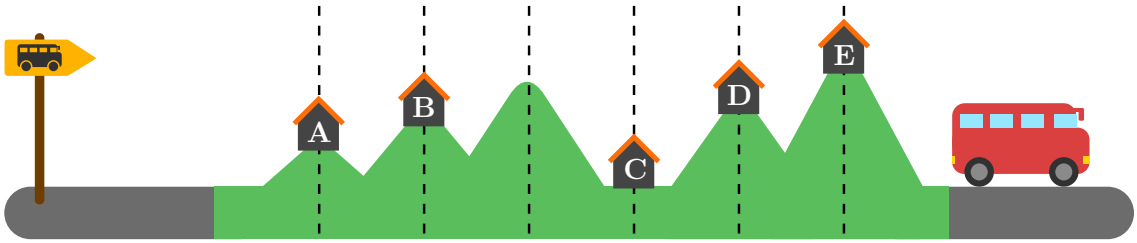
E) 6 randonnées

F) 7 randonnées

G) 8 randonnées

Solution

La bonne réponse est E) 6 randonnées.



D'abord, nous constatons que Mia doit passer la nuit à **B** et **C**, car la distance entre ces deux endroits (2) est la distance maximale qu'elle peut parcourir en un jour. Mia n'a donc qu'une possibilité de faire le chemin entre **B** et **C**.

Nous pouvons maintenant calculer le nombre de possibilités pour les autres parties de sa randonnée : Mia peut faire le chemin du départ à **B** en une fois ou passer la nuit à **A**, ce sont deux possibilités (comme pour les randoonnées 1 et 2). Mia doit parcourir trois sections entre **C** et le but et elle peut passer la nuit à chacun des deux endroits **D** et **E**. Elle peut donc diviser le chemin en toutes les combinaisons possibles d'une et deux sections :

- **C** → **D** → **E** → ;
- **C** → **E** → ;
- **C** → **D** → .

Le nombre total de randonnées que Mia peut faire est donc $2 \times 1 \times 3 = 6$.

C'est de l'informatique !

Parfois, le nombre total de possibilités de compléter une tâche est très grand. Il y a par exemple environ 14 millions de possibilités de tirer 6 nombres différents parmi les nombres entre 1 et 49. Et il y a environ un demi-milliard de possibilités d'écrire les nombres de 1 à 12 dans des ordres différents. Même un ordinateur a besoin d'un peu de temps pour le faire.

Dans cet exercice du Castor, heureusement qu'il n'y a pas de possibilité de passer la nuit après la troisième section et que l'on peut ainsi diviser l'exercice en trois parties ! Le problème est ainsi partagé en trois plus petits problèmes. En informatique, des méthodes divisant un problème en sous-problèmes sont souvent utilisées lors de la conception d'algorithmes. Ce principe est aussi connu sous le nom de *diviser pour régner*.

Plusieurs algorithmes de tri importants fonctionnent d'après ce principe. La *programmation dynamique*, une méthode de résolution algorithmique de problèmes d'optimisation (décrite par Richard Bellman en 1957), se base aussi sur ce principe : si l'on voit que la solution optimale d'un problème est composée des solutions optimales de sous-problèmes, on peut l'utiliser et commencer « petit ». On calcule d'abord directement les solutions des plus petits sous-problèmes, puis on utilise les solutions pour résoudre les problèmes plus grands, et ainsi de suite jusqu'à trouver la solution du problème

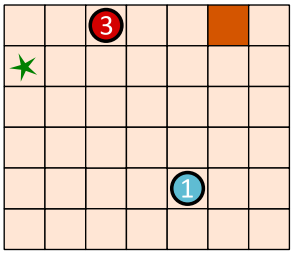
complet. Comme les solutions de sous-problèmes peuvent souvent être utilisées pour résoudre plusieurs problèmes plus grands, elles sont enregistrées pour éviter de devoir répéter les mêmes calculs. La programmation dynamique peut aussi être utile pour compter le nombre de solutions possibles à un problème donné.

Mots clés et sites web

- Diviser pour régner : `https://fr.wikipedia.org/wiki/Diviser_pour_régner_(informatique)`
- Programmation dynamique : `https://fr.wikipedia.org/wiki/Programmation_dynamique`

15. Robots tamponneurs

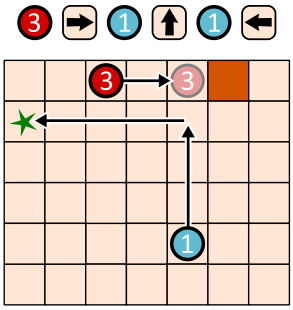
Les robots tamponneurs sont des robots très simples. Ils roulent sur un plateau de jeu divisé en cases.



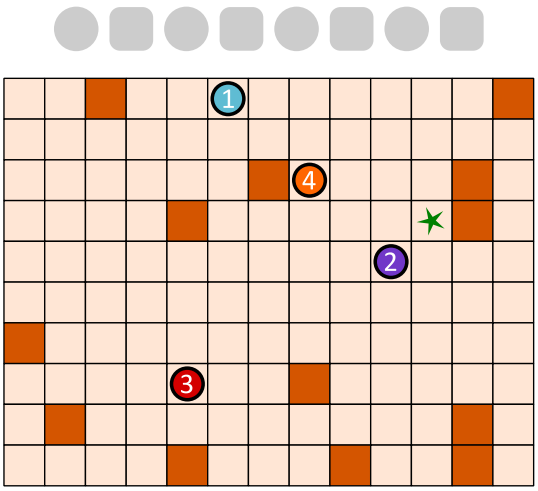
Pour les diriger, on commence par sélectionner un robot tamponneur. On l'envoie ensuite dans une certaine direction à l'aide d'une flèche-commande : en haut, en bas, à gauche ou à droite. Le robot tamponneurs roule ensuite tout droit dans la direction de la commande jusqu'à ce qu'il rencontre un obstacle ou un autre robot. Il s'arrête alors et ne bouge plus jusqu'à ce qu'il reçoive une nouvelle commande.

En utilisant un suite de commandes adaptée, tu dois faire en sorte que le robot tamponneur atteigne le but et y reste.

Le robot tamponneur atteint le but en suivant la suite de commandes suivante :



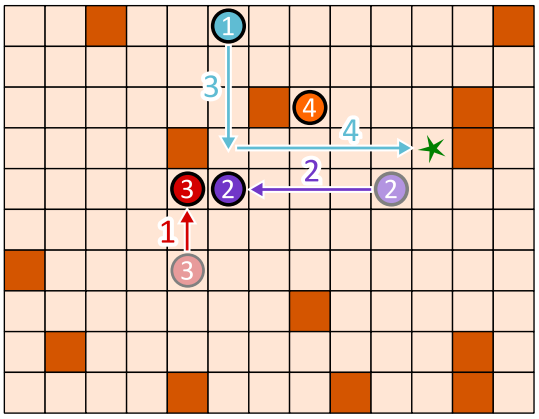
Crée une suite de commandes avec quatre flèches permettant au robot tamponneur d'atteindre le but.



Solution

Voici la bonne suite de commandes : 3 ↑ 2 ← 1 ↓ 1 →

Afin que le robot 1 atteigne le but grâce à une suite de commandes à quatre flèches, trois robots tamponneurs doivent coopérer. D'abord, 3 roule vers le haut jusqu'à ce qu'il rencontre un obstacle. Il devient ainsi lui-même un obstacle pour 2, qui roule vers la gauche. 2 va maintenant arrêter 1 sur son chemin vers le bas à la hauteur du but. 1 va aller à droite après 2 et s'arrêter devant l'obstacle, juste sur la case du but.



Comment trouver la bonne suite de commandes ? Nous pouvons commencer par la fin et réfléchir à quel doit être le dernier mouvement de 1 avant le but. Il n'y a que deux possibilités : a) il vient de la gauche, comme dans notre exemple ; b) il vient d'en haut. Dans ce cas, le robot tamponneur 4 devrait être déplacé vers le haut à droite à l'aide de trois commandes afin d'être un obstacle pour 1. Nous aurions alors besoin de $3 + 2 = 5$ flèches-commandes.

Nous cherchons cependant une solution avec 4 flèches-commandes, donc la solution a), dans laquelle 1 vient de la gauche, doit être la bonne. L'avant-dernier mouvement de 1 est alors de haut en bas. Pour qu'il s'arrête au bon endroit, il faut d'abord que les robots 2 et 3 se déplacent comme montré sur l'image.

C'est de l'informatique !

Dans cet exercice du Castor, plusieurs robots ont travaillé ensemble pour atteindre un but. Ils avaient des tâches différentes : le robot bleu devait atteindre le but alors que les autres servaient d'obstacles.

La répartition des tâches est un aspect important de la robotique. Par exemple, dans un entrepôt automatisé, plusieurs robots différents travaillent ensemble pour ranger des marchandises, les touver et les transporter. Toutes les activités sont coordonnées de manière à ce que le moins de pauses inutiles aient lieu, à ce que les chemins de transport soient les plus courts possible, à ce que le moins d'énergie possible soit utilisé et donc à ce que l'entrepôt marche de manière aussi efficace que possible.

La *robotique en essaim* est un domaine particulier de la robotique. Les robots y sont, comme les robots tamponneurs, des machines simples qui effectuent une tâche en groupe. En agriculture, des robots travaillant en essaim peuvent semer le maïs, observer le développement des plantes et la

qualité du sol et même récolter les céréales. Chaque robot de l'essaim est petit et construit de manière simple, mais l'essaim dans son ensemble est un outil puissant. Ce principe est aussi valable pour les systèmes multi-agents : ce sont de simples unités logicielles qui peuvent ensemble résoudre des problèmes complexes. Le rôle de l'informatique est de développer des algorithmes permmettant une coordination et coopération optimales des agents – qu'il s'agisse de logiciel ou de matériel – composant ces systèmes.

Mots clés et sites web

- Robotique : `https://fr.wikipedia.org/wiki/Robotique`
- Robotique en essaim : `https://fr.wikipedia.org/wiki/Robotique_en_essaim`
- Robotique industrielle : `https://fr.wikipedia.org/wiki/Robotique_industrielle`
- Système multi-agents : `https://fr.wikipedia.org/wiki/Système_multi-agents`

16. Les courses d'Emma

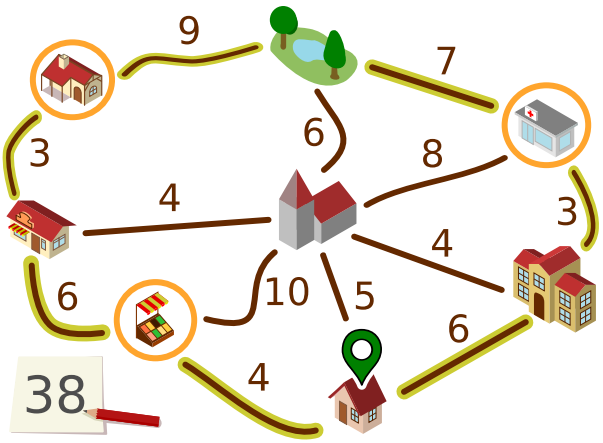
Emma est à la maison . Elle doit faire trois courses et revenir à la maison :

- Aller chercher un paquet au kiosque ,
- Aller acheter des fruits au marché ,
- Aller récupérer un médicament à la pharmacie .

Emma ne sait pas de combien de temps elle aura besoin dans chaque magasin, mais son trajet doit être le plus court possible.

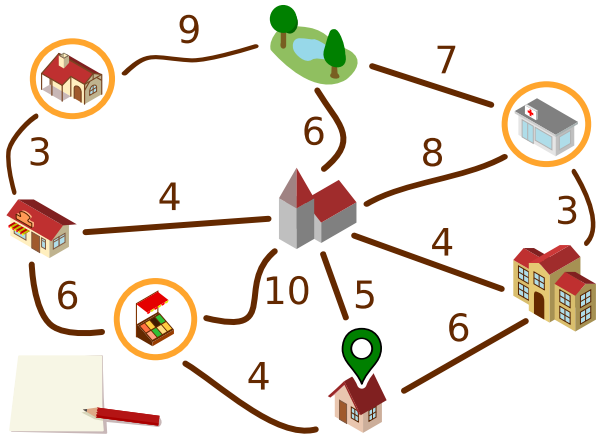
Emma a noté sur un plan de combien de minutes elle a besoin pour parcourir les chemins entre différents endroits de sa ville. Elle a aussi noté quels chemins elle prend pour faire ses courses.

Pour le trajet en entier, Emma a besoin de $6 + 3 + 7 + 9 + 3 + 6 + 4 = 38$ minutes.



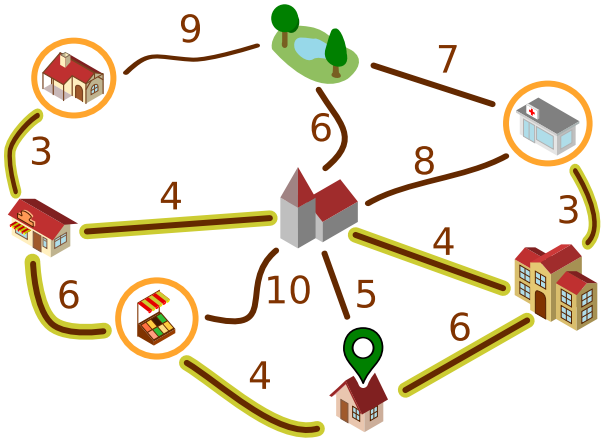
Emma se demande si elle pourrait être plus rapide. Peut-être en faisant l'aller-retour sur certains chemins ?

Détermine le trajet le plus court qu'Emma peut faire pour effectuer ses trois courses.

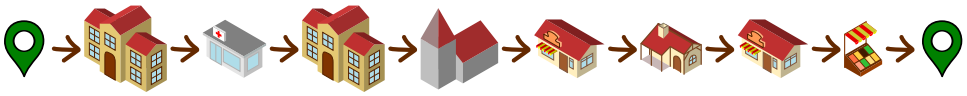


Solution

Voici la bonne réponse :

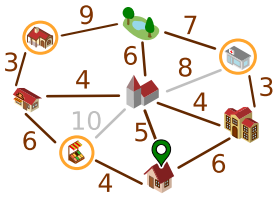


Emma peut faire le trajet suivant le long des chemins sélectionnés (ou dans la direction opposée) :

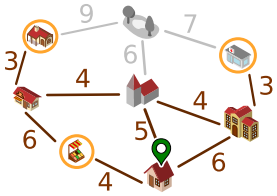


Pour ce trajet, elle a besoin de $6 + 3 + 3 + 4 + 4 + 3 + 3 + 6 + 4 = 36$ minutes.

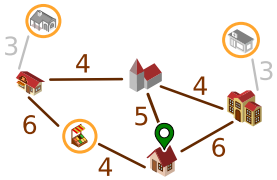
Nous voulons maintenant démontrer qu'il ne peut pas y avoir de trajet plus court. Pour cela, nous utilisons une version simplifiée du plan.



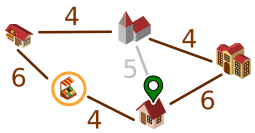
Nous pouvons ignorer les chemins en gris. Il existe des chemins plus courts passant par d'autres endroits entre les endroits qu'ils relient.



Nous pouvons également ignorer le parc. Emma ne doit pas aller au parc, et il existe un chemin plus court pour chaque chemin passant par le parc.

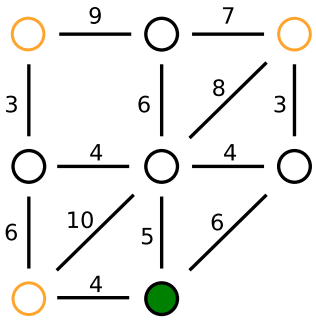


Emma doit aller à la pharmacie et au kiosque. Elle ne peut y aller que depuis la boulangerie et l'école, respectivement. Elle doit faire l'aller-retour entre ces endroits, ce qui dure $3 + 3 = 6$ minutes pour chacun, donc 12 minutes en tout. Nous en prenons note et simplifions la plan en enlevant les deux endroits déjà visités.
Il nous reste à présent le plan à droite. Le début et la fin du trajet se trouvent ici. Il faut passer par les trois endroits, et. Pour cela, le trajet le plus court passe par tous les cinq endroits restants sur le plan en passant par tous les chemins sauf le gris et dure $4 + 6 + 4 + 4 + 6 = 24$ minutes. Avec les 12 minutes de l'étape du haut, on arrive à 36 minutes. Les réflexions précédentes montrent qu'il n'y a pas de trajet plus court.



C'est de l'informatique !

Nous avons utilisé un plan simplifié pour démontrer la bonne réponse. Il aurait été possible de réprésenter le plan de manière encore plus abstraite :



Cette représentation contient toutes les informations importantes pour le trajet d'Emma :

- Les objets : les endroits, avec une mise en évidence des endroits importants pour le trajet ;

- Les relations entre les endroits : les chemins reliant deux endroits avec une indication de la lonueur du chemin.

Les *graphes* sont un outil important pour la modélisation des relations entre objets. Les graphes sont composés de nœuds (représentant les objets) et d'arêtes (reliant des paires d'objets et représentant leur relation). Le plan d'Emma peut être modélisé par un *graphe orienté* dans lequel un nombre (le poids) est indiqué pour chaque relation.

L'informatique s'intéresse aux problèmes qui peuvent être représentés par des graphes et aux algorithmes avec lesquels on peut résoudre ces problèmes. Une question importante relative aux graphes orientés est : quel est le chemin le plus court (ou le plus rapide) entre deux nœuds ? La question de cet exercice du Castor est similaire : quel est le plus court trajet circulaire partant d'un nœud et passant par un ensemble d'autres nœuds ? Beaucoup d'algorithmes capables de calculer le plus court chemin dans un graphe de manière efficace sont connus en informatique. De tels algorithmes sont par exemple utilisés dans les logiciels de navigation.

Mots clés et sites web

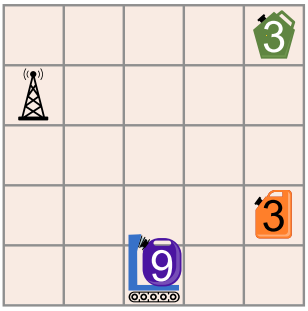
- Théorie des graphes : `https://fr.wikipedia.org/wiki/Théorie_des_graphes`
- Graphe : `https://fr.wikipedia.org/wiki/Graphe_(mathématiques_discrètes)`
- Problème du plus court chemin : `https://fr.wikipedia.org/wiki/Problème_de_plus_court_chemin`

17. La mission de Zérobot

Zérobot a un réservoir de carburant échangeable. Il se déplace dans une grille : vers le haut, le bas, la gauche et la droite. Le niveau de son réservoir baisse de 1 à chaque déplacement d'une case.

Il y a des réservoirs de rechange sur certaines cases ; le chiffre écrit dessus indique leur niveau de carburant. Lorsque Zérobot arrive sur une de ces cases, il change son réservoir indépendemment du niveau de carburant de celui-ci. Il prend le réservoir de rechange, dépose son réservoir précédent sur la case et continue sa route.

La position de Zérobot et le niveau de son réservoir sont représentés comme cela sur l'image : 



Alarme : les réservoirs sont défectueux et pourraient exploser !

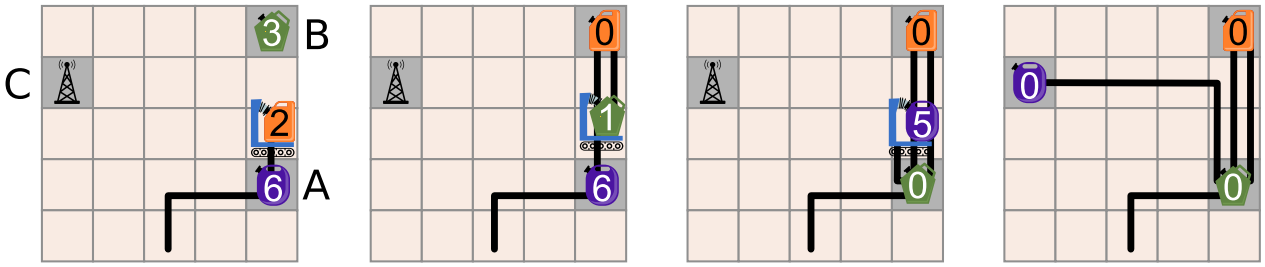
Voici la mission de Zérobot : il doit aller à la station de base  en vidant tous les réservoirs (niveau 0).

Comment Zérobot doit-il se déplacer pour accomplir sa mission ?

Solution

Voici la bonne réponse :

Zérobot peut rouler jusqu'à la station de base en 15 déplacements de manière à ce que tous les réservoirs aient un niveau de carburant 0 :



Pour pouvoir expliquer la bonne réponse plus facilement, nous nommons les cases contenant les réservoirs et la station de base A, B et C, respectivement.

Zérobot se déplace de trois cases jusqu'à la case A et échange (niveau 6) contre (niveau 3). Il se déplace ensuite de trois cases jusqu'à la case B et échange (niveau 0) contre (niveau 3). Il se déplace ensuite à nouveau jusqu'à la case A et échange (niveau 0) contre (niveau 6). Il se déplace ensuite de 6 cases jusqu'à la station de base C. a ensuite le niveau de carburant 0. Mission accomplie !

Est-ce la seule bonne réponse ? Zérobot doit effectuer exactement 15 déplacements : 15 déplacements sont nécessaires pour utiliser tout le carburant disponible, soit $9 + 3 + 3 = 15$ unités. Il n'y a pas assez de carburant pour plus de déplacements. Pour vider tous les réservoirs, Zérobot doit passer par les deux cases contenant les réservoirs de rechange, et même deux fois par la case A. Si Zérobot commençait par passer par la case B, il aurait besoin de 17 déplacements pour atteindre la station de base, ce qui n'est pas possible. La réponse ci-dessus est donc la seule solution possible.

C'est de l'informatique !

Cet exercice du Castor aborde des problèmes fondamentaux de la mobilité autonome : chaque robot mobile autonome (comme une voiture autonome) doit considérer quelle quantité d'énergie, sous forme de carburant ou de charge des batteries, il a à disposition lorsqu'il planifie ses activités. D'un côté, il doit s'assurer d'atteindre une station service ou une station de charge avant d'avoir épuisé ses réserves d'énergie ; d'un autres côté, il y a certaines conditions à respecter. Dans cet exercice, la condition est que tout le carburant doit être utilisé à la fin de la mission. En réalité, les conditions sont plutôt liées à la position et disponibilité de stations de charge. Les logiciels qui dirigent les robots mobiles contiennent des éléments qui assurent un niveau d'énergie suffisant par la recharge (systèmes de contrôle de batterie intelligents).

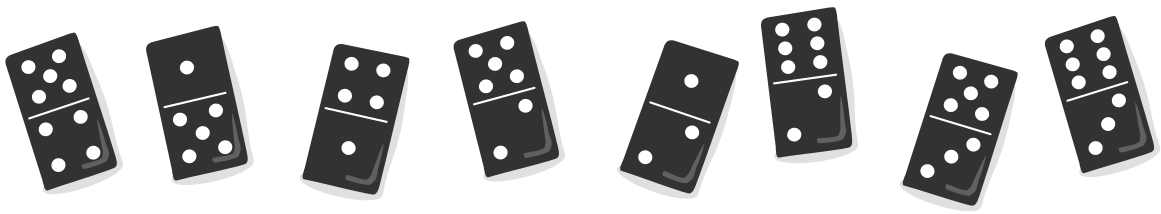
En plus de cela, des programmes informatiques sont utilisés pour planifier et gérer des réseaux de recharge efficaces. Les informaticiens et informaticiennes étudient le problème du placement des stations de charge : les stations de charge pour les robots mobiles doivent être placées de manière à ce qu'un robot ayant une certaine charge minimale puisse atteindre une des stations de charge disponible. Des protocoles de communication entre les stations de charge et les voitures autonomes comme l'OCPP (*Open Charge Point Protocol*) ont été dévelopés.

Mots clés et sites web

- Véhicule autonome : `https://fr.wikipedia.org/wiki/Véhicule_autonome`
- Station de recharge : `https://fr.wikipedia.org/wiki/Station_de_recharge#Infrastructures_de_recharge`

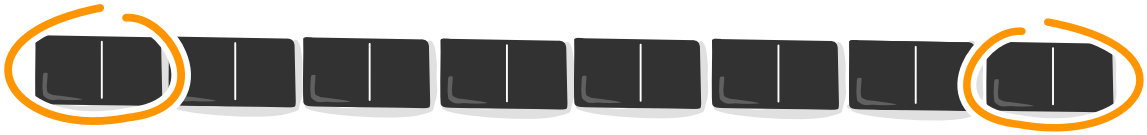
18. Dominos

Chaque pièce de domino a deux cases. Il y a entre 1 et 6 points sur chaque case. Tu as ces huit dominos :



Tu dois aligner ces huit dominos de manière à ce que les cases voisines de deux dominos bout à bout aient toujours le même nombre de points.

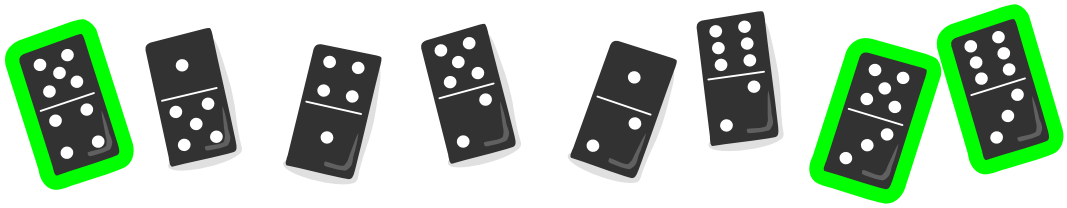
Il y a plusieurs manières d'aligner ces huit dominos, mais certaines pièces ne peuvent jamais se trouver au début ou à la fin de la ligne.



De quelles pièces s'agit-il ?

Solution

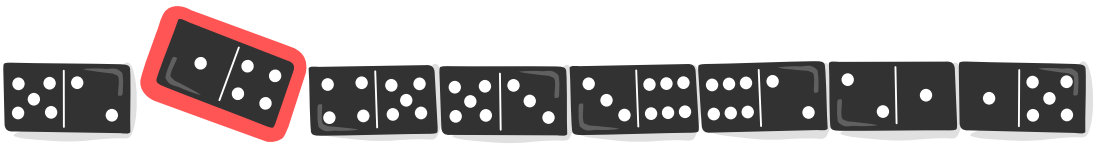
Trois des huits pièces ne peuvent pas être posées au début ou à la fin de la ligne :



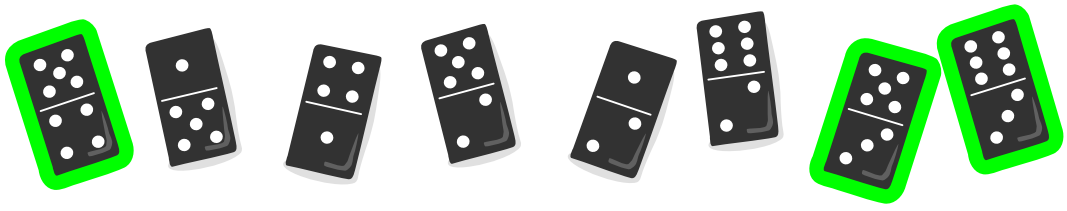
Pour résoudre l'exercice, on considère les nombres de points sur les 16 cases des dominos. Nous comptons combien de fois chaque nombre de points est présent et regardons si cette fréquence est paire ou impaire :

Nombre de points	Fréquence	Paire/impaire
	3	impaire
	3	impaire
	2	paire
	2	paire
	4	paire
	2	paire

Les cases présentes un nombre pair de fois peuvent se trouver en paires à l'intérieur de la ligne ou aux deux extrémités en même temps. Les cases présentes un nombre impair de fois ne peuvent pas toutes se trouver à l'intérieur de la ligne : on ne peut en effet pas trouver de case correspondante pour chacune des cases avec le même nombre de points ; ce n'est possible qu'en cas de fréquence paire. Tu peux voir ci-dessous une ligne dans laquelle un domino avec un point présent trois fois ne passe plus à l'intérieur de la ligne.

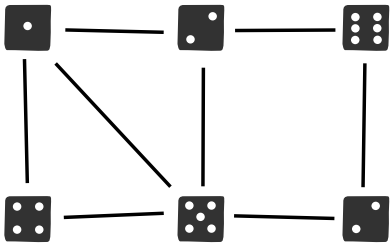


Comme il y a des cases présentes un nombre impair de fois parmi les huit dominos de cet exercice, ce sont eux qui doivent se trouver aux extrémités. Les dominos ayant deux cases avec fréquence paire ne peuvent donc pas être posés aux extrémités de la ligne. Ce sont les dominos suivants :

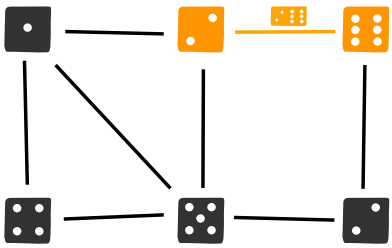


C'est de l'informatique !

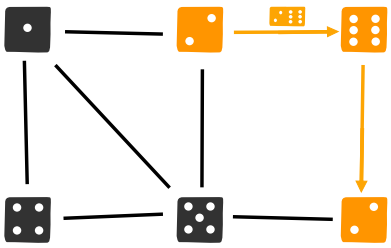
Il y a plusieurs possibilités de poser les dominos de cet exercice du Castor en une ligne correcte. Pour avoir une meilleure vue d'ensemble, les informaticiens et informaticiennes utilisent des *graphes* :



Dans les graphes ci-dessus, on voit des carrés (appelés *nœuds*) représentant les six nombres de points des cases de dominos. Les huit lignes (appelées *arêtes*) les reliant représentent les dominos ; chaque ligne relie deux cases. Par exemple, le domino 2−6 est représenté par l'arête suivante :



Pour résoudre l'exercice, les huit dominos doivent être alignés de manière appropriée. Le nombre de points devant être présent sur la première case du deuxième domino est déjà clair après avoir posé le premier domino, état donné que les cases voisines de deux dominos doivent toujours avoir le même nombre de points. Dans le graphe, c'est visible au fait que les arêtes des dominos pouvant être posés bout à bout se retrouvent au même nœud. Les dominos 2−6 et 6−3, par exemple, peuvent être posés bout à bout car ils contiennent les deux une case à six points :



L'alignement des dominos peut être vu comme un *chemin* (une suite d'arêtes) parcourant le graphe. Ce chemin doit passer *exactement une fois* par chaque arête, car chaque domino doit être posé, mais pas plus d'une seule fois. Un chemin passant exactement une fois par chaque arête est appelé *chemin eulérien*, nommé d'après le mathématicien suisse et créateur de la théorie des graphes Leonhard Euler. Euler a démontré qu'un chemin eulérien existe dans un graphe connexe si et seulement si deux nœuds au maximum sont reliés par un nombre d'arêtes impair.

Mots clés et sites web

- Graphe : `https://fr.wikipedia.org/wiki/Graphe_(mathématiques_discrètes)`
- Nœud : `https://fr.wikipedia.org/wiki/Sommet_(théorie_des_graphes)`
- Arête : `https://fr.wikipedia.org/wiki/Arête_(théorie_des_graphes)`
- Chemin : `https://fr.wikipedia.org/wiki/Chemin_(théorie_des_graphes)`
- Chemin eulérien : `https://fr.wikipedia.org/wiki/Graphe_eulérien`
- Leonhard Euler : `https://fr.wikipedia.org/wiki/Leonhard_Euler`
- Dominos : `https://fr.wikipedia.org/wiki/Dominos_(jeu)`

A. Auteur·e·s des exercices

Akram Ahmed
Somayah Albaradei
Laila Alharthi
Esraa Almajhad
James Atlas
Leonardo Barichello
Liam Baumann
Wilfried Baumann
Leonardo Cavalcante
Diego César
Sarah Chan
Zaheer Chothia
Marios Omar Choudary
Eimear Colreavy
Kris Coolsaet
Lucia Crivelli
María Eugenia Curi
Valentina Dagienė
Darija Dasović
Christian Datzko
Justina Dauksaite
Nora A. Escherle
Gerald Futschek
Bence Gaál
Emily Gates
Anaclara Gerosa
Adam Grodeck
Štefan Gura
Juan Gutiérrez
Josefine Hiebler
Mathias Hiron
Alisher Ikramov
Hyun-seok Jeon
Merel Kämper
Mhairi King
Jia-Ling Koh
Sophie Koh
Víctor Koleszar
Taina Lehtimäki
Marielle Léonard
Carlos Luna
Michael Weigend
Yong Mao
Yoshiaki Matsuzawa
Madhavan Mukund
Natalia Natalia
Tom Naughton
Jalil Nedaeepour
Graciela Oyhenard
Özgür Özdemir
Marika Parviainen
Elsa Pellet

Jean-Philippe Pellet

Zsuzsa Pluhár

Wolfgang Pohl

Estela Ramić

Chris Roffey

Kirsten Schlüter

Eljakim Schrijvers

Rosario Schunk

Giovanni Serafini

Kim Seulki

Vipul Shah

Jacqueline Staub

Alieke Stijf

Gabrielė Stupurienė

Marianne Thut

Monika Tomcsányiová

Jiří Vaníček

Michael Weigend

Manuel Wettstein

Kyra Willekes

B. Partenaires académiques

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

C. Sponsoring

HASLERSTIFTUNG

http://www.haslerstiftung.ch/

Kanton Zürich
Volkswirtschaftsdirektion
Amt für Wirtschaft und Arbeit

Standortförderung beim Amt für Wirtschaft und Arbeit Kanton Zürich

UBS

http://www.ubs.com/

verkehrshaus.ch

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

i-factory (Musée des transports, Lucerne)

senarclens
leu+partner
strategische kommunikation

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

D. Offres supplémentaires

IT tout feu tout flamme : `https://it-feuer.ch/fr/`
En Suisse, un nombre considérable d'organisations s'engagent à promouvoir la prochaine génération d'informaticiennes et d'informaticiens. L'initiative « IT tout feu tout flamme » souhaite unir ces forces et contribuer ensemble à mieux faire connaître le sujet au public dans toute la Suisse. IT tout feu tout flamme présente une variété d'offres destinées au corps enseignant et aux élèves.

Coding club des filles :
`https://www.epfl.ch/education/education-and-science-outreach/fr/jeunepublic/coding-club/`
Programmer une application ? Inventer un jeu vidéo ? Créer une animation ? Si une de ces activités t'intéresse, cet espace est fait pour toi ! Viens échanger et partager tes idées, apprendre à coder et découvrir les métiers liés à l'informatique. Les filles de 11 à 15 ans intéressées par la programmation et l'informatique peuvent participer aux ateliers du Coding club des filles.

Roteco : `https://www.roteco.ch/fr/`
Le projet Roteco existe autour d'une communauté d'enseignantes et enseignants qui souhaitent préparer leurs élèves à évoluer dans une société numérique. Au sein de cette communauté, ils cherchent, testent, développent et partagent des activités de robotique éducative et de science informatique adaptées pour leurs classes. Ils sont informés des derniers événements ou ateliers concernant la robotique et plus largement des activités de science informatique à proximité de leur établissement.

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.