INFORMATIK-BIBER SCHWEIZ
CASTOR INFORMATIQUE SUISSE
CASTORO INFORMATICO SVIZZERA

Exercices et solutions 2023

Années HarmoS 7/8

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

Éditeurs :

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

0101001101010110010010010000010010110101010011
0101001101001010010100010100110101010011010011
0010110101010100110101001100100101001001010000

SSIE

www.svia-ssie-ssii.ch
schweizerischervereinfürinformatikinderausbildung//sociétésuissepourl'informatiquedansl'enseignement//societàsvizzeraperl'informaticanell'insegnamento

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

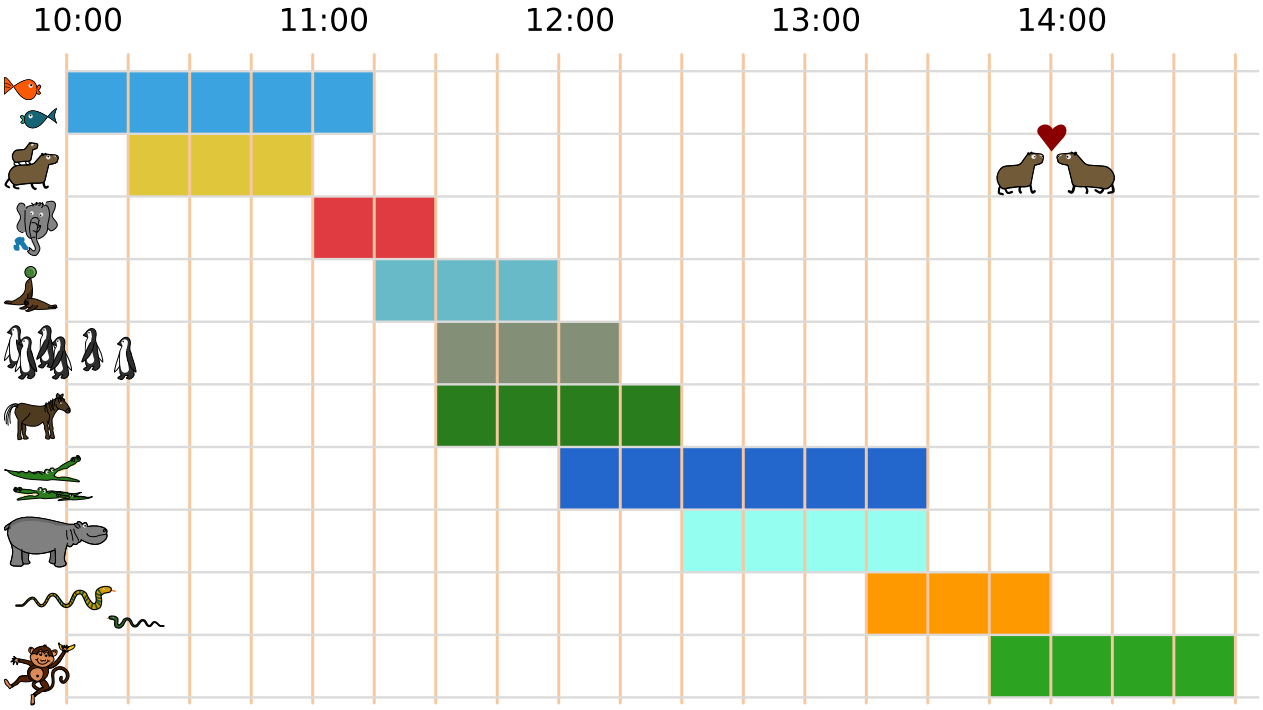
`https://www.castor-informatique.ch/fr/kontaktieren/`
`https://www.castor-informatique.ch/`

Table des matières

1. Visite au zoo

Aujourd'hui, Anja passe la journée au zoo. Elle veut voir le plus de présentations possible.

Voici un programme avec toutes les présentations. Tout en bas, tu vois par exemples que la présentation des singes commence à 13h45 et finit à 14h45.

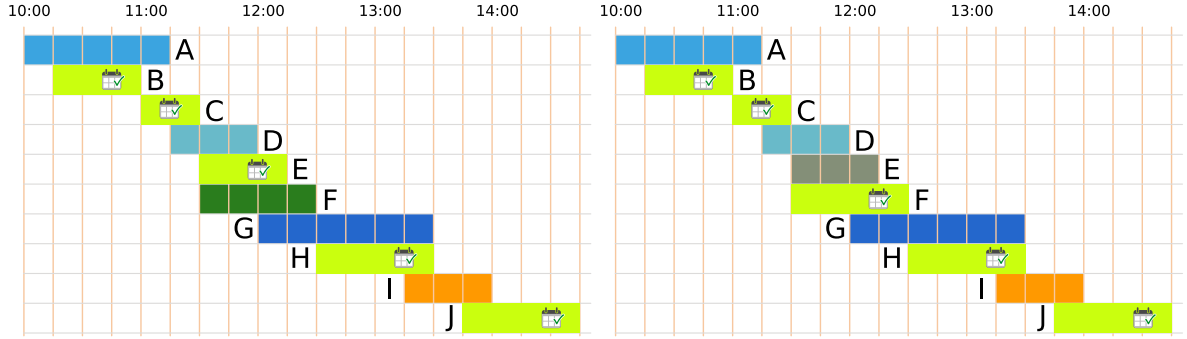


Anja assiste toujours à une présentation en entier, du début à la fin. Peux-tu l'aider ?

Choisis le plus de présentations possible qu'Anja peut voir les unes après les autres.

Solution

Anja peut voir au maximum cinq présentations les unes après les autres. Voici les deux réponses justes :



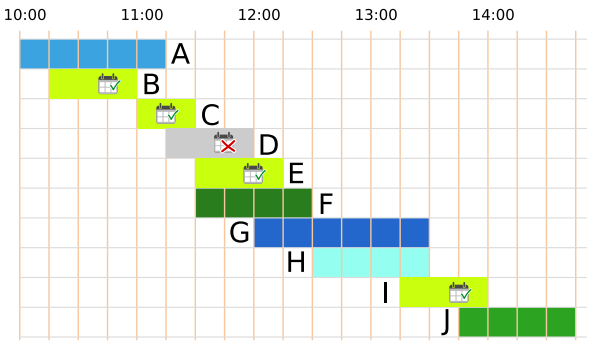
Il y a différentes façons d'arriver aux bonnes réponses.

Un plan de visite pour Anja est une sélection de présentations qu'elle peut voir les unes après les autres. Un moyen de trouver les bonnes réponses est de faire une liste de tous les plans de visite possibles. Les bonnes réponses sont les plans contenant le plus de présentations dans cette liste. Cela prend malheureusement beaucoup de temps pour trouver tous les plans possibles.

Ne pourrait-il pas y avoir de plan de visite avec six présentations ? Essayons d'en faire un. Pour commencer, nous observons la durée des présentations : sur le programme, la journée est divisée en 19 unités de temps d'un quart d'heure chacune. Les présentations durent 2, 3, 4, 5 ou 6 unités de temps.

Unités de temps	**Présentation**
2	C
3	B, D, E, I
4	F, H, J
5	A
6	G

Pour pouvoir mettre le plus de présentations possible dans un plan de visite, nous choisissons les présentations les plus *courtes*. Les six présentations les plus courtes durent en tout 18 unités de temps ($2+3+3+3+3+4$). Les présentations C, D et E font partie des 6 présentations les plus courtes ; mais comme les présentations C et E sont directement l'une après l'autre, Anja ne peut pas aller voir la présentation D entre deux.



Nous devons donc remplacer la présentation D par une autre présentation aussi courte que possible. Il ne reste que des présentations durant au moins quatre unités de temps. Sans la présentation D, nous avons donc besoin d'au moins 19 unités de temps pour voir six présentations : $2+3+3+3+4+4$. Mais quelles que soient les deux présentations à quatre unités de temps que nous choisissons, l'une d'entre elles a lieu en même temps qu'une présentation à 3 unités de temps. Nous devrions donc remplacer l'une d'elles par une présentation d'au moins quatre unités de temps et aurions besoin d'au moins 20 unités de temps en tout pour voir six présentations. Mais nous n'avons que 19 unités de temps à disposition et ne pouvons donc pas faire de plan de visite à plus de cinq présentations.

C'est de l'informatique !

Cet exercice du Castor contient un horaire des présentations du zoo. Ce n'est pas facile de créer de tels horaires ; en informatique, on parle de *séquençage de tâches*. Le zoo aimerait évidemment permettre à ses visiteurs de voir le plus de présentations possible, mais d'autres contraintes doivent aussi être prises en compte. Par exemple, une présentation ne peut être proposée que quand les gardiens animaliers sont disponibles, que les arènes sont libres et que les heures sont compatibles avec le rythme de vie des animaux.

Il existe beaucoup de problèmes de ce type dans la vie quotidienne auxquels les mêmes réflexions peuvent être appliquées, par exemple l'élaboration d'un horaire pour l'école ou la programmation des films dans les salles d'un cinéma. L'élaboration de ces horaires est si compliquée que même ces simples exemples (les horaires de ton école) ne peuvent pas être fait manuellement. Les *processeurs* de ton ordinateurs doivent eux aussi effectuer beaucoup de tâches les unes après les autres. Le programme déterminant quel processeur fait quoi à quel moment est créé très rapidement par le *système d'exploitation* sans que l'on ne le remarque. Le *séquençage de tâches* est un thème important en informatique et en recherche.

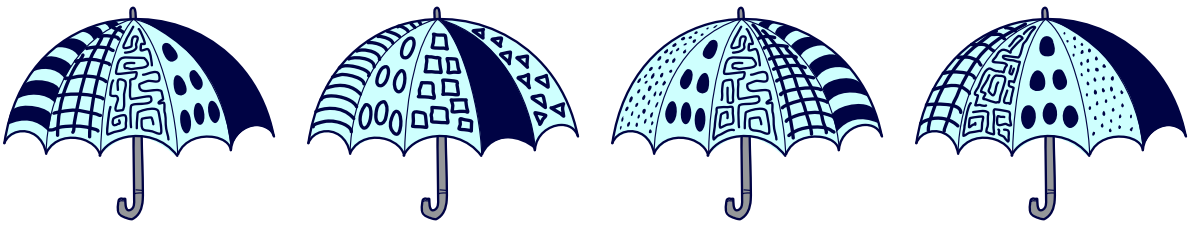
Mots clés et sites web

- Ordonnancement : `https://fr.wikipedia.org/wiki/Ordonnancement_de_travaux_informatiques`
- Système d'exploitation : `https://fr.wikipedia.org/wiki/Système_d'exploitation`
- Séquençage des tâches : `https://fr.wikipedia.org/wiki/Séquençage_de_tâches`

2. Parapluie

Voici le parapluie d'Anna :

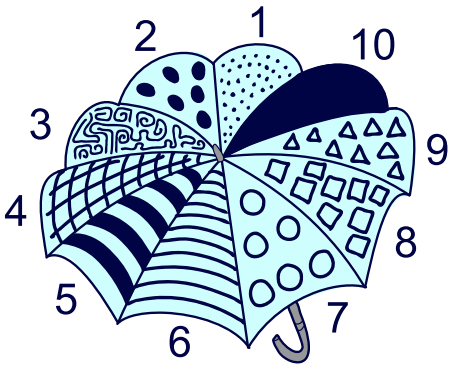
Une des quatre images montre le parapluie d'Anna. Laquelle ?



A) B) C) D)

Solution

Chaque motif n'apparaît qu'une seule fois sur le parapluie d'Anna.



Pour trouver la bonne image, nous comparons chacune des images l'une après l'autre avec le parapluie d'Anna :

- Nous cherchons la position du motif situé tout à gauche du parapluie de la réponse possible sur le parapluie d'Anna,
- Nous vérifions que les motifs voisins soient les mêmes sur le parapluie de la réponse possible que sur le parapluie d'Anna.

	A)	**B)**	**C)**	**D)**
Réponse possible				
Parapluie d'Anna				

Chacune des quatre images montre une suite de seulement cinq motifs et pas tous les dix. Nous ne pouvons pas savoir si la suite de cinq motifs d'une des quatre images correspond à la suite de dix motifs complète du parapluie d'Anna.

L'image C est la seule qui montre un parapluie avec cinq motifs correspondants à ceux présents sur le parapluie d'Anna. Toutes les autres images montrent des suites de motifs qui ne correpondent pas, ou seulement en partie, au parapluie d'Anna. Seule l'image C peut donc montrer le parapluie d'Anna.

C'est de l'informatique !

Les réponses possibles ne montrent qu'une partie de la suite de motifs. Même si elles ne contiennt que des *informations partielles*, nous pouvons déterminer laquelle des quatre images montre le parapluie d'Anna : une image ne montre le parapluie d'Anna que si sa suite de motifs correspond exactement à une partie de la suite de motifs du parapluie d'Anna.

Lors d'une recherche dans un document texte, le même principe est appliqué que pour la recherche de motifs sur les parapluies. L'ordinateur recherche des chaînes de caractères correspondant à une information partielle donnée (le mot recherché) dans le document. Une chaîne de caractères est une suite de caractères (par exemple des lettres, des chiffres, des caractères spéciaux). Lors d'une recherche :

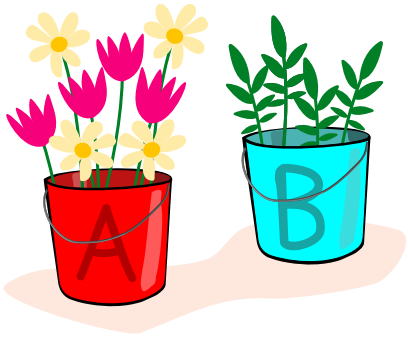
- plus le mot recherché est long, moins il y a de correspondances possible dans le texte et plus la chance de trouver l'endroit recherché dans le texte est élevée,
- plus le mot recherché est court, plus il y a de correspondances possible dans le texte et moins la recherche est exacte.

Pour améliorer la recherche et le parcours des données, différentes méthodes de recherche (ou *algorithmes de recherche*) ont été développées. Leur but est d'effectuer une recherche exacte le plus rapidement possible et de générer un résultat adapté. Ces algorithmes de recherche sont sans cesse améliorés et peuvent parcourir d'immenses quantités de données en très peu de temps (les moteurs de recherche sur internet utilisent de tels algorithmes).

Mots clés et sites web

- Chaîne de caractères, string : `https://fr.wikipedia.org/wiki/Chaîne_de_caractères`
- Algorithme de recherche : `https://fr.wikipedia.org/wiki/Algorithme_de_recherche`

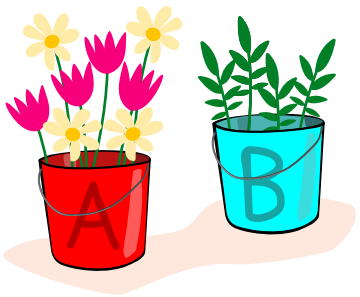
3. Fleuriste

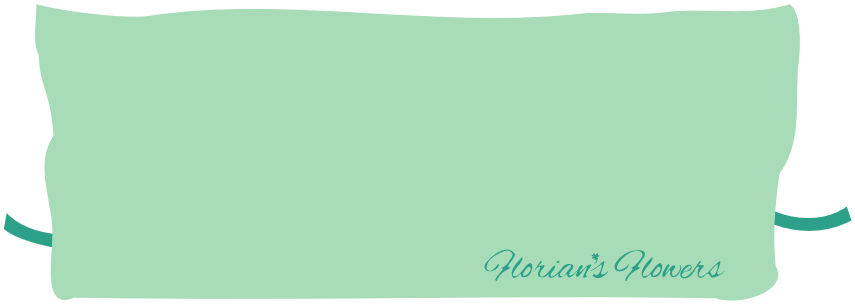


Florian vend des bouquets de fleurs. Il assemble chaque bouquet d'après ces instructions :

1. Prendre une première fleur du seau A.
2. Si cette première fleur est une marguerite, prendre une deuxième marguerite.
3. Prendre maintenant une branche du seau B jusqu'à ce que le bouquet ait quatre éléments. Voilà !

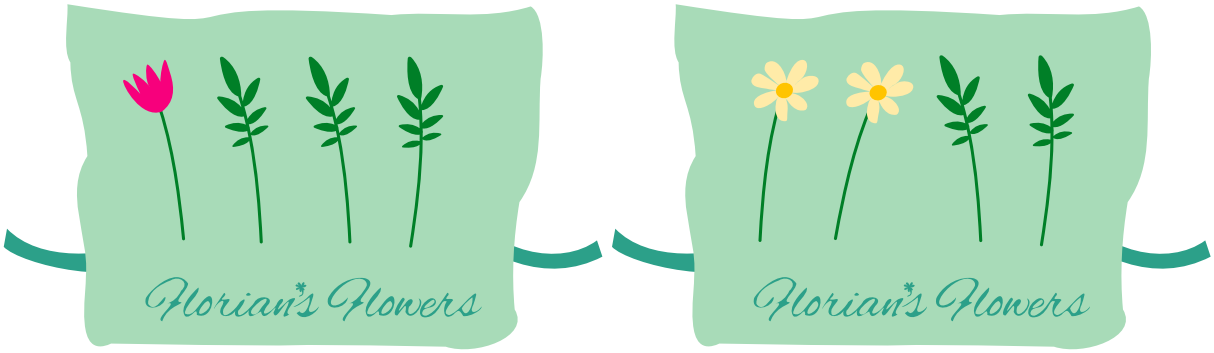
Aide Florian : suis les instructions et choisis des fleurs et des branches pour un bouquet.



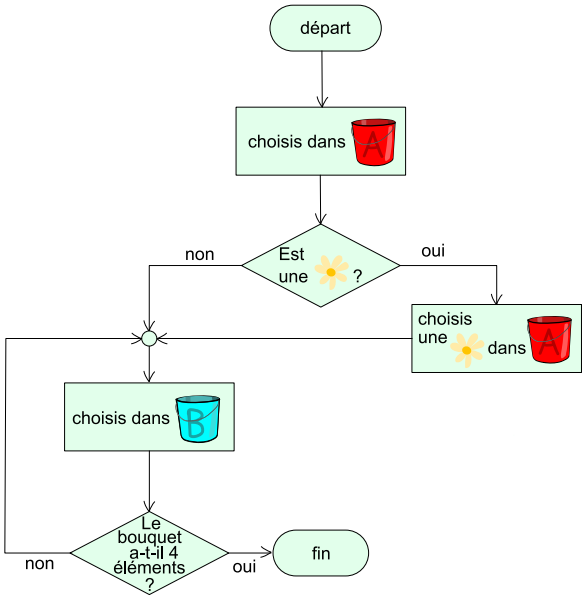


Solution

Il y a deux solutions possibles :



Pour assembler les bouquets de fleurs correctement, Florian doit suivre les instructions. On peut représenter ces instructions à l'aide d'un diagramme :



Après que Florian a choisi la première fleur du seau A, il doit prendre une décision qui dépend de la première fleur. Soit il prend une deuxième marguerite, soit il suit la flèche « non » et prend une branche.

Ensuite, il vérifie si son bouquet a déjà quatre éléments. Si non, il suit la flèche « non » et prend encore une branche avant de vérifier à nouveau le nombre d'éléments.

S'il commence par prendre une marguerite, il va donc prendre une deuxième marguerite puis deux fois une branche. Par contre, s'il commence par prendre une tulipe, il va ensuite directement prendre des branches du seau B jusqu'à avoir 4 éléments, donc 3 branches en tout.

C'est de l'informatique !

Les *instructions* pour l'assemblage de bouquets de fleurs sont claires et pourraient être effectuées par une machine. En informatique, cela s'appelle un *algorithme*. Certaines instructions utilisées ici sont aussi souvent utilisées dans les programmes informatiques :

- La première instruction est la sélection d'un objet au hasard parmi un ensemble d'objets ;
- La deuxième instruction s'appelle une *instruction conditionnelle*, car il faut choisir entre deux possibilités ou plus ;
- La troisième instruction a l'air relativement simple, mais doit être bien structurée dans un programme informatique. La partie intérieure de l'instruction (une instruction en elle-même : « prend une branche dans le seau B ») doit être répétée plusieurs fois jusqu'à ce que le bouquet de fleurs soit composé de quatre éléments. L'instruction intérieure est donc effectuée jusqu'à ce que la condition « le bouquet a quatre éléments » soit remplie. Un telle *instruction itérative* est aussi appelée *boucle*.

Il existe différentes manières de représenter un algorithme. Dans cet exercice, l'algorithme « bouquet » de Florian est formulé par des instructions en langage naturel. Dans l'explication de la solution, il est représenté sous la forme d'un organigramme de programmation.

Les fleuristes sont des artisans. Il existe des traditions et des règles gouvernant l'assemblage des bouquets et couronnes de fleurs. C'est un exemple de situation de la vie quotidienne dans laquelle les instructions et les algorithmes jouent un rôle.

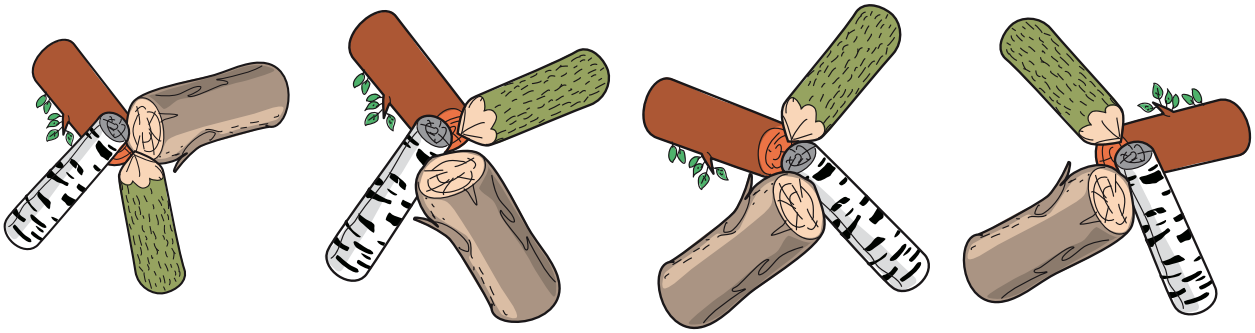
Mots clés et sites web

- Instruction conditionnelle :
  `https://fr.wikipedia.org/wiki/Instruction_conditionnelle_(programmation)`
- Boucle : `https://fr.wikipedia.org/wiki/Structure_de_contrôle#Boucles`
- Organigramme de programmation :
  `https://fr.wikipedia.org/wiki/Organigramme_de_programmation`
- Fleuriste : `https://fr.wikipedia.org/wiki/Fleuriste`

4. Photo

Le castor vient de prendre une photo.

Laquelle des quatre photos a-t-il prise ?



A) B) C) D)

Solution

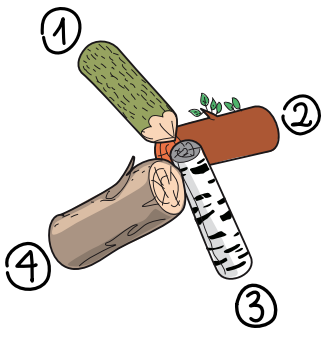
La bonne réponse est D). 

Les troncs que le castor a photographiés sont arrangés en rond. Pour trouver quelle photo est la bonne, nous considérons l'ordre des troncs dans cet arrangemet. Nous choisissons un tronc (par exemple le tronc pointu) et lui donnons le numéro 1. Nous regardons ensuite quel tronc se trouve à sa droite et lui donnons le numéro 2. Nous continuons ainsi jusqu'à ce que chaque tronc ait un numéro. Dans la situation photographiée par le castor, les troncs sont arrangés dans l'ordre 1 – tronc pointu, 2 – tronc brun avec des feuilles, 3 – tronc de bouleau, 4 – gros tronc brunc.

Nous regardons maintenant l'ordre des troncs sur les photos A à D. Comme plus haut, nous commençons par le tronc pointu numéro 1 et allons vers la droite, dans le sens des aiguilles d'une montre :

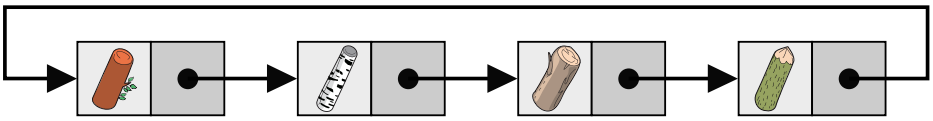
- Photo A : 1 – 3 – 2 – 4
- Photo B : 1 – 4 – 3 – 2
- Photo C : 1 – 3 – 4 – 2
- Photo D : 1 – 2 – 3 – 4

Seule la photo D montre les troncs dans le bon ordre.



C'est de l'informatique !

Dans cet exercice du Castor, nous considérons l'ordre des troncs. Ce qui est visible à l'œil nu avec peu d'*éléments* (ici quatre troncs) nécessite une méthode automatisée pour les problèmes ayant beaucoup d'éléments. Un programme devant traiter des éléments voisins peut utiliser une structure de données adaptée pour stocker les éléments, comme une liste chaînée :



Dans une liste chaînée, chaque élément est stocké dans une cellule différente. En plus, un *pointeur* vers la cellules suivante est aussi stocké dans chaque cellule. Si la dernière cellule contient un pointeur vers la première cellule, il s'agit d'une structure de données cyclique. C'est important dans notre exemple pour pouvoir commencer par n'importe quel tronc tout en parcourant la liste entière.

Mots clés et sites web

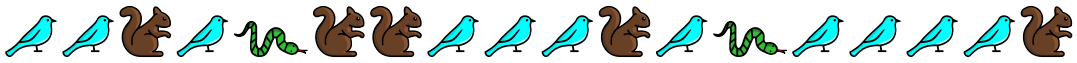
- Liste chaînée : `https://fr.wikipedia.org/wiki/Liste_chaînée`

5. 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	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

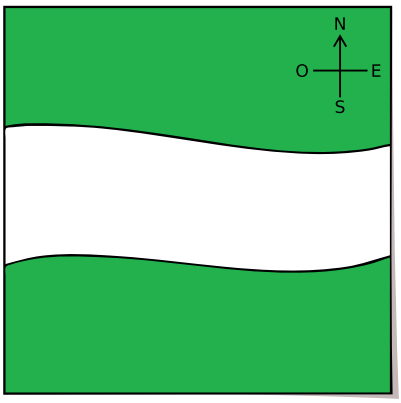
propriétés de ce programme. Une telle analyse de programmes (informatiques) fait partie des tâches des informaticiens et informaticiennes.

Mots clés et sites web

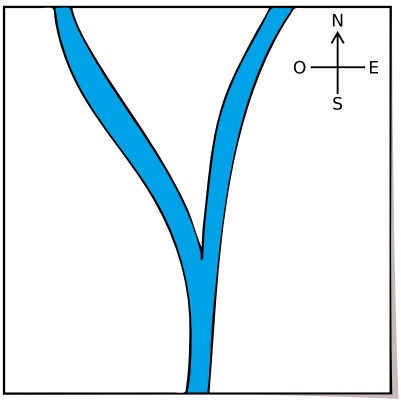
- Variable : `https://fr.wikipedia.org/wiki/Variable_(informatique)`
- État d'un programme : `https://fr.wikipedia.org/wiki/État_(informatique)#Processus`

6. La maison de Karla

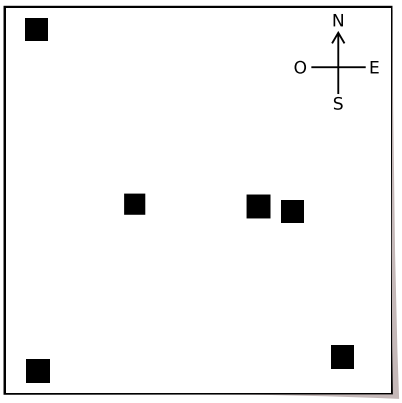
Karla a trois cartes qui montrent exactement la même région. Une carte montre les forêts ; une autre, les rivières, et la troisième, les maisons dans cette région. La maison de rêve de Karla se trouve dans la forêt et près d'une rivière.



Carte des forêts



Carte des rivières

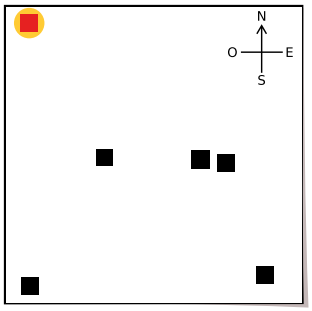


Carte des maisons

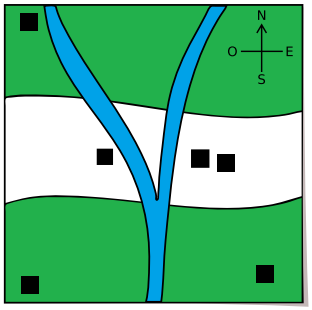
Quelle est la maison de rêve de Karla ?

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)`

7. Riccas

Évelyne a cinq images de riccas. Elle écrit des phrases qui les décrivent.

Son amie Lydia lui montre une sixième image de ricca :

Évelyne remarque alors qu'une de ses phrases sur les riccas est fausse.

Laquelle de ces phrases sur les riccas est fausse ?

A) Tous les riccas ont des dents.

B) Certains riccas ont des ailes.

C) Les riccas ont soit des cornes, soit trois yeux, mais jamais des cornes *et* trois yeux.

D) Si un ricca a exactement deux bras, alors il a aussi exactement deux jambes.

Solution

La réponse D) est la bonne réponse : *Si un ricca a exactement deux bras, alors il a aussi exactement deux jambes.*

La réponse A) est une affirmation qui doit être vraie pour tous les riccas. Si un seul ricca n'avait pas de dents, l'affirmation serait fausse. Comme tous les riccas qu'Évelyne connaît ont des dents, la phrase de la réponse A) n'est pas forcément fausse.

La réponse B) est une affirmation qui ne doit être vraie que pour certains riccas. Comme un des six riccas qu'Évelyne connaît a des ailes, cette phrase est juste pour les six riccas. Même si aucun des six riccas n'avait d'ailes, ce serait possible que d'autres riccas en aient, et la phrase pourrait quand même être juste. Cette phrase ne serait forcément fausse que si Évelyne connaissait tous les riccas et qu'aucun n'avait d'ailes.

La réponse C) relie deux affirmations avec « soit-soit ». Cette affirmation reliée est vraie lorsqu'exactement une des deux affirmations simples est vraie. C'est le cas pour les six images : quatre riccas ont des cornes mais n'ont pas trois yeux et les deux autres riccas n'ont pas de cornes, mais trois yeux. Pour que la phrase soit fausse, il faudrait qu'il y ait un ricca avec des cornes *et* trois yeux, ou un ricca sans cornes et avec un autre nombre d'yeux que trois. Ce n'est le cas d'aucun des six riccas qu'Évelyne ne connaît ; la phrase n'est donc pas forcément fausse.

Il reste la phrase de la réponse D). Elle est formulée à l'aide d'une condition « si-alors » : l'affirmation qui suit le « alors » doit être vraie chaque fois que l'affirmation qui suit le « si » est vraie. La condition « si » est vraie pour tous les six riccas qu'Évelyne connaît : ils ont tous exactement deux bras. Tous les riccas sur les cinq premières images d'Évelyne ont également deux jambes ; pour eux, la phrase d'Évelyne est donc juste. Par contre, le ricca sur l'image de Lydia a plus de deux jambes, cinq exactement. Cette phrase est donc forcément fausse.

C'est de l'informatique !

Le nombre d'ailes, de bras, de jambes et d'yeux, et la présence de dents ou de cornes sont des *propriétés* des riccas. Lorsque l'on décrit des riccas, on formule des *affirmations* sur ces propriétés. Cela mène à un *modèle* de ce que sont les riccas.

Les ordinateurs utilisent beaucoup de modèles. Certains sont formulés de manière explicite, comme un modèle des écolières et écoliers sous forme d'une banque de données contenant noms, dates de naissance et adresses. D'autres modèles sont construits par les ordinateurs lorsqu'on leur donne, par exemple, des images à comparer pour entraîner un réseau de neurones.

Les phrases d'Évelyne – donc son modèle des riccas dans cet exercice – sont formulées sous la forme d'*affirmations logiques*. Certaines ont des quantifications (« tous », « certains », « il existe »), d'autres utilisent des *opérateurs logiques* (« soit-soit », « si-alors »). Ces expressions logiques sont *formalisées* : cela veut dire que leur utilisation et signification sont bien définies.

Cette définition permet de :

- relier des affirmations (simples) à l'aide de quantifications et d'opérateurs pour en faire des affirmations complexes,
- dériver la signification des affirmations complexes à partir des affirmations simples.

Les affirmations logiques sont une méthode répandue pour décrire des modèles en informatique.

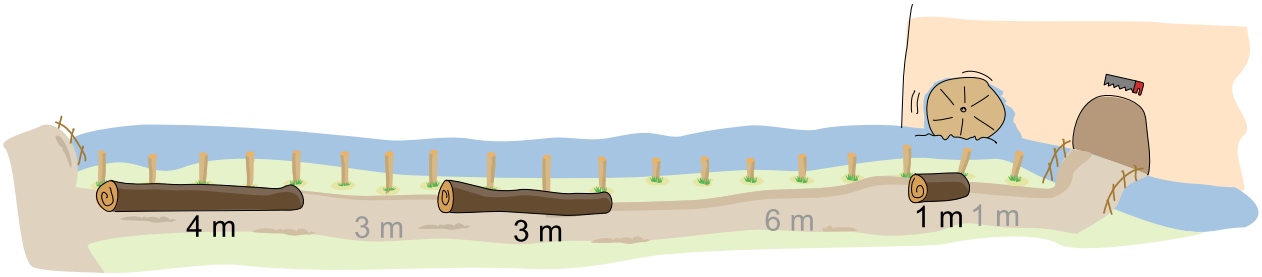
Mots clés et sites web

- Modèle : `https://fr.wikipedia.org/wiki/Modèle`
- Modélisation : `https://fr.wikipedia.org/wiki/Modélisation`
- Apprentissage automatique : `https://fr.wikipedia.org/wiki/Apprentissage_automatique`

8. Les troncs de Timea

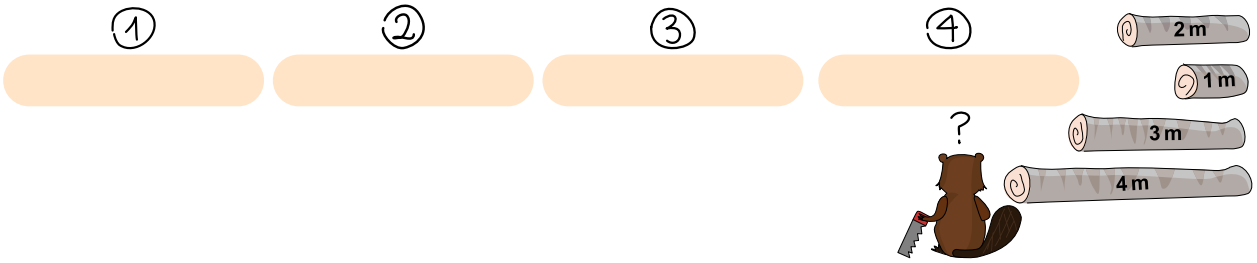
Timea la castor coupe des troncs d'arbre de différentes longueurs, puis les vend. Dès qu'elle a coupé un tronc, elle le pose sur le chemin long de 18 mètres. Timea suit pour cela la règle suivante : en commençant à gauche, elle place le tronc dans le premier espace vide assez grand pour l'y mettre.

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



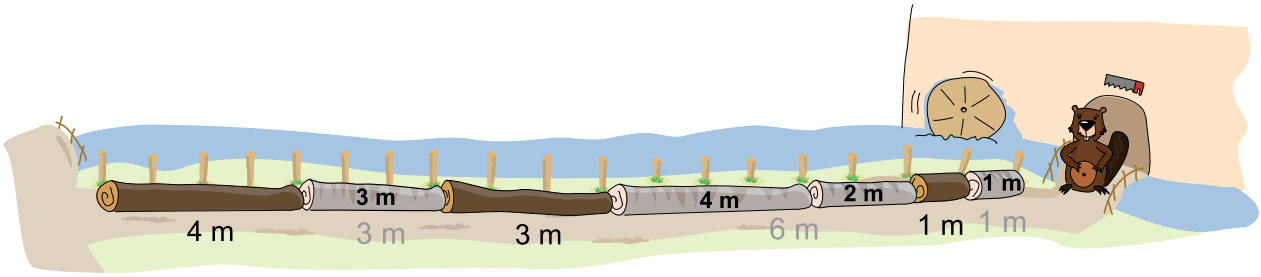
Timea veut maintenant couper quatre troncs longs de 1, 2, 3, et 4 mètres.

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

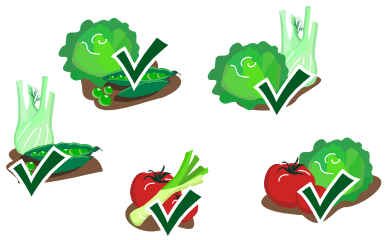
stratégie peut mener à de mauvais résultats : les troncs doivent être placés dans un certain ordre pour pouvoir remplir tous les espaces vides sur le chemin.

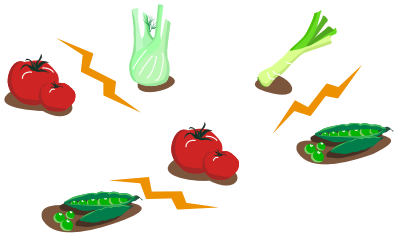
Mots clés et sites web

- 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)`

9. Jardin potager

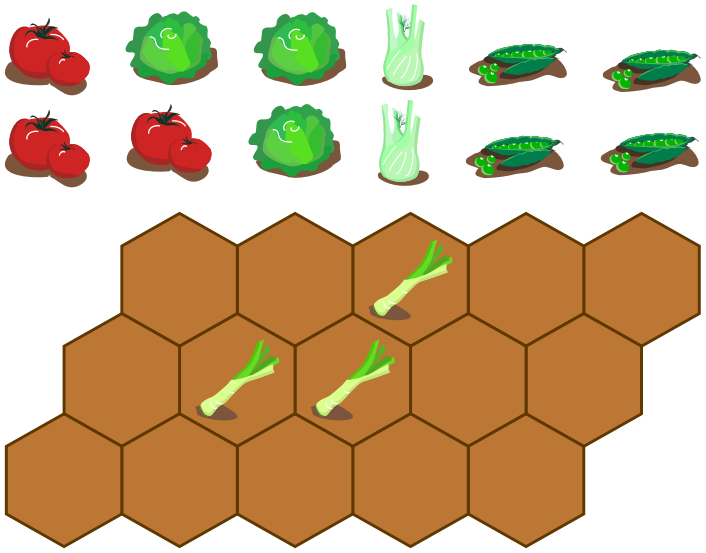
Lisa prépare un jardin potager. Elle veut y cultiver cinq sortes de légumes différentes. Certaines sortes de légumes se supportent bien et sont compatibles ✔, d'autres sont incompatibles ↯ :





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.

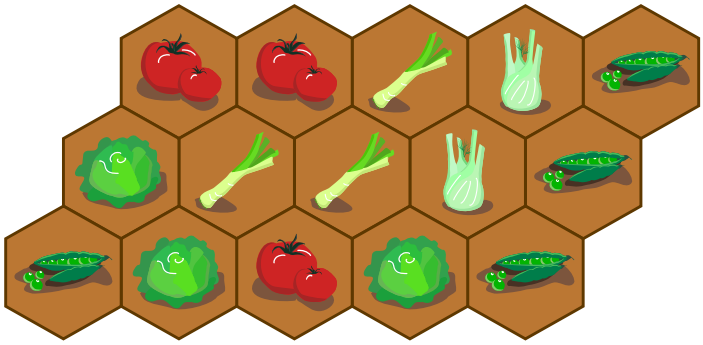


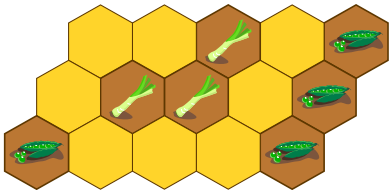
Lisa plante en suivant la règle suivante : les légumes incompatibles ne peuvent pas être plantés dans des domaines qui se touchent.

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

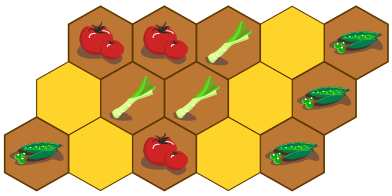
Solution

Voici la bonne réponse :

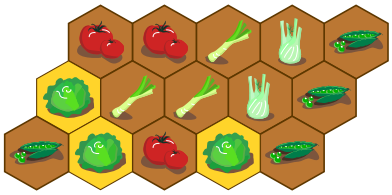




Comme les petits pois ne sont pas compatibles avec les poireaux, Lisa ne plante pas de petits pois dans les domaines jaunes. Il ne reste que les autres domaines pour le petits pois.



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 !

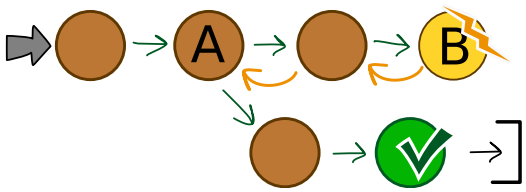
Pour planter des légumes afin d'avoir une récolte aussi grande que possible, il faut respecter beaucoup de *conditions* : chaque sorte a des besoins de place, de nutriments et de lumière différents, par exemple. Dans cet exercice du Castor, nous ne considérons qu'une sorte de condition : la compatiblité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

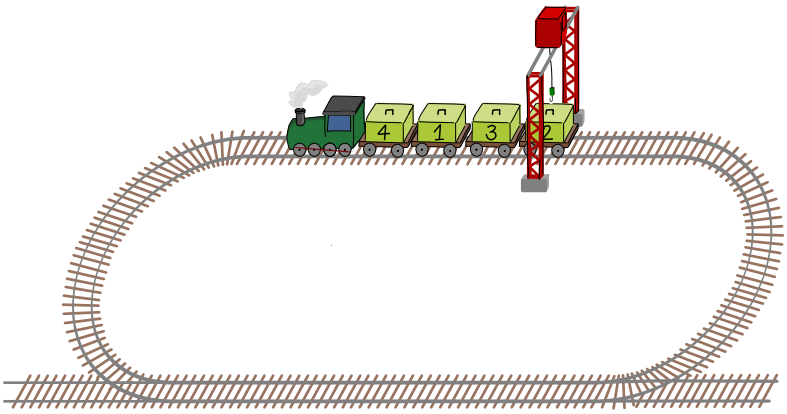
Ce retour en arrière est appelé *retour sur trace* en informatique (*backtracking* en anglais).

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`

10. Train de marchandises

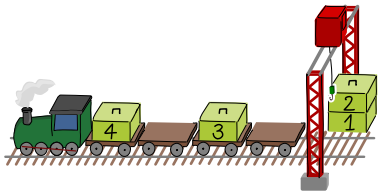
Un train tire des wagons chargés de caisses numérotées. La grue est à une position fixe et décharge les caisses. Pour décharger une caisse, la caisse doit être positionnée directement sous la grue.



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.

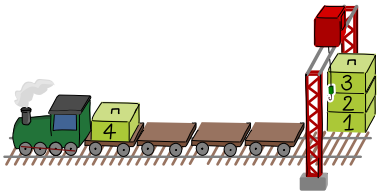
Voilà comment il décharge les caisses 1, 2, 3 et 4 :

Tour 1 :



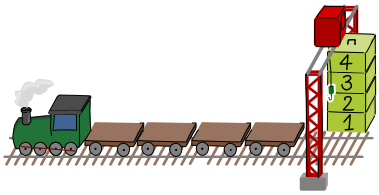
Il saute la caisse 4, décharge la caisse 1, saute la caisse 3 et décharge la caisse 2.

Tour 2 :



Il saute la caisse 4 et décharge la caisse 3.

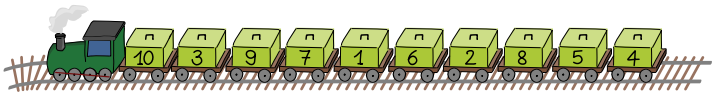
Tour 3 :



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 ?



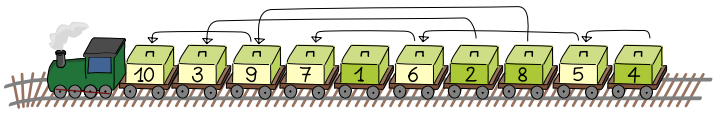
A) 1 tour
B) 2 tours
C) 3 tours
D) 4 tours
E) 5 tours
F) 6 tours
G) 7 tours
H) 8 tours
I) 9 tours
J) 10 tours

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.

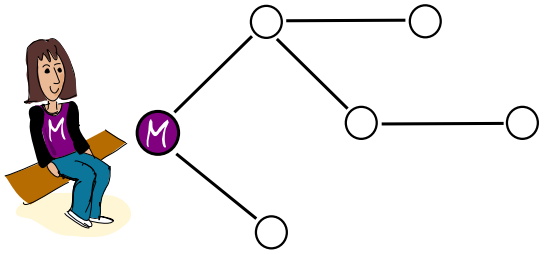
Mots clés et sites web

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

11. Le village de Martina

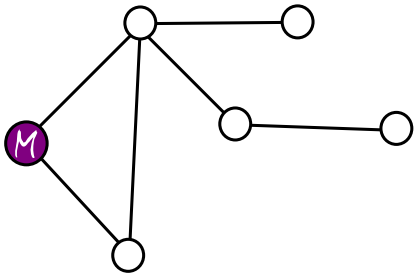
Il y a six maisons dans le village de Martina. Il y a aussi des chemins pour aller d'une maison à la suivante. Martina met le même temps à parcourir chacun de ces chemins.

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.

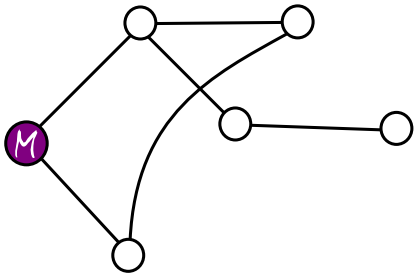


Il existe bien sûr aussi une vraie carte du village avec tous les chemins.

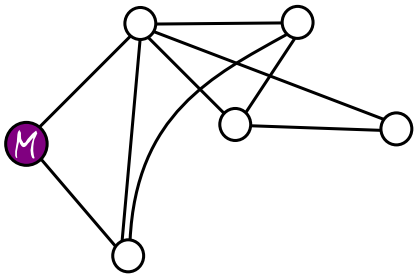
Lequel de ces dessins ne peut-il pas *être la vraie carte ?*



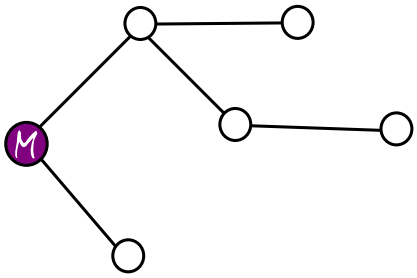
A)



B)



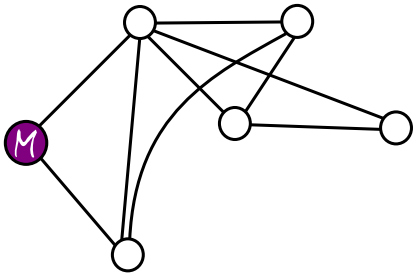
C)



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.

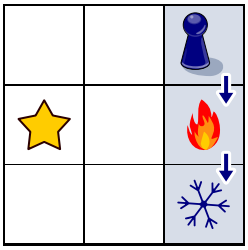
Mots clés et sites web

- 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`

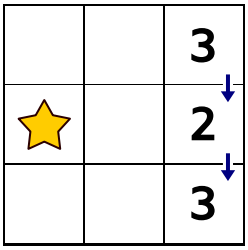
12. Chaud ou froid

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.



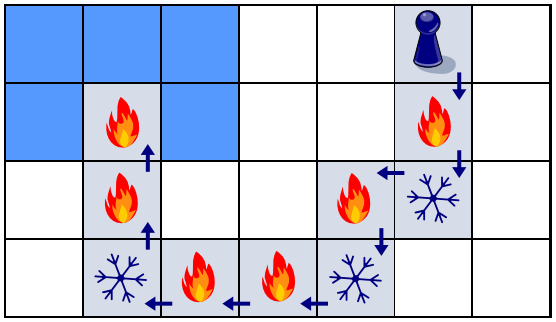
Pour commencer, Nina et Daniel prennent un petit plateau. Nina cache le trésor sur la case avec l'étoile. Daniel commence en haut à droite et fait deux pas en suivant les flèches. Après chaque pas, Nina lui dit s'il est plus près ou plus loin du trésor qu'avant.



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.

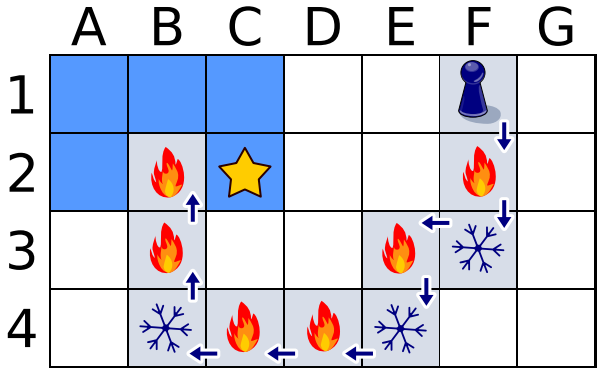
Ils prennent maintenant une plus grande planche de jeu. Nina cache le trésor sur une des cases bleues. L'image montre les pas de Daniel et ce que Nina dit après chaque pas.

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

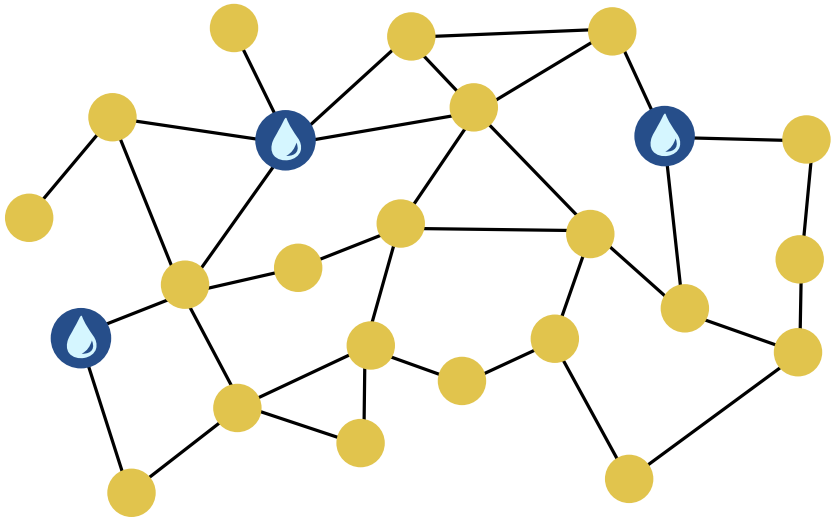
- Distance de Manhattan : `https://fr.wikipedia.org/wiki/Distance_de_Manhattan`
- Distance de Hamming : `https://fr.wikipedia.org/wiki/Distance_de_Hamming`
- Distance de Levenshtein : `https://fr.wikipedia.org/wiki/Distance_de_Levenshtein`

13. 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.

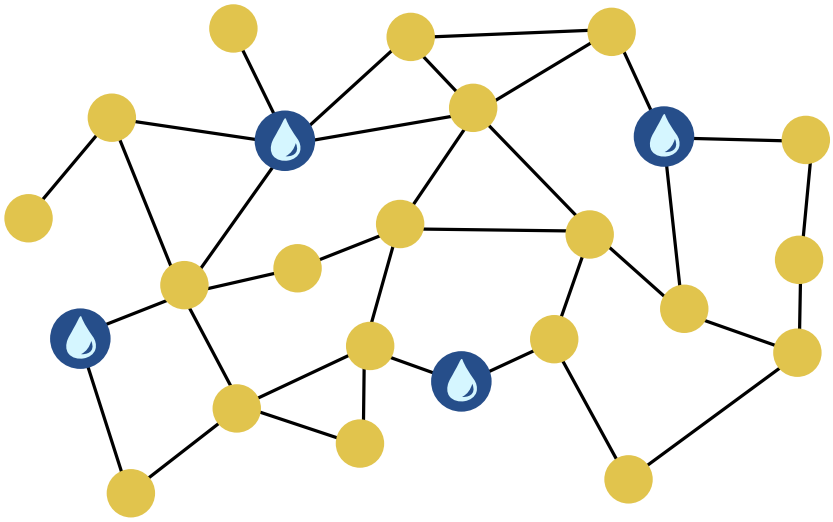
Voici un plan de la ville. Les lignes sont les tronçons de rue et les points les coins de rue. Il y a déjà des fontaines à trois coins de rue.



Installe une fontaine supplémentaire pour satisfaire la maire.

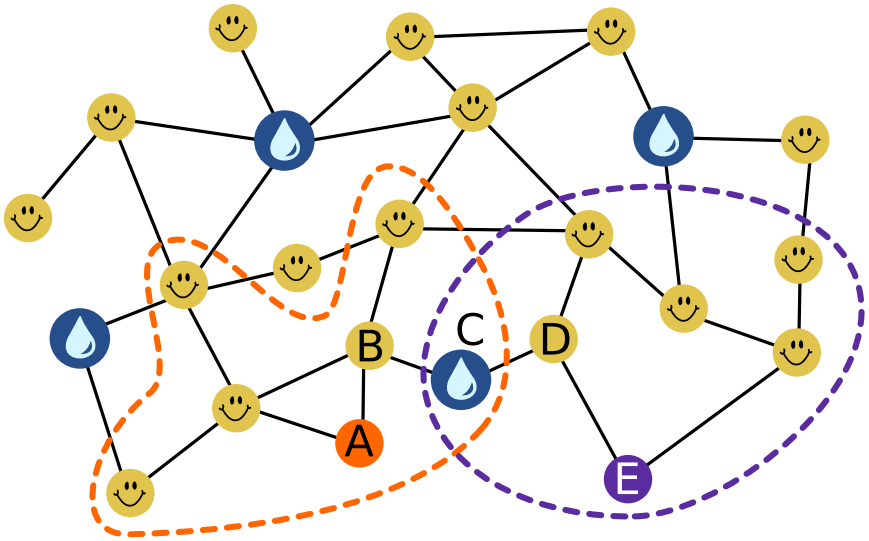
Solution

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

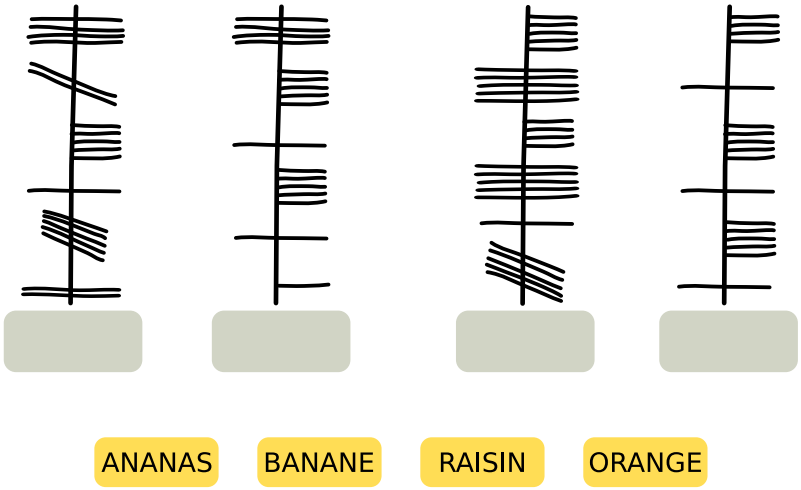
- Ensemble dominant : `https://fr.wikipedia.org/wiki/Ensemble_dominant`
- Théorie des graphes : `https://fr.wikipedia.org/wiki/Théorie_des_graphes`
- Analyse des réseaux sociaux : `https://fr.wikipedia.org/wiki/Analyse_des_réseaux_sociaux`

14. 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.

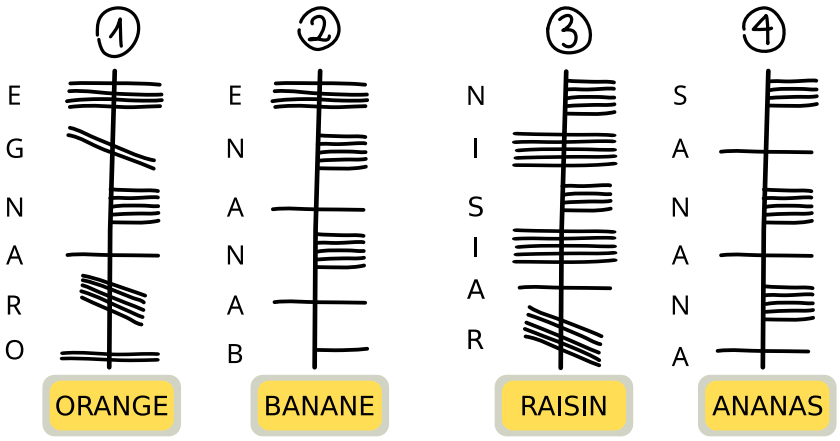
Sue utilise l'écriture oghamique comme code secret. Elle écrit ainsi quatre mots – ses fruits préférés : ANANAS, BANANE, RAISIN et ORANGE.

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.

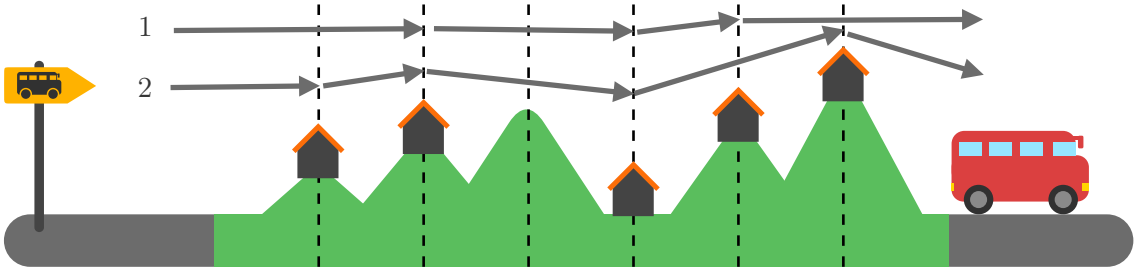
En informatique aussi, il faut déchiffrer des textes et des symboles – après qu'ils ont été encryptés pour le transfert de données sécurisé. Pour cela, des méthodes très différentes de celles utilisées pour coder des mots dans d'autres écritures sont appliquées. De tels chiffres simples sont trop faciles 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`

15. 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

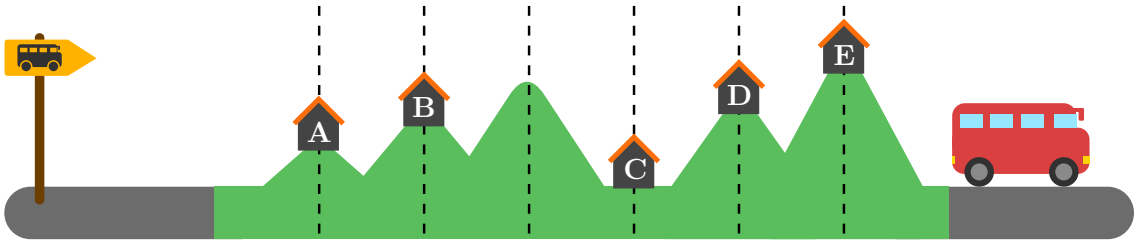
E) 6 randonnées

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`

16. Les courses d'Emma

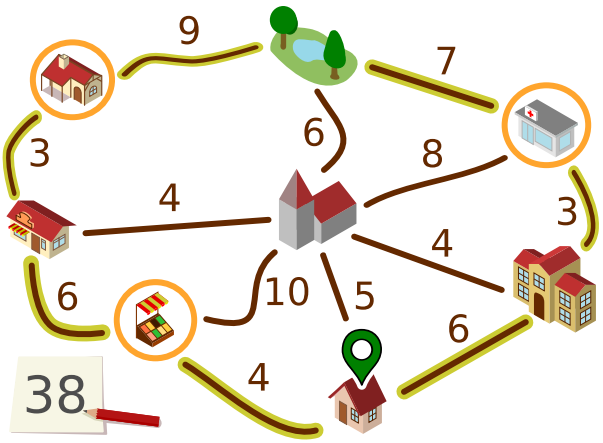
Emma est à la maison . Elle doit faire trois courses et revenir à la maison :

- 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.

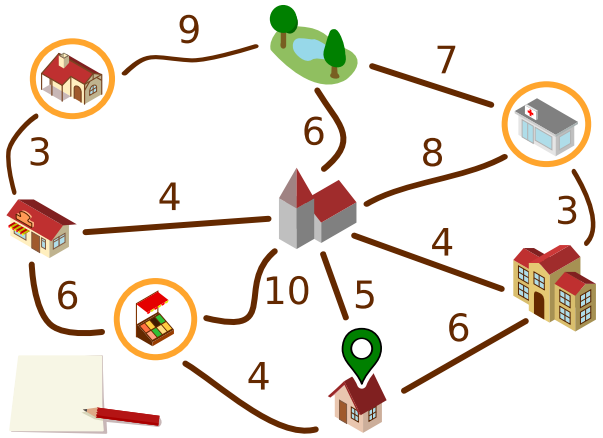
Emma a noté sur un plan de combien de minutes elle a besoin pour parcourir les chemins entre différents endroits de sa ville. Elle a aussi noté quels chemins elle prend pour faire ses courses.

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



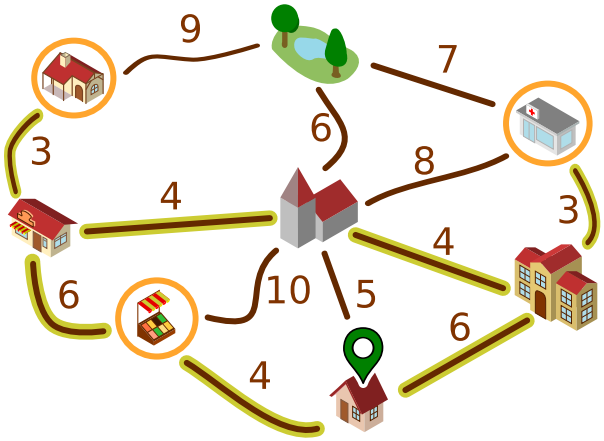
Emma se demande si elle pourrait être plus rapide. Peut-être en faisant l'aller-retour sur certains chemins ?

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

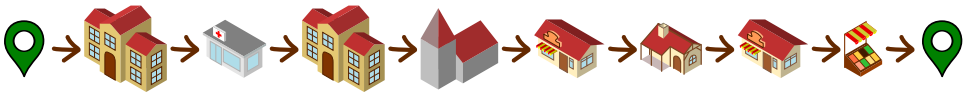


Solution

Voici la bonne réponse :

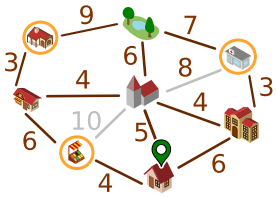


Emma peut faire le trajet suivant le long des chemins sélectionnés (ou dans la direction opposée) :

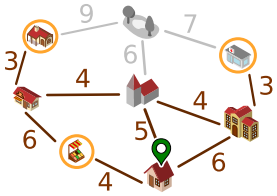


Pour ce trajet, elle a besoin de $6 + 3 + 3 + 4 + 4 + 3 + 3 + 6 + 4 = 36$ minutes.

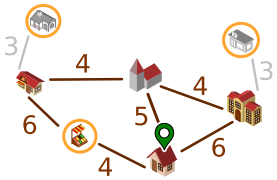
Nous voulons maintenant démontrer qu'il ne peut pas y avoir de trajet plus court. Pour cela, nous utilisons une version simplifiée du plan.



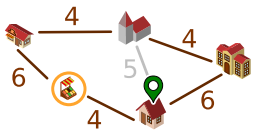
Nous pouvons ignorer les chemins en gris. Il existe des chemins plus courts passant par d'autres endroits entre les endroits qu'ils relient.



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.

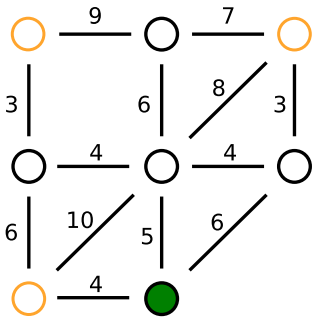


Emma doit aller à la pharmacie et au kiosque. Elle ne peut y aller que depuis la boulangerie et l'école, respectivement. Elle doit faire l'aller-retour entre ces endroits, ce qui dure $3 + 3 = 6$ minutes pour chacun, donc 12 minutes en tout. Nous en prenons note et simplifions la plan en enlevant les 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`

A. Auteur·e·s des exercices

Somayah Albaradei

Laila Alharthi

Aldrich Ellis Catapang Asuncion

Leonardo Barichello

Liam Baumann

Wilfried Baumann

Javier Bilbao

Diego César

Sarah Chan

Marios Omar Choudary

Eimear Colreavy

Kris Coolsaet

Valentina Dagienė

Darija Dasović

Christian Datzko

Justina Dauksaite

Nora A. Escherle

Gerald Futschek

Bence Gaál

Emily Gates

Juan Gutiérrez

Josefine Hiebler

Mathias Hiron

Alisher Ikramov

Hyun-seok Jeon

Merel Kämper

David Khachatryan

Vaidotas Kinčius

Mhairi King

Jia-Ling Koh

Sophie Koh

Víctor Koleszar

Taina Lehtimäki

Angélica Herrera Loyo

Carlos Luna

Yong Mao

Yoshiaki Matsuzawa

Madhavan Mukund

Natalia Natalia

Tom Naughton

Marika Parviainen

Elsa Pellet

Jean-Philippe Pellet

Zsuzsa Pluhár

Wolfgang Pohl

Estela Ramić

Chris Roffey

Kirsten Schlüter

Eljakim Schrijvers

Giovanni Serafini

Alieke Stijf

Marianne Thut

Monika Tomcsányiová

Svetlana Unković

Florentina Voboril

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

http://www.haslerstiftung.ch/

Standortförderung beim Amt für Wirtschaft und Arbeit Kanton Zürich

http://www.ubs.com/

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

i-factory (Musée des transports, Lucerne)

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.