INFORMATIK-BIBER SCHWEIZ
CASTOR INFORMATIQUE SUISSE
CASTORO INFORMATICO SVIZZERA

Exercices et solutions 2023

Toutes les catégories

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

Éditeurs :

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

0101001101010110010010010000010010110101010011
0101001101001010010100101010010110101010011010
0100100101001001001000010

SSIE

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

Ont collaboré au Castor Informatique 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

Ont collaboré au Castor Informatique 2023 i

Préambule iii

Table des matières v

1. Moitiés de pommes 1

2. Terre – eau 5

3. Tri par chapeau 9

4. Visite au zoo 13

5. Parapluie 17

6. Fleuriste 21

7. Photo 25

8. L'arbre magique 29

9. La maison de Karla 33

10. Graines de carottes 35

11. Le trésor de Barbe-de-Castor 39

12. Riccas 43

13. Les troncs de Timea 47

14. Jardin potager 51

15. Train de marchandises 55

16. Le village de Martina 57

17. Chaud ou froid 61

18. Briques castor 65

19. Répartition des tâches 69

20. Fontaine 73

21. Chantier castor 77

22. Ogham 81

23. Randonnée 85

24. Robots tamponneurs **89**

25. Les courses d'Emma **93**

26. La mission de Zérobot **97**

27. Raccordement **101**

28. Notation postfixe **105**

29. Cadenas **109**

30. Dominos **113**

31. Nouveau pantalon **117**

32. Détecteur de conflit **121**

33. Peinture récursive **125**

34. Décryptage **127**

A. Auteur·e·s des exercices **129**

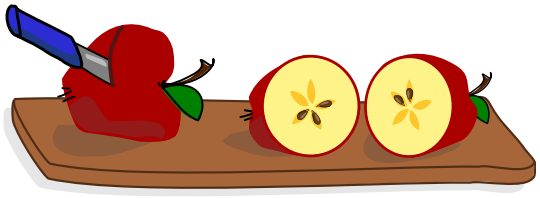
B. Partenaires académiques **131**

C. Sponsoring **132**

D. Offres supplémentaires **133**

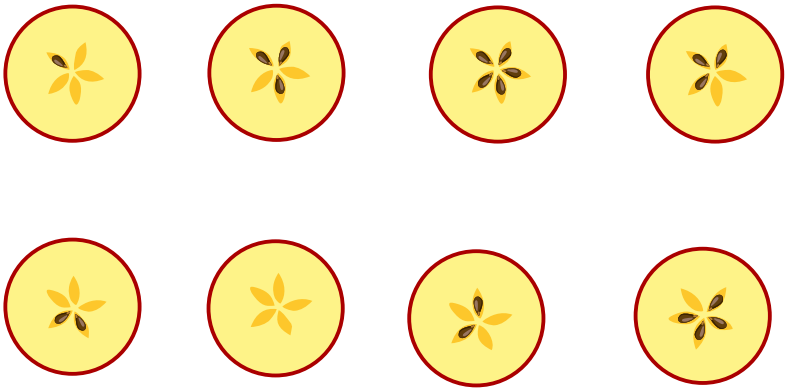
1. Moitiés de pommes

On peut partager des pommes en deux, ce qui donne la moitié du bas et la moitié du haut. Certains pépins de pomme restent dans la moitié du haut, les autres dans la moitié du bas. On peut voir que les moitiés vont ensemble en regardant les pépins et les trous :



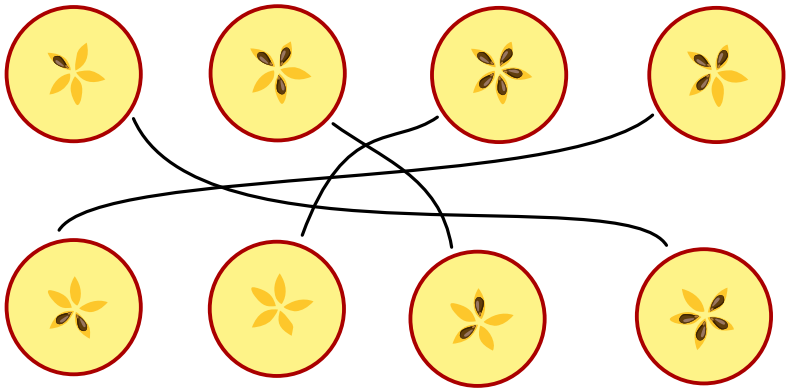
On fait cela pour Noël en Tchéquie. Gala partage quatre pommes. Elle arrange les moitiés du haut et celles du bas en deux rangées.

Quelles moitiés de pommes vont ensemble ? Relie les moitiés de pommes.

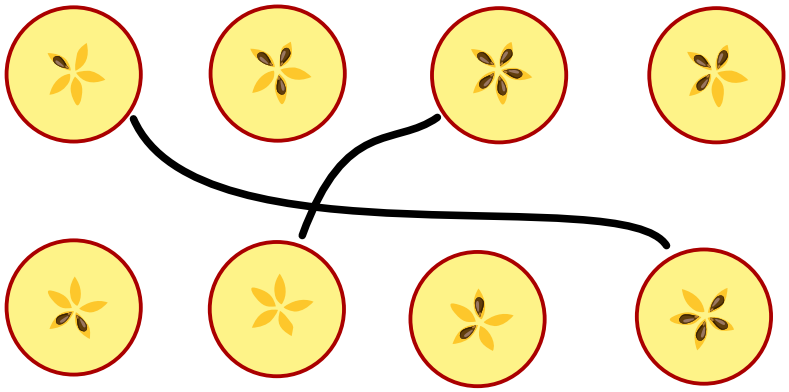


Solution

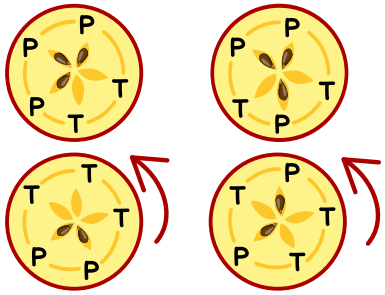
Voici la bonne réponse :



Chaque pomme a cinq pépins. Deux moitiés de pomme allant ensemble doivent donc avoir cinq pépins en tout. Ces moitiés peuvent donc facilement être reliées :



Les quatre moitiés restantes ne sont pas si faciles à relier. Les deux moitiés du haut ont trois pépins chacune, les deux moitiés du bas deux pépins chacune. Il faut donc regarder le motif formé par les pépins plus exactement, car si deux moitiés vont ensemble, leurs motifs vont aussi ensemble. Pour vérifier cela, il peut être nécessaire de tourner les moitiés de pomme. Les moitiés de pommes peuvent ensuite être reliées comme sur l'image ci-dessous : à gauche, la moitié du haut a trois pépins côte à côte, puis deux trous (P-P-P-T-T), la moitié du bas a trois trous côte à côte, puis deux pépins (T-T-T-P-P) : les moitiés vont ensemble. Les moitiés de droite vont également ensemble : la moitié du haut a le motif P-T-P-T-P (en commençant en haut au centre), et celle du bas T-P-T-P-T.



C'est de l'informatique !

On a vu dans l'explication de la réponse que si deux moitiés de pomme vont ensemble, il n'y a pas que le nombre de pépins qui correspond, mais aussi l'ordre des pépins et des trous. Il faut donc également considérer cet ordre lors de l'attribution des moitiés de pomme. Il ne suffit pas de savoir combien de pépins a chaque moitié.

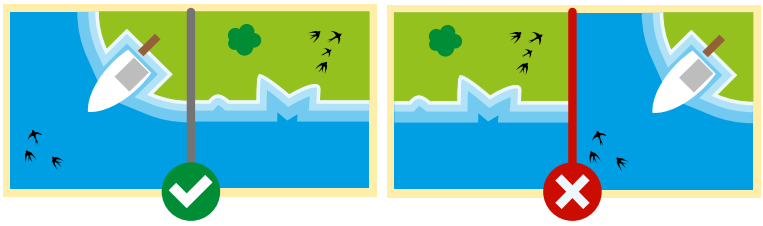
Des questions semblables sont présentes dans des problèmes qui doivent être résolus à l'aide d'ordinateurs. Les informaticiennes et informaticiens doivent réfléchir à la manière dont les informations que le programme doit considérer sont décrites. On essaie en général de faire aussi simple que possible : les programmes simples ont moins de risque de contenir des erreurs. Pour le problème des moitiés de pomme, il semblait d'abord suffisant de décrire les moitiés uniquement à l'aide du nombre de pépins. Ensuite, on a vu que ce n'était pas suffisant dans toutes les situations. Il faut donc pouvoir décrire un ordre afin de pouvoir décrire les moitiés de pomme dans un programme informatique. C'est possible à l'aide de la structure de données appelée *liste*, par exemple, qui existe dans la plupart des langages de programmation.

Mots clés et sites web

- Ordre
- Liste : `https://fr.wikipedia.org/wiki/Liste_(informatique)`

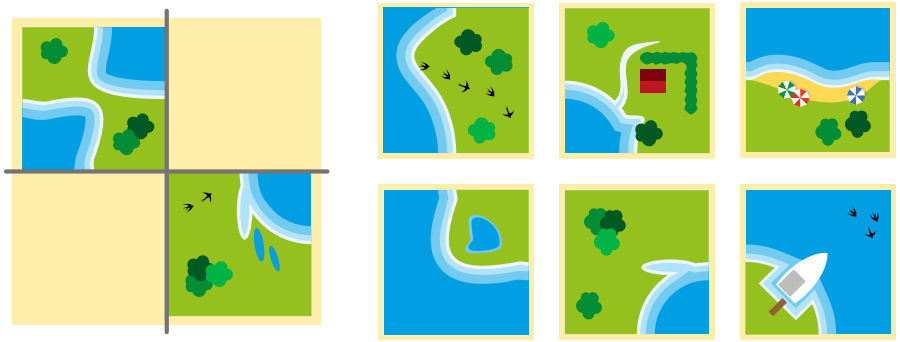
2. Terre – eau

Edu a un nouveau jeu. Il est fait de cartes avec des zones de terre et d'eau. Edu peut former des paysages avec ses cartes. Les cartes doivent *aller ensemble* : terre contre terre, eau contre eau.



Edu pose deux cartes et laisse deux trous.

Quelles cartes vont dans les trous ? Tu n'as pas le droit de tourner les cartes.



Solution

Voici la bonne réponse :

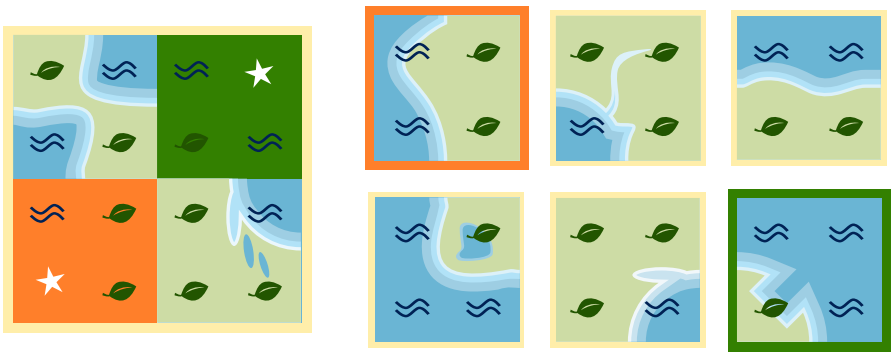
Les deux cartes vont dans les trous : il y a partout de la terre contre de la terre et de l'eau contre de l'eau. Seules ces deux cartes vont dans les trous parmi les six cartes.

Si l'on avait le droit de tourner les cartes, d'autres cartes iraient aussi dans les trous.

C'est de l'informatique !

Regardons les cartes d'Edu plus en détail. Chaque carte peut être divisée en quatre parties. Les bords extérieurs de ces parties montrent de la terre ( ) ou de l'eau ( ) .

Il n'y a donc que deux sortes de parties, car les bords extérieurs montrent soit de l'eau ( ), soit de la terre ( ).



Deux cartes ne vont ensemble que si leurs parties voisines sont les mêmes. Pour trois parties de chaque trou, nous pouvons donc indiquer quelle sorte de partie est nécessaire. La quatrième partie peut être de l'eau ou de la terre, donc nous indiquons ★.

De cette manière, nous créons un motif pour chaque trou. Les cartes devant aller dans ces trous doivent correspondre à ces motifs : pour et , la partie de la carte doit également être et ; pour ★, la partie de la carte peut être ou .

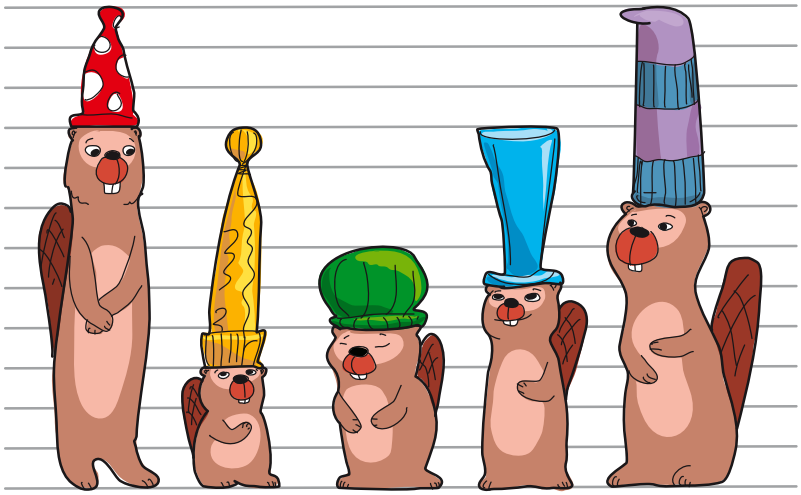
Nous avons découvert une propriété des cartes. Nous avons utilisé cette propriété pour remplacer les cartes par un arrangement des symboles et . Nous avons ainsi réduit largement la quantité d'information présente dans les images. Nous nous concentrons sur les informations qui sont nécessaires à la résolution de cet exercice. Les informaticiens et informaticiennes se réfèreraient à l'arrangement des symboles sur les images. Le fait de réduire les images aux types de partie et crée un modèle des cartes manquantes. L'*abstraction* est nécessaire à la *modélisation*, et l'abstraction réduit la quantité d'information. Les ordinateurs doivent travailler avec des modèles de la réalité. Il faut être attentif à ne pas perdre certaines propriétés importantes de la réalité lors de la création de tels modèles.

Mots clés et sites web

- Modélisation : `https://fr.wikipedia.org/wiki/Modélisation`
- Codage : `https://fr.wikipedia.org/wiki/Codage_de_l'information`
- Abstraction : `https://fr.wikipedia.org/wiki/Abstraction_(informatique)`

3. Tri par chapeau

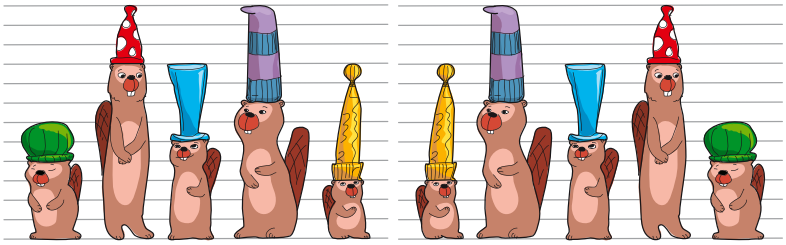
Les castors ont de nouveaux chapeaux.



Trie les chapeaux selon leur taille.

Solution

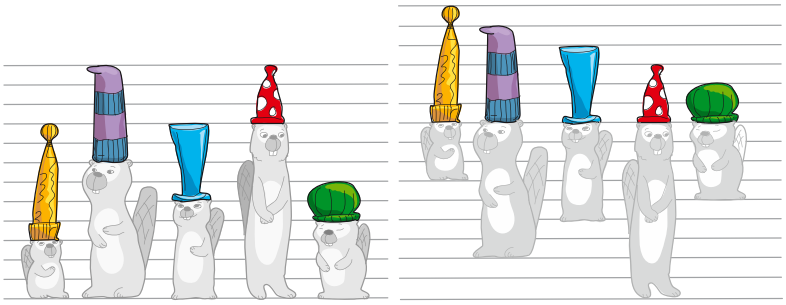
Voici les deux manières de trier les chapeaux :



Il y a deux bonnes réponses, les chapeaux peuvent devenir

- de plus en plus grands, ou
- de plus en plus petits en allant de gauche à droite.

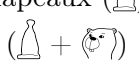
En triant les castors, on ne fait attention qu'aux chapeaux. C'est alors beaucoup plus facile de les trier par taille.

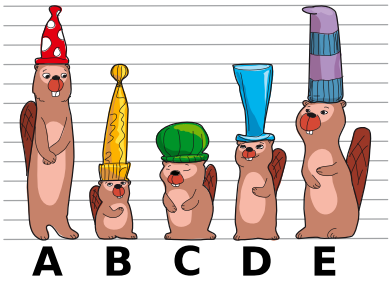


C'est de l'informatique !

Beaucoup d'objets autour de nous sont triés pour mieux pouvoir choisir parmi eux : si les outils sont triés par taille, c'est plus facile de trouver un outil précis. On peut trouver un mot facilement dans un dictionnaire parce que les mots y sont triés par ordre alphabétique.

Dans cet exercice, tu devais trier les castors d'après la taille de leur chapeau. La difficulté est que la *propriété* « taille du chapeau » n'est pas facile à reconnaître. Nous pouvons trier les castors d'après au moins trois tailles :

- Taille des castors ()
- Taille des chapeaux ()
- Taille totale ()



Le tri des castors est différent pour chacune de trois propriétés de taille.

Castor			
A	3	9	12
B	6	3	9
C	2	4	6
D	4	5	9
E	5	7	12

Pour trier, il est donc important de commencer par bien définir la propriété d'après laquelle il faudra trier. Ensuite, les valeurs de cette propriété doivent être triables : on peut trier d'après des propriétés qui sont exprimées en nombre (comme la taille, la longueur, le poids…) : on peut dire quel nombre est le plus petit entre deux nombres. On peut trier des mots car l'ordre des lettres dans l'alphabet est défini et que c'est donc clair lequel de deux mots vient avant dans le dictionnaire. De manière générale, on peut dire que l'on peut trier d'après une propriété s'il existe une relation « plus petit que » (une *relation d'ordre*) entre ses valeurs individuelles.

Les ordinateurs gèrent de grands volumes de données. Pour pouvoir y trouver des données précises, les données doivent être triées. Il existe beaucoup de méthodes rapides de tri en informatique, et quelle méthode doit être utilisée dans quel cas est un sujet bien étudié.

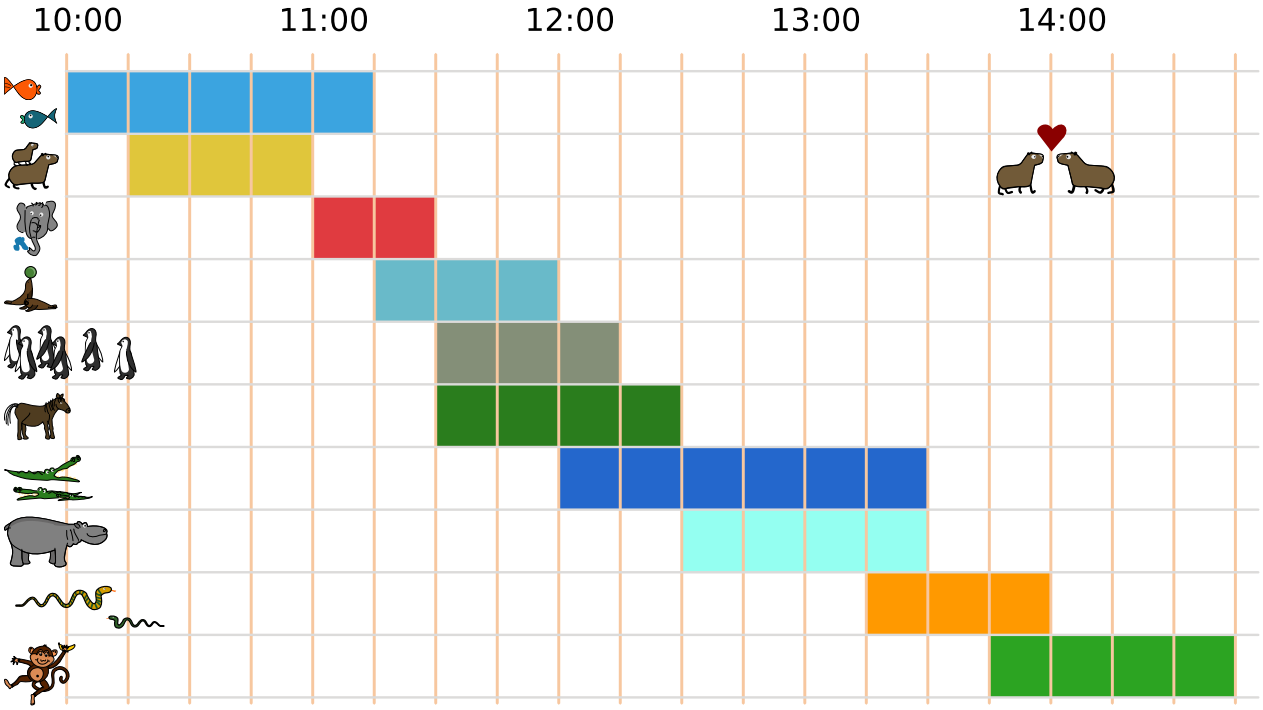
Mots clés et sites web

- Algorithme de tri : `https://fr.wikipedia.org/wiki/Algorithme_de_tri`
- Relation d'ordre : `https://fr.wikipedia.org/wiki/Relation_d'ordre`
- Algorithme de recherche : `https://fr.wikipedia.org/wiki/Algorithme_de_recherche`

4. 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.

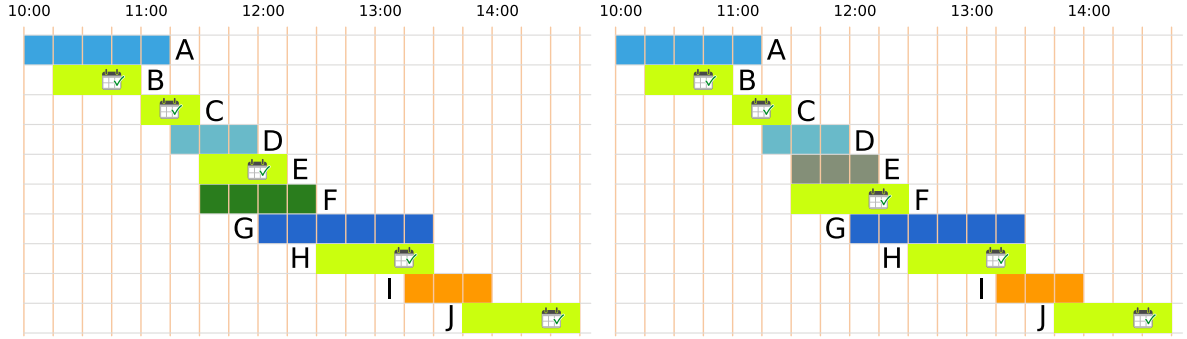


Anja assiste toujours à une présentation en entier, du début à la fin. Peux-tu l'aider ?

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 :



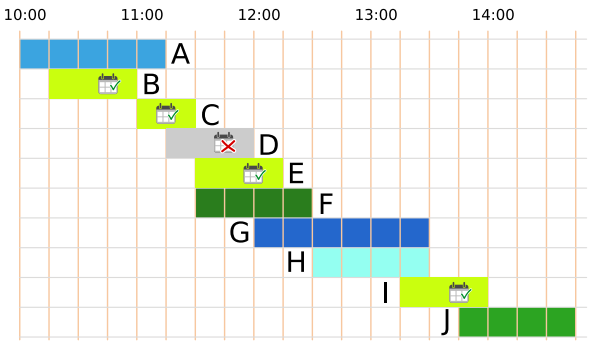
Il y a différentes façons d'arriver aux bonnes réponses.

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.

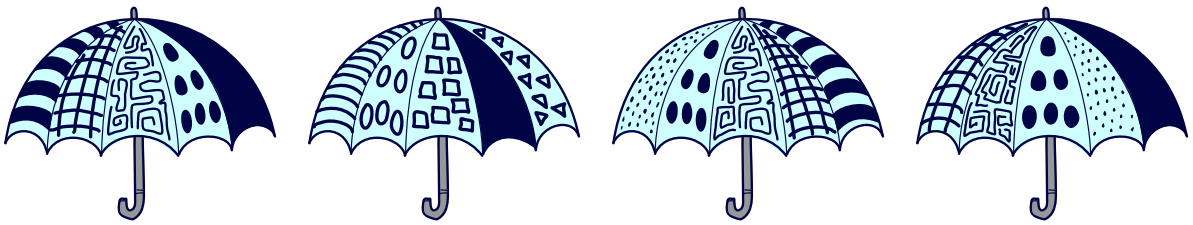
Mots clés et sites web

- 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`

5. Parapluie

Voici le parapluie d'Anna :

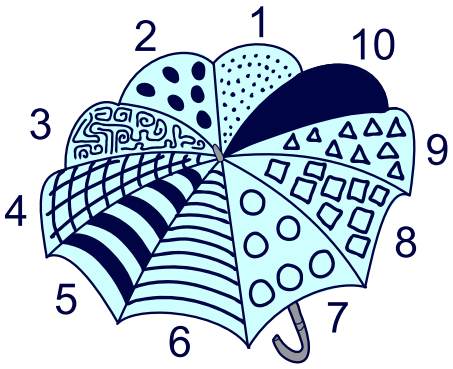
Une des quatre images montre le parapluie d'Anna. Laquelle ?



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 :

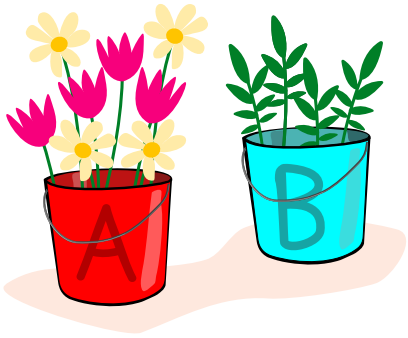
- plus le mot recherché est long, moins il y a de correspondances possible dans le texte et plus la chance de trouver l'endroit recherché dans le texte est élevée,
- plus le mot recherché est court, plus il y a de correspondances possible dans le texte et moins la recherche est exacte.

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`

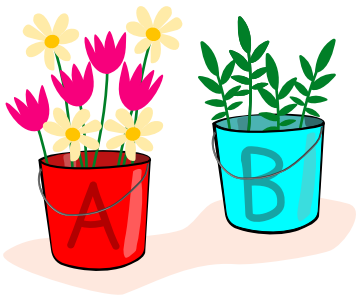
6. Fleuriste

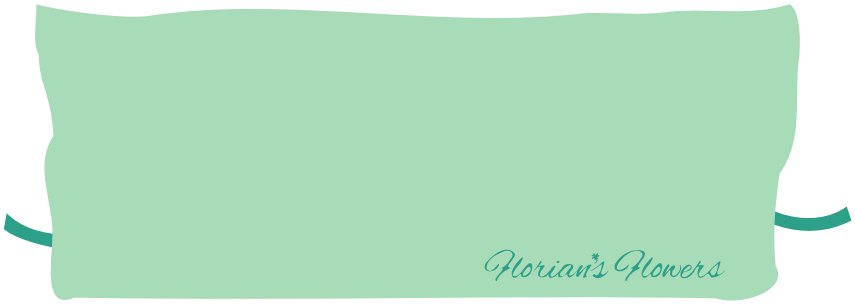


Florian vend des bouquets de fleurs. Il assemble chaque bouquet d'après ces instructions :

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 !

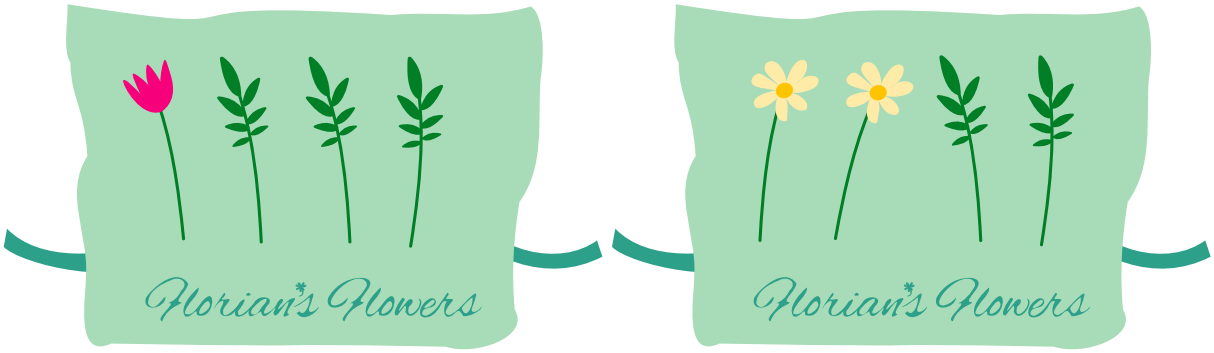
Aide Florian : suis les instructions et choisis des fleurs et des branches pour un bouquet.



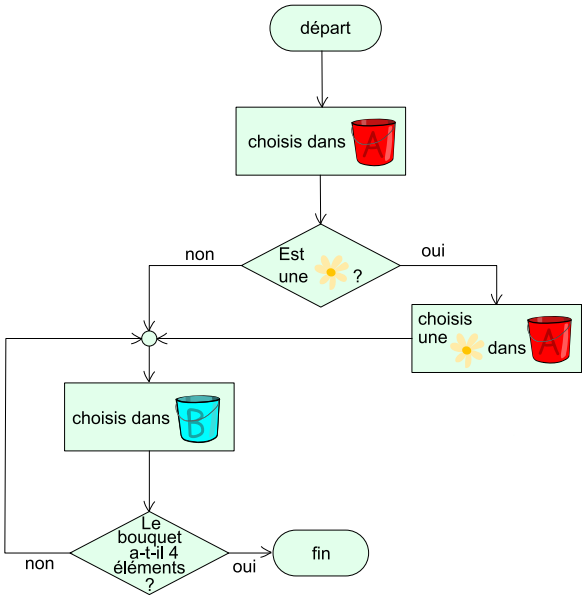


Solution

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.

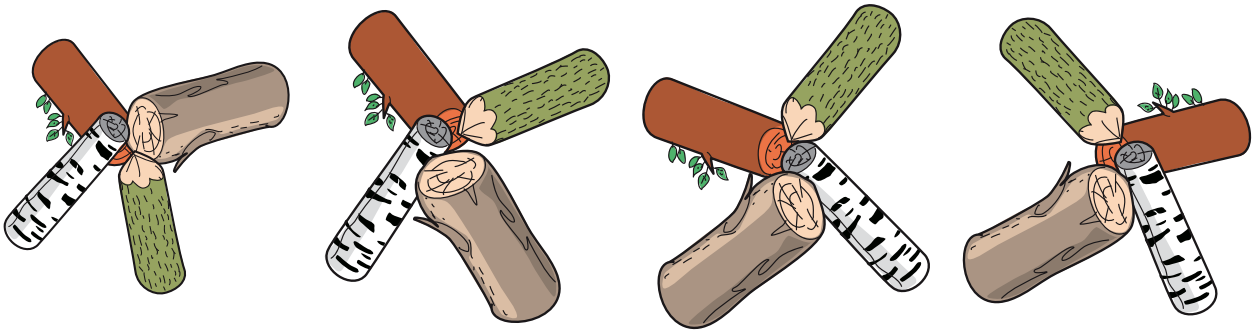
Mots clés et sites web

- 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`

7. Photo

Le castor vient de prendre une photo.

Laquelle des quatre photos a-t-il prise ?



A) B) C) D)

Solution

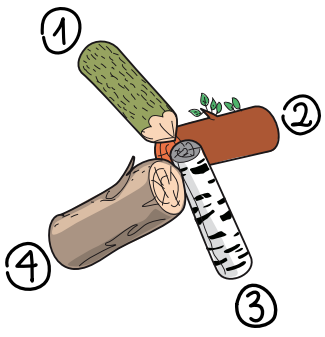
La bonne réponse est D). 

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 :

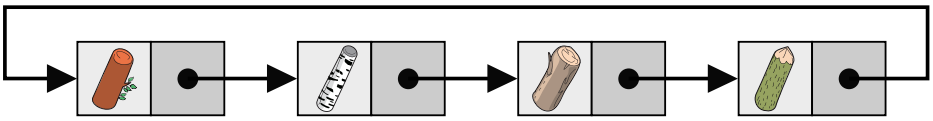
- Photo A : 1 – 3 – 2 – 4
- Photo B : 1 – 4 – 3 – 2
- Photo C : 1 – 3 – 4 – 2
- Photo D : 1 – 2 – 3 – 4

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

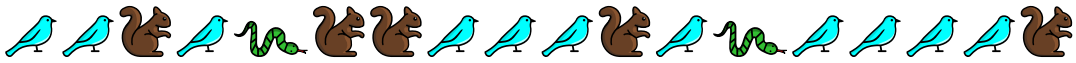
- Liste chaînée : `https://fr.wikipedia.org/wiki/Liste_chaînée`

8. 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

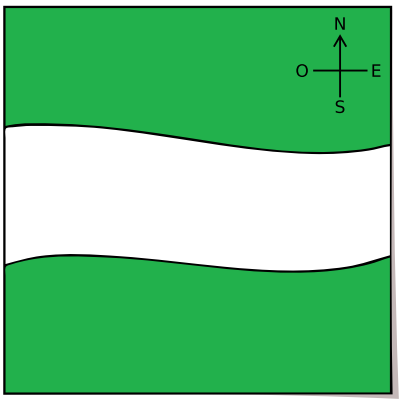
propriétés de ce programme. Une telle analyse de programmes (informatiques) fait partie des tâches des informaticiens et informaticiennes.

Mots clés et sites web

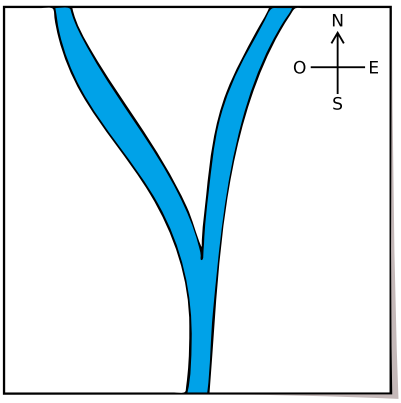
- Variable : `https://fr.wikipedia.org/wiki/Variable_(informatique)`
- État d'un programme : `https://fr.wikipedia.org/wiki/État_(informatique)#Processus`

9. La maison de Karla

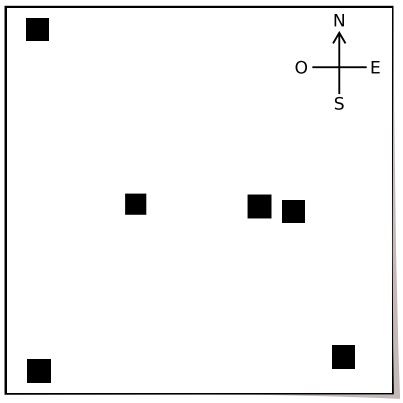
Karla a trois cartes qui montrent exactement la même région. Une carte montre les forêts ; une autre, les rivières, et la troisième, les maisons dans cette région. La maison de rêve de Karla se trouve dans la forêt et près d'une rivière.



Carte des forêts



Carte des rivières

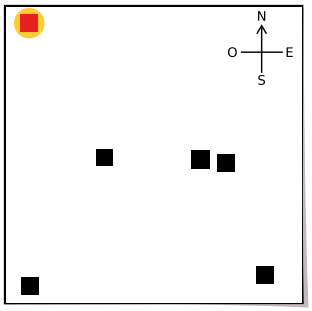


Carte des maisons

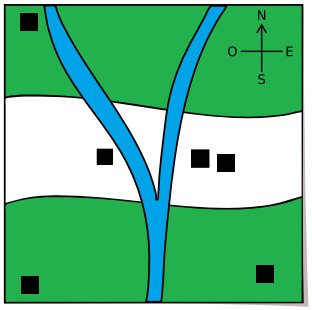
Quelle est la maison de rêve de Karla ?

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.

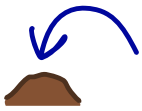
Les programmes graphiques utilisent aussi plusieurs *niveaux* avec des informations graphiques différentes (appelés *calques*). Une question importante est toujours quel niveau est le plus haut et est donc représenté au premier plan. Ici, ce sont par exemple les maisons qui doivent être au premier plan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

Mots clés et sites web

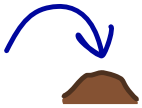
- SIG : `https://fr.wikipedia.org/wiki/Système_d'information_géographique`
- Calque : `https://fr.wikipedia.org/wiki/Calque_(infographie)`

10. 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 :

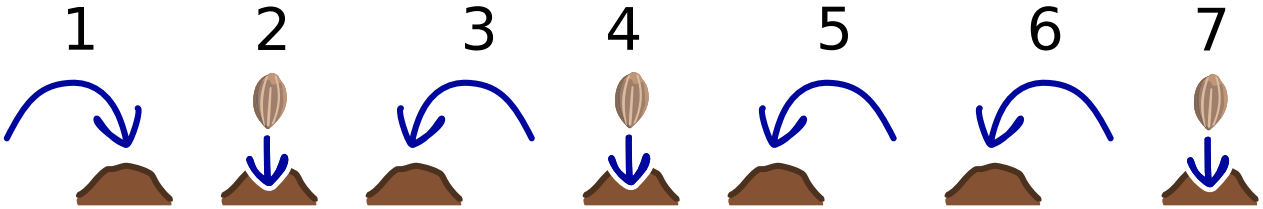
Il a passé sur quatre collines. Nous ne savons pas sur quelle colline il a commencé.

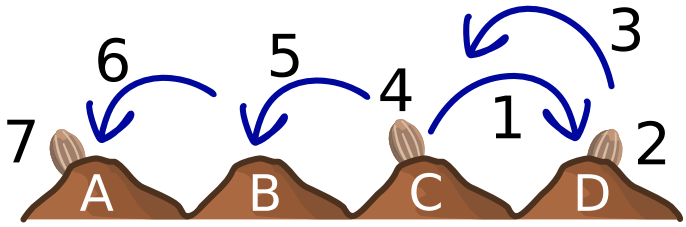
Sur quelles collines le robot a-t-il planté des graines de carottes ?

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`

11. 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`

12. 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.

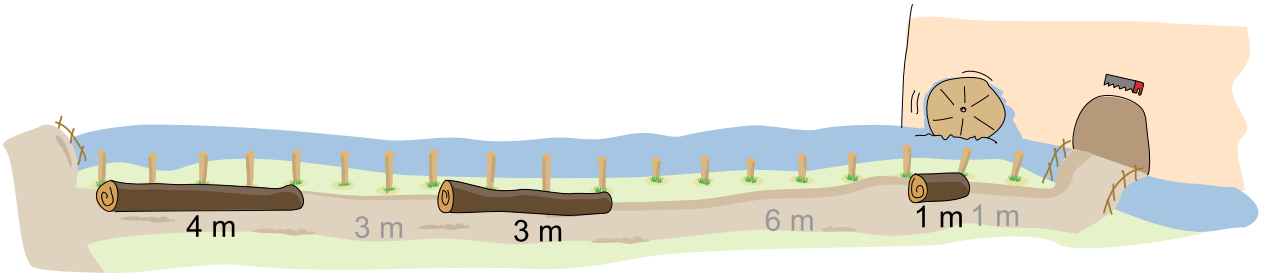
Mots clés et sites web

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

13. Les troncs de Timea

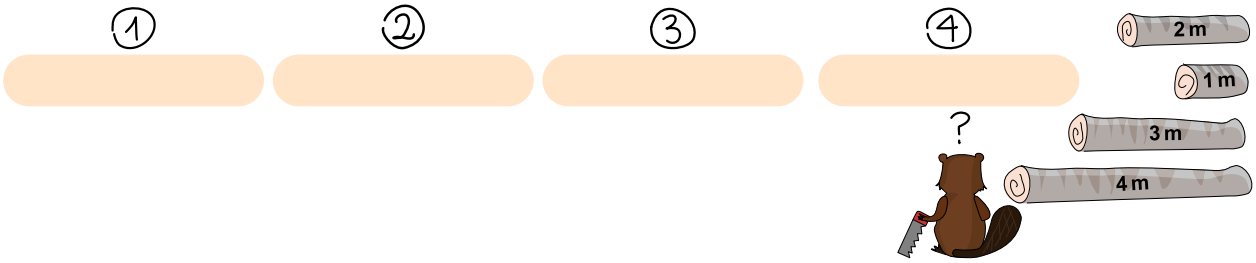
Timea la castor coupe des troncs d'arbre de différentes longueurs, puis les vend. Dès qu'elle a coupé un tronc, elle le pose sur le chemin long de 18 mètres. Timea suit pour cela la règle suivante : en commençant à gauche, elle place le tronc dans le premier espace vide assez grand pour l'y mettre.

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



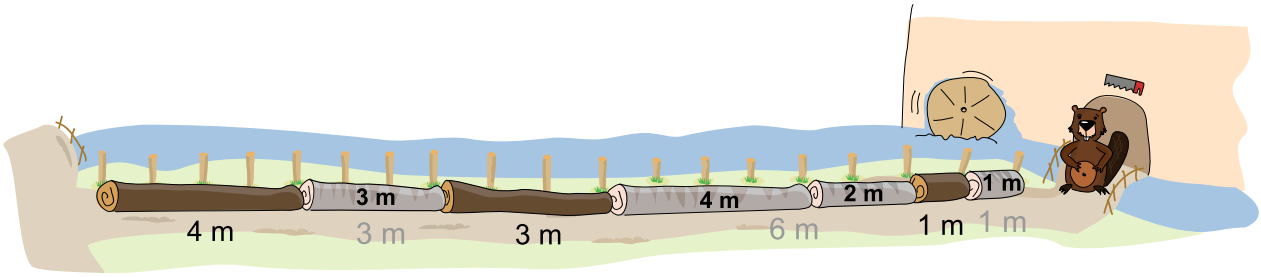
Timea veut maintenant couper quatre troncs longs de 1, 2, 3, et 4 mètres.

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

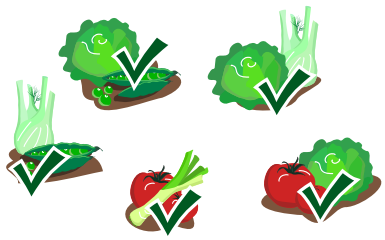
stratégie peut mener à de mauvais résultats : les troncs doivent être placés dans un certain ordre pour pouvoir remplir tous les espaces vides sur le chemin.

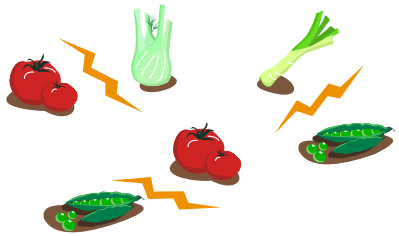
Mots clés et sites web

- 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)`

14. Jardin potager

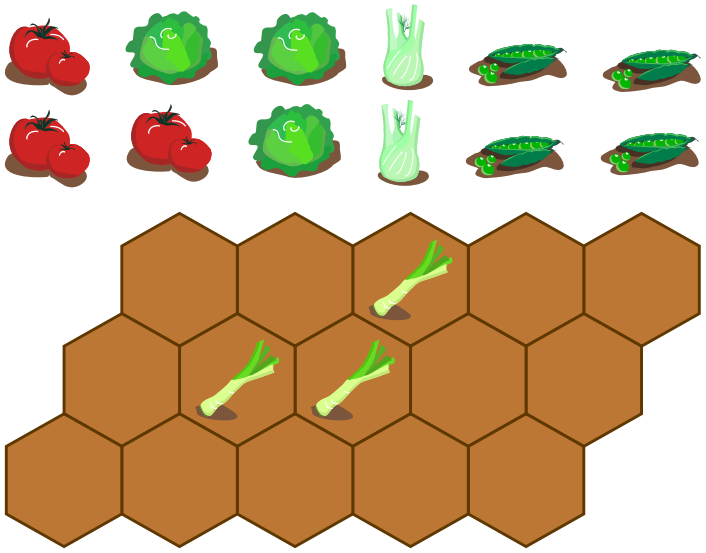
Lisa prépare un jardin potager. Elle veut y cultiver cinq sortes de légumes différentes. Certaines sortes de légumes se supportent bien et sont compatibles ✓, d'autres sont incompatibles ⚡ :





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.

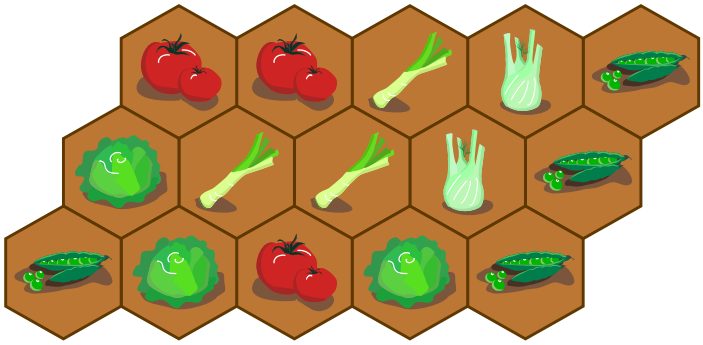


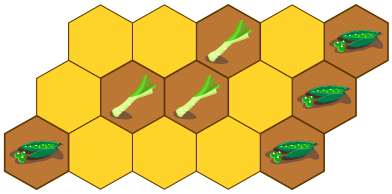
Lisa plante en suivant la règle suivante : les légumes incompatibles ne peuvent pas être plantés dans des domaines qui se touchent.

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

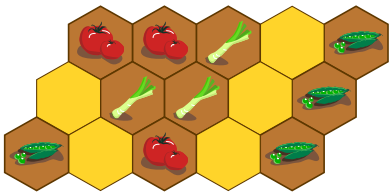
Solution

Voici la bonne réponse :

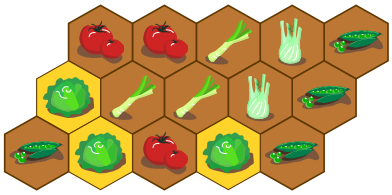




Comme les petits pois ne sont pas compatibles avec les poireaux, Lisa ne plante pas de petits pois dans les domaines jaunes. Il ne reste que les autres domaines pour le petits pois.



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 !

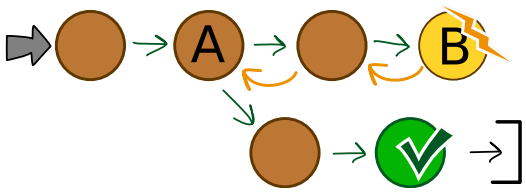
Pour planter des légumes afin d'avoir une récolte aussi grande que possible, il faut respecter beaucoup de *conditions* : chaque sorte a des besoins de place, de nutriments et de lumière différents, par exemple. Dans cet exercice du Castor, nous ne considérons qu'une sorte de condition : la compatiblité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

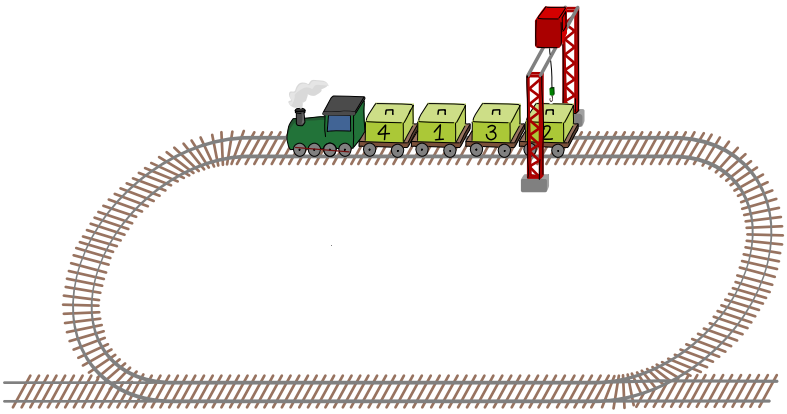
Ce retour en arrière est appelé *retour sur trace* en informatique (*backtracking* en anglais).

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`

15. Train de marchandises

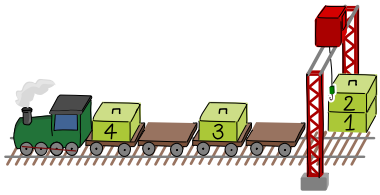
Un train tire des wagons chargés de caisses numérotées. La grue est à une position fixe et décharge les caisses. Pour décharger une caisse, la caisse doit être positionnée directement sous la grue.



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.

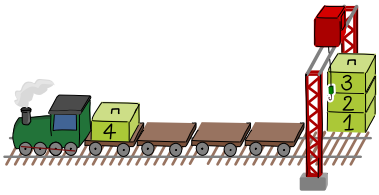
Voilà comment il décharge les caisses 1, 2, 3 et 4 :

Tour 1 :



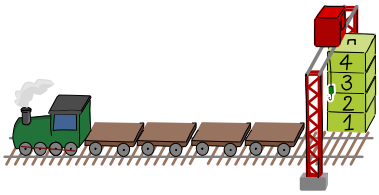
Il saute la caisse 4, décharge la caisse 1, saute la caisse 3 et décharge la caisse 2.

Tour 2 :



Il saute la caisse 4 et décharge la caisse 3.

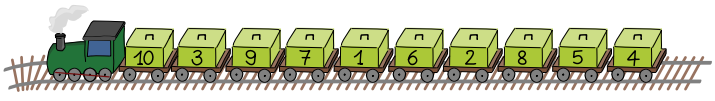
Tour 3 :



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 ?



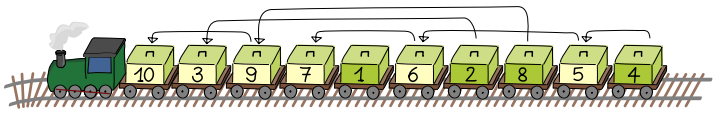
A) 1 tour
B) 2 tours
C) 3 tours
D) 4 tours
E) 5 tours
F) 6 tours
G) 7 tours
H) 8 tours
I) 9 tours
J) 10 tours

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.

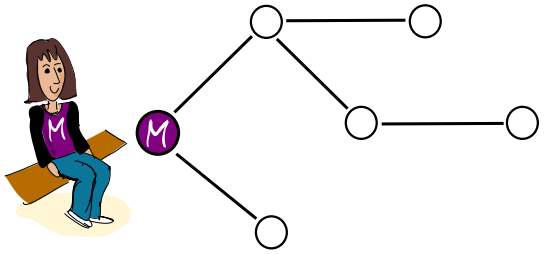
Mots clés et sites web

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

16. Le village de Martina

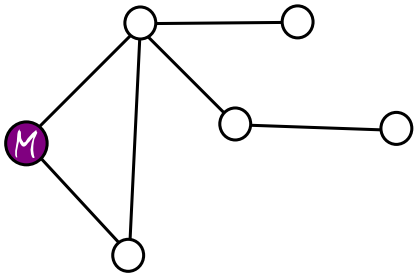
Il y a six maisons dans le village de Martina. Il y a aussi des chemins pour aller d'une maison à la suivante. Martina met le même temps à parcourir chacun de ces chemins.

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.

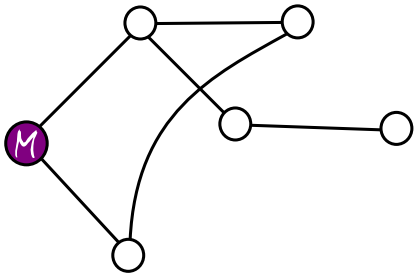


Il existe bien sûr aussi une vraie carte du village avec tous les chemins.

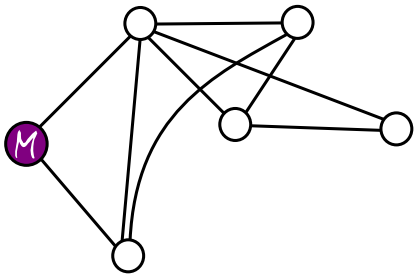
Lequel de ces dessins ne peut-il pas *être la vraie carte ?*



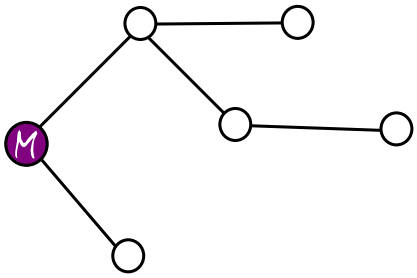
A)



B)



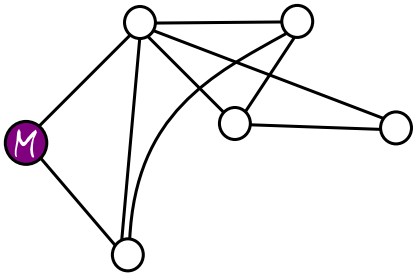
C)



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.

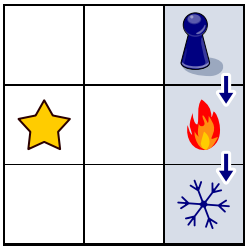
Mots clés et sites web

- 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`

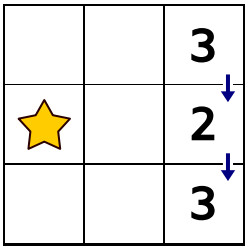
17. Chaud ou froid

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.



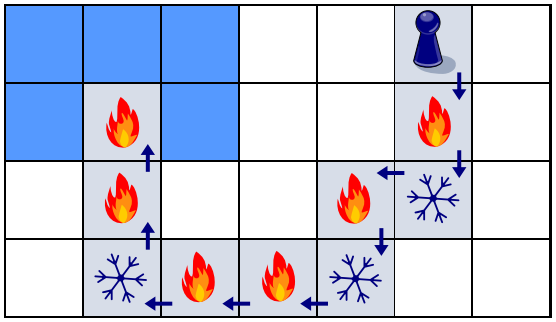
Pour commencer, Nina et Daniel prennent un petit plateau. Nina cache le trésor sur la case avec l'étoile. Daniel commence en haut à droite et fait deux pas en suivant les flèches. Après chaque pas, Nina lui dit s'il est plus près ou plus loin du trésor qu'avant.



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.

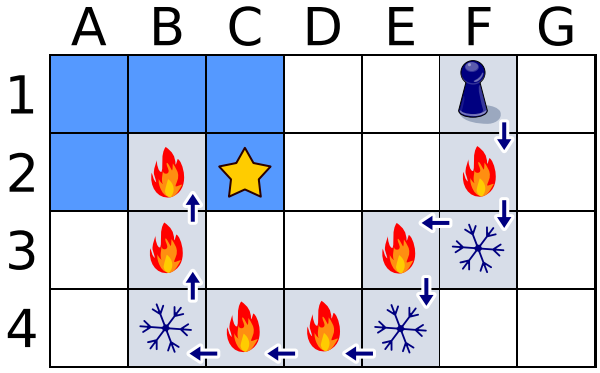
Ils prennent maintenant une plus grande planche de jeu. Nina cache le trésor sur une des cases bleues. L'image montre les pas de Daniel et ce que Nina dit après chaque pas.

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.

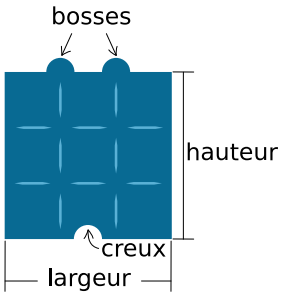
Mots clés et sites web

- 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`

18. Briques castor

Les briques castor ont quatre propriétés qui les différencient :

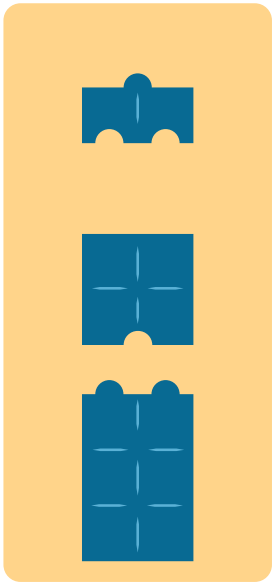
1. Largeur : étroite, moyenne, large
2. Hauteur : petite, moyenne, grande
3. Nombre de bosses en haut : zéro, une, deux
4. Nombre de creux en bas : zéro, un, deux



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.



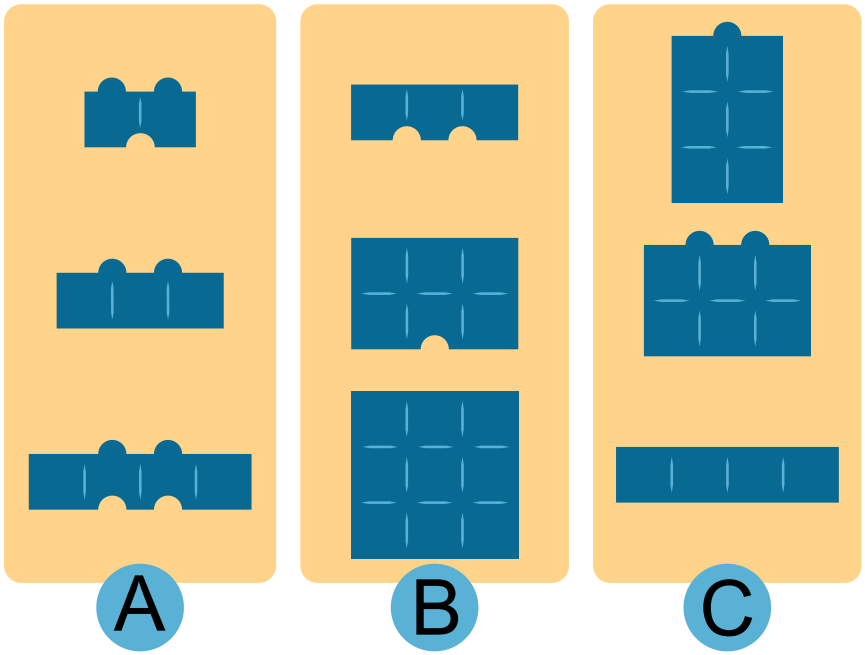
Ces trois briques ont :

- 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
...	...	...	...	...

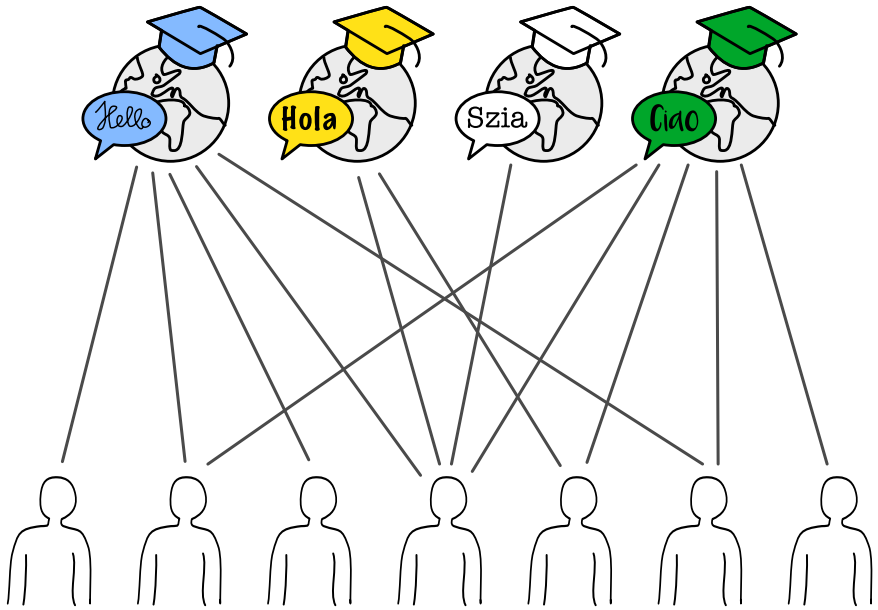
L'élaboration de bases de données est une tâche habituelle pour les informaticiennes et informaticiens. C'est un travail minutieux et il faut bien réfléchir à quels attributs des objets sont importants pour le traitement des données par un programme informatique. Ce n'est pas facile de faire des changements 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)`

19. Répartition des tâches

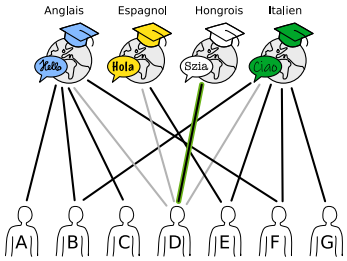
Une école de langue organise quatre cours d'été. Sur l'image ci-dessous, les lignes montrent quel enseignant de l'école est capable de donner quel cours.



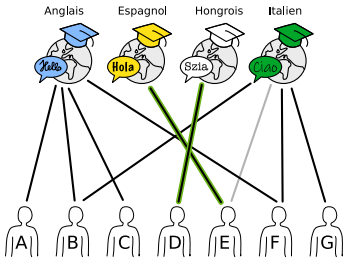
Chaque enseignant ne peut donner qu'un seul cours. Il y a quand même plusieurs possibilités d'assigner un enseignant capable à chaque cours.

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

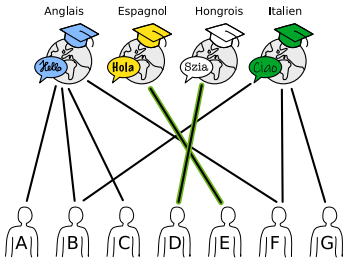
Solution



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.



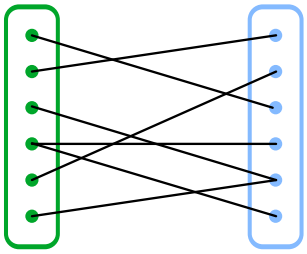
Pour les deux cours restants, l'anglais et l'italien, on a le choix. B et F ne peuvent être assignés qu'à un seul cours chacun, même s'ils sont capables d'enseigner les deux.

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

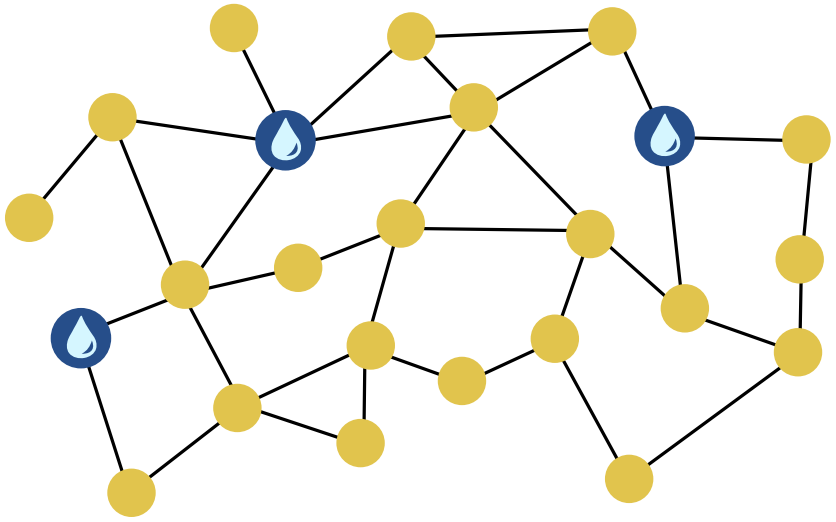
- Graphe biparti : `https://fr.wikipedia.org/wiki/Graphe_biparti`
- Problème d'affectation : `https://fr.wikipedia.org/wiki/Problème_d'affectation`
- Programme pour résoudre l'exercice :
  `https://www.coding4you.at/dachu_2023/ir02/index.html`
- Lemme des mariages : `https://images.math.cnrs.fr/Le-lemme-des-Mariages.html`

20. 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.

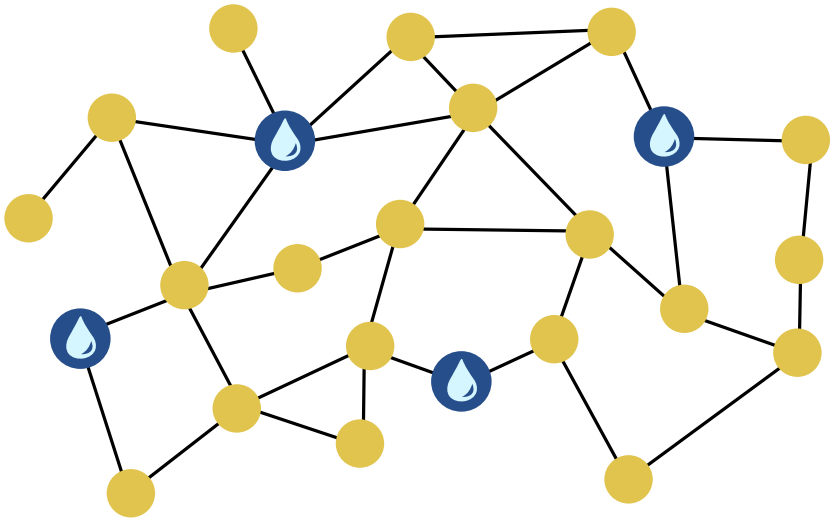
Voici un plan de la ville. Les lignes sont les tronçons de rue et les points les coins de rue. Il y a déjà des fontaines à trois coins de rue.



Installe une fontaine supplémentaire pour satisfaire la maire.

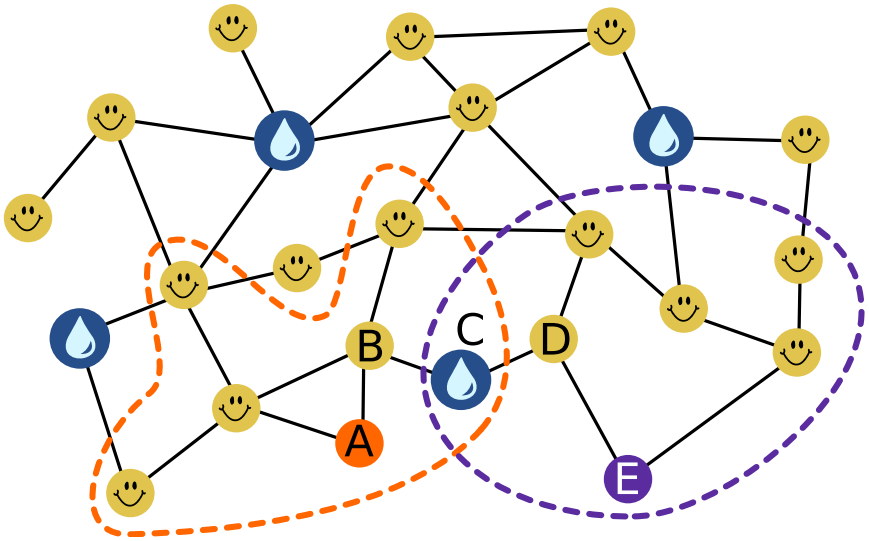
Solution

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`

21. Chantier castor

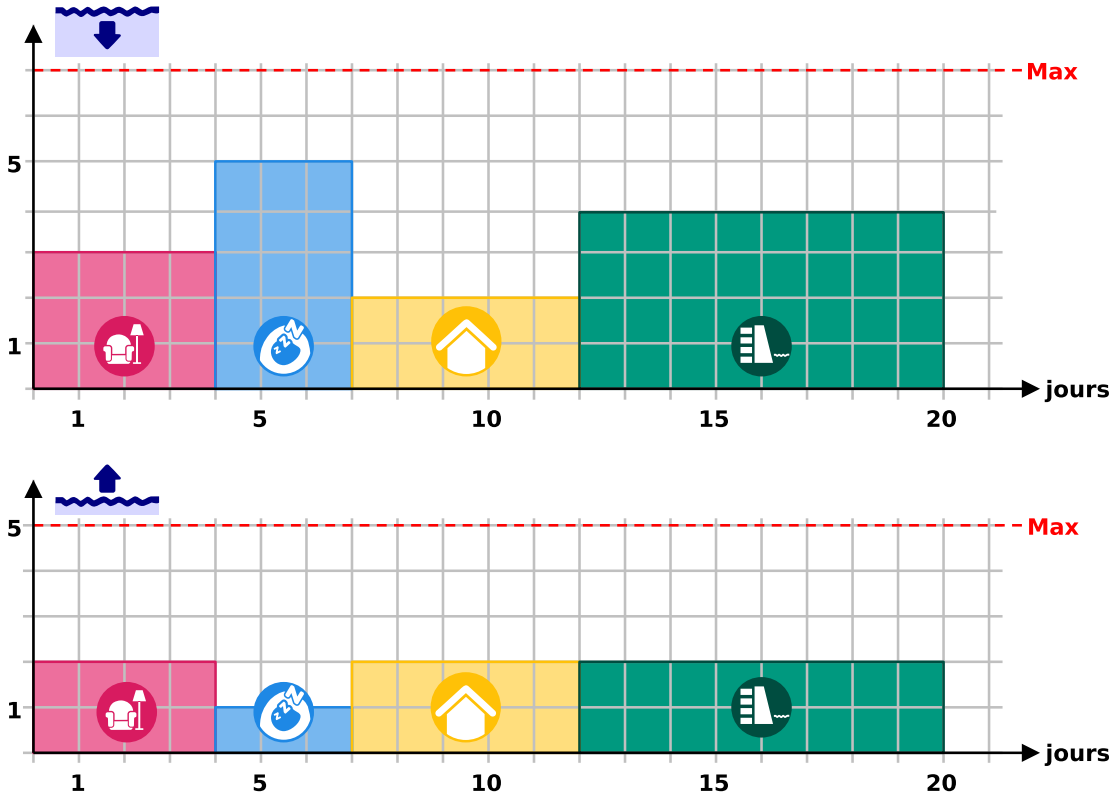
Une hutte de castor est composée de quatre parts qui sont toutes en partie sous l'eau et en partie au-dessus de l'eau.

Lors de la construction d'une hutte, chaque ouvrier travaille soit uniquement sous l'eau, soit uniquement au-dessus de l'eau. Chaque part est construite simultanément sous et au-dessus de l'eau. Le tableau montre de combien de temps et d'ouvriers sous et au-dessus de l'eau la société de construction « Castor SA » a besoin pour chaque part de hutte.

Part	**Salon**	**Chambre**	**Toit**	**Barrage**
Durée	4 jours	3 jours	5 jours	8 jours
	3	5	2	4
	2	1	2	2

Le toit ne peut être construit que lorsque la chambre est finie. L'ordre est sans importance pour toutes les autres parts.

Seuls 7 ouvriers sous l'eau et 5 ouvriers au-dessus de l'eau sont disponibles pour la construction d'une nouvelle hutte. Ils peuvent également contruire plusieurs parts en même temps. Voici un plan de travail permettant de construire la hutte en 20 jours :

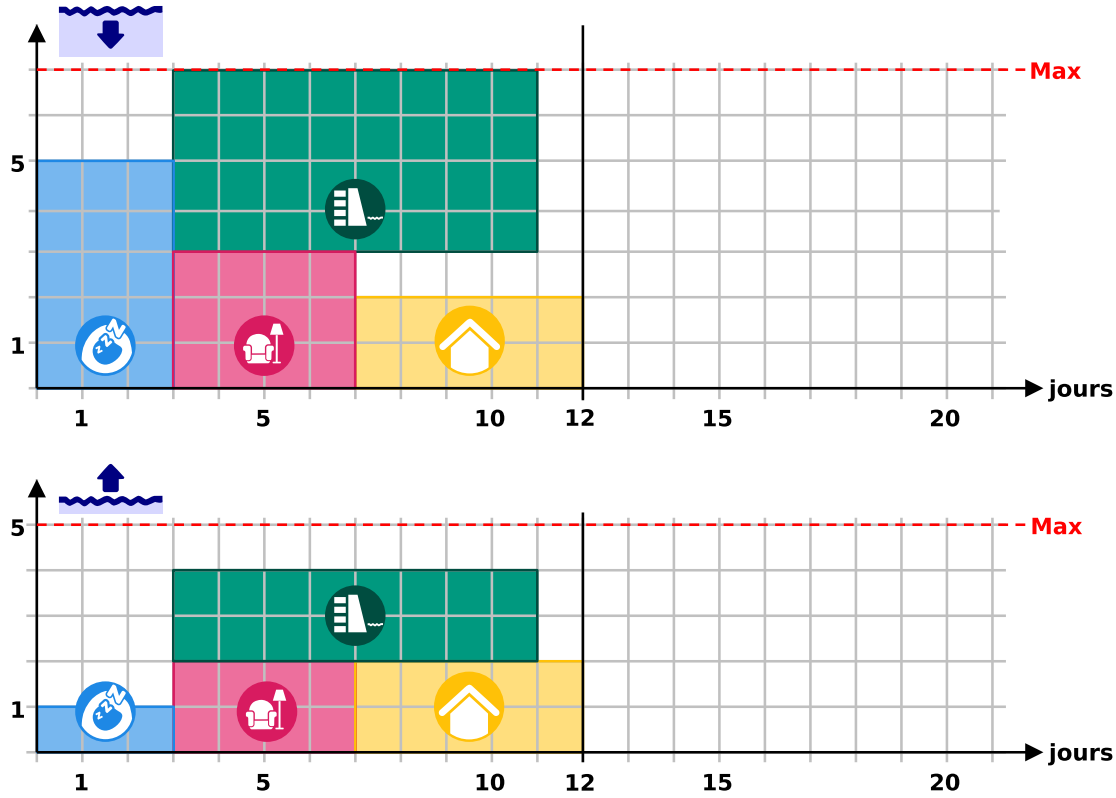


Développe un plan de travail permettant de finir la hutte en le moins de jours possible. Combien de jours faut-il ?

Solution

La bonne réponse est 12 jours.

Voici un plan permettant de construire la hutte en 12 jours :



Un tel plan de durée minimale peut être développé en deux étapes :

1. D'abord, la chambre doit être préparée avant le toit. Comme la chambre a besoin de cinq ouvriers sous l'eau, le barrage de 3 et le salon de 4, la chambre ne peut pas être construite en même temps que le salon ou le barrage avec 7 ouvriers sous l'eau en tout. La chambre doit donc être construite en premier et les autres parts ensuite.
2. Le barrage et le salon peuvent être construits en même temps après la chambre, ou l'une des deux parts en même que le toit. Il n'est pas possible de construire les trois parts en même temps, parce qu'il faudrait $3 + 4 + 2 = 9$ ouvriers sous l'eau et qu'il n'y en a que sept à disposition. La durée de construction la plus courte peut être atteinte en construisant les deux parties construites le plus rapidement (le toit et le salon) l'une après l'autre et le barrage en même temps.

C'est de l'informatique !

C'est une tâche difficile de planifier le déroulement rapide d'un projet en respectant certaines conditions. Les différentes tâches faisant partie d'un projet dépendent souvent les unes des autres ; par exemple, certaines tâches ne pourront être effectuées qu'après la fin d'autre tâches – comme ici

pour la construction de la chambre et du toit. De plus, chaque tâche nécessite certaines ressources, comme des travailleurs, du temps et des machines. Il est plus facile de réaliser un plan de projet si ce plan peut bien être représenté. Les diagrammes montrés dans cet exercice du Castor sont une sorte de diagramme de Gantt, diagrammes qui ont été développés par Henry Gantt (1861−1919) entre 1910 et 1915 ; des représentations similaires ont été utilisées indépendemment de Gantt à la même période en Allemagne. Ces diagrammes montrent l'utilisation des ressources (ici, des ouvriers) au cours du temps.

On peut développer le plan optimal pour la hutte castor de tête en essayant toutes les possibilités. Cela durerait trop longtemps et serait trop complexe pour de plus grands projets. Dans ces cas-là, des programmes informatiques sont utiles, et le développement d'horaires et de plans (*scheduling* en anglais) est un thème important en informatique. Comme souvent pour les problèmes complexes, des méthodes permettant de développer de bons horaires, mais pas forcément le meilleur horaire possible, ont été développées. Le *scheduling* est aussi utilisé dans la gestion des ordinateurs eux-même et est appelé *ordonnancement*, certaines tâches étant en compétition pour certaines ressources (mémoire, capacité de calcul, accès à des appareils externes comme des imprimantes, réseaux ou disques durs).

Mots clés et sites web

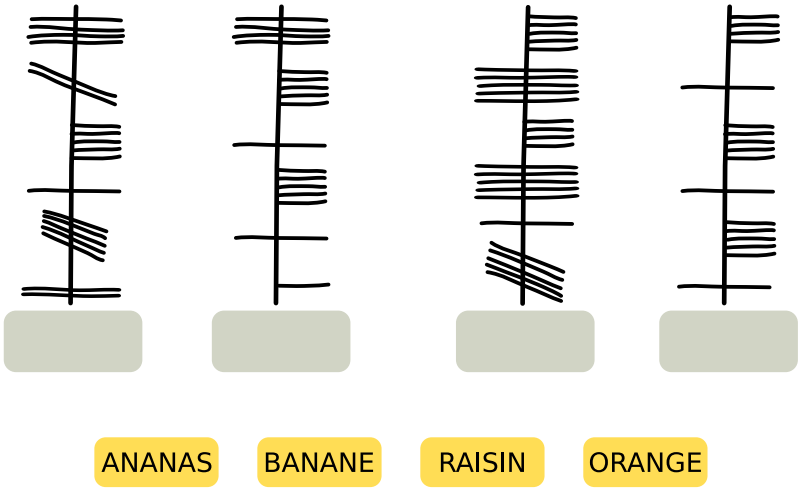
- Ordonnancement : `https://fr.wikipedia.org/wiki/Ordonnancement_de_travaux_informatiques`
- Diagramme de Gantt : `https://fr.wikipedia.org/wiki/Diagramme_de_Gantt`
- Logiciel de gestion de projets : `https://fr.wikipedia.org/wiki/Logiciel_de_gestion_de_projets`

22. 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.

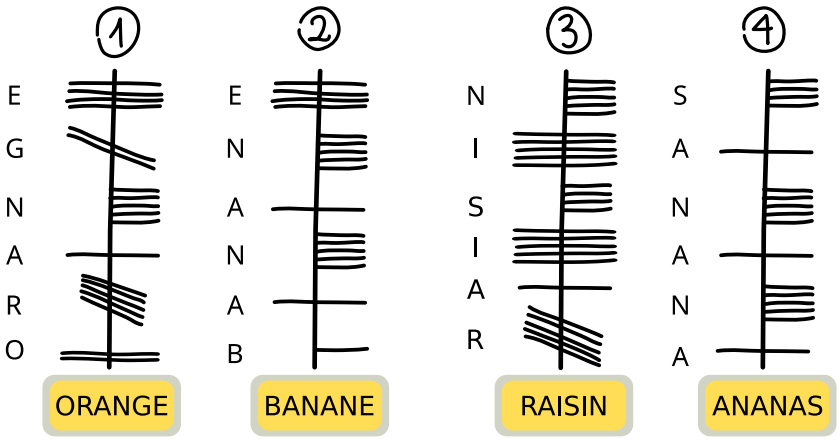
Quel mot correspond à quel code en Ogham ?



ANANAS BANANE RAISIN ORANGE

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.

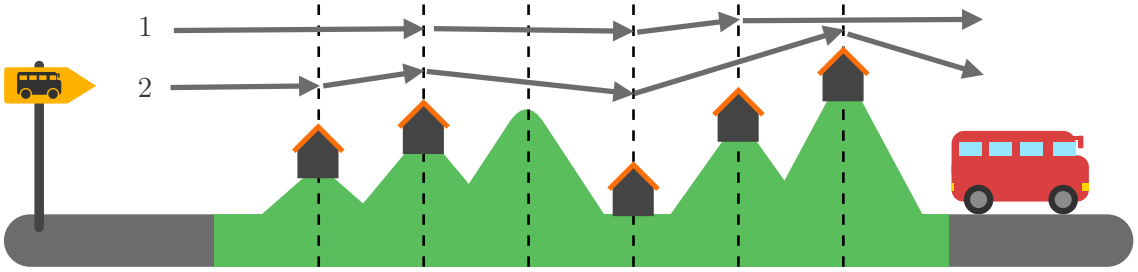
En informatique aussi, il faut déchiffrer des textes et des symboles – après qu'ils ont été encryptés pour le transfert de données sécurisé. Pour cela, des méthodes très différentes de celles utilisées pour coder des mots dans d'autres écritures sont appliquées. De tels chiffres simples sont trop faciles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

23. 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

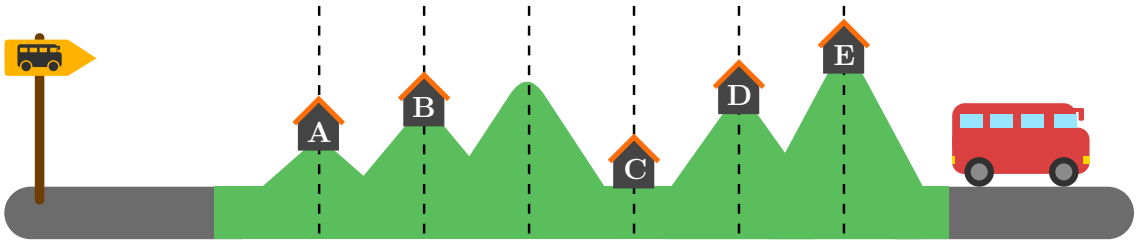
E) 6 randonnées

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

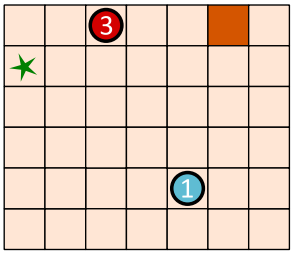
complet. Comme les solutions de sous-problèmes peuvent souvent être utilisées pour résoudre plusieurs problèmes plus grands, elles sont enregistrées pour éviter de devoir répéter les mêmes calculs. La programmation dynamique peut aussi être utile pour compter le nombre de solutions possibles à un problème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

24. Robots tamponneurs

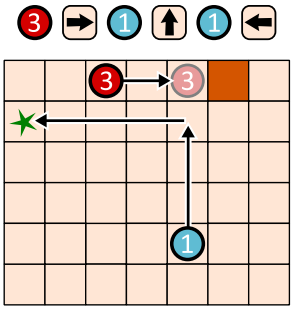
Les robots tamponneurs sont des robots très simples. Ils roulent sur un plateau de jeu divisé en cases.



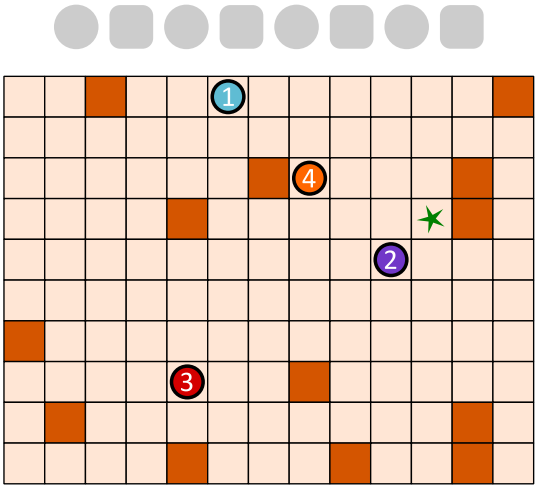
Pour les diriger, on commence par sélectionner un robot tamponneur. On l'envoie ensuite dans une certaine direction à l'aide d'une flèche-commande : en haut, en bas, à gauche ou à droite. Le robot tamponneurs roule ensuite tout droit dans la direction de la commande jusqu'à ce qu'il rencontre un obstacle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



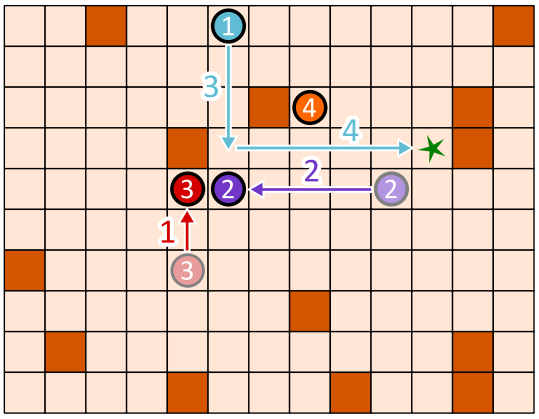
Crée une suite de commandes avec quatre flèches permettant au robot tamponneur d'atteindre le but.



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`

25. Les courses d'Emma

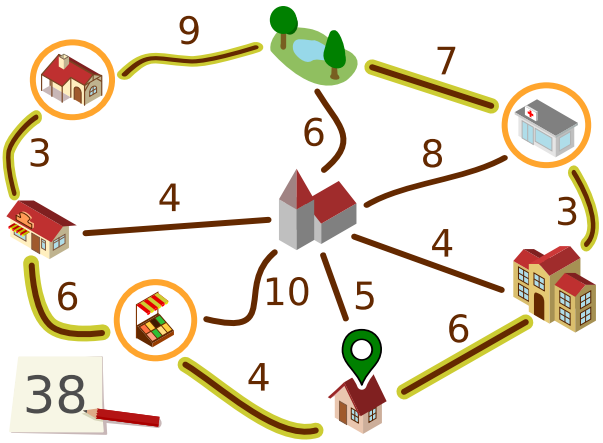
Emma est à la maison . Elle doit faire trois courses et revenir à la maison :

- 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.

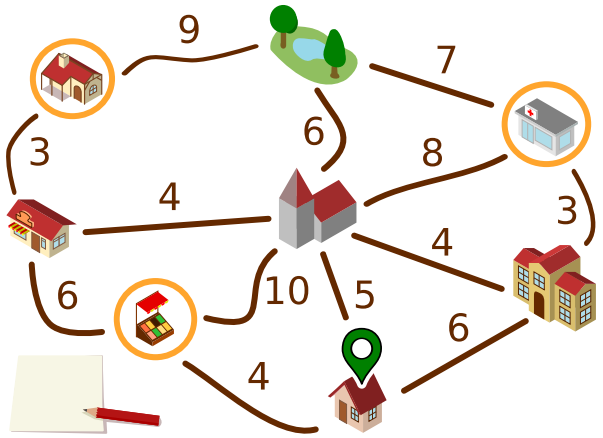
Emma a noté sur un plan de combien de minutes elle a besoin pour parcourir les chemins entre différents endroits de sa ville. Elle a aussi noté quels chemins elle prend pour faire ses courses.

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



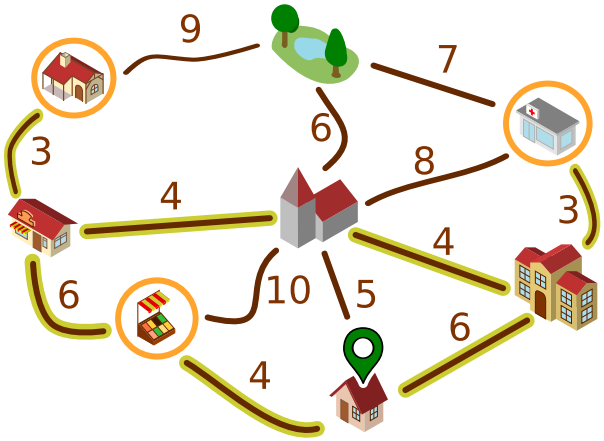
Emma se demande si elle pourrait être plus rapide. Peut-être en faisant l'aller-retour sur certains chemins ?

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

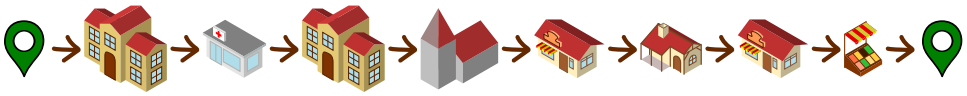


Solution

Voici la bonne réponse :

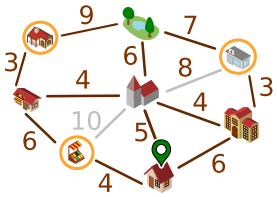


Emma peut faire le trajet suivant le long des chemins sélectionnés (ou dans la direction opposée) :

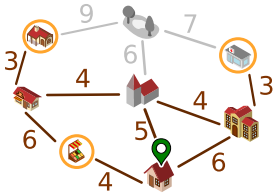


Pour ce trajet, elle a besoin de $6 + 3 + 3 + 4 + 4 + 3 + 3 + 6 + 4 = 36$ minutes.

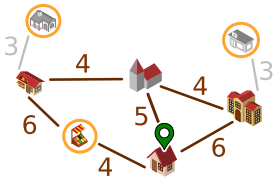
Nous voulons maintenant démontrer qu'il ne peut pas y avoir de trajet plus court. Pour cela, nous utilisons une version simplifiée du plan.



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

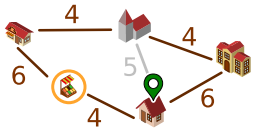


Nous pouvons également ignorer le parc. Emma ne doit pas aller au parc, et il existe un chemin plus court pour chaque chemin passant par le parc.



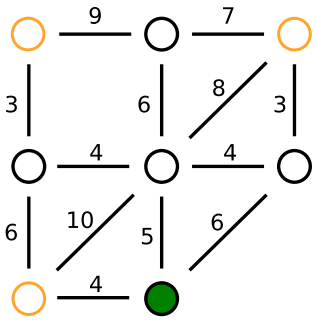
Emma doit aller à la pharmacie et au kiosque. Elle ne peut y aller que depuis la boulangerie et l'école, respectivement. Elle doit faire l'aller-retour entre ces endroits, ce qui dure $3 + 3 = 6$ minutes pour chacun, donc 12 minutes en tout. Nous en prenons note et simplifions la plan en enlevant les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

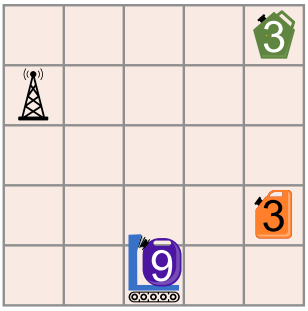
- Théorie des graphes : `https://fr.wikipedia.org/wiki/Théorie_des_graphes`
- Graphe : `https://fr.wikipedia.org/wiki/Graphe_(mathématiques_discrètes)`
- Problème du plus court chemin : `https://fr.wikipedia.org/wiki/Problème_de_plus_court_chemin`

26. 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 !

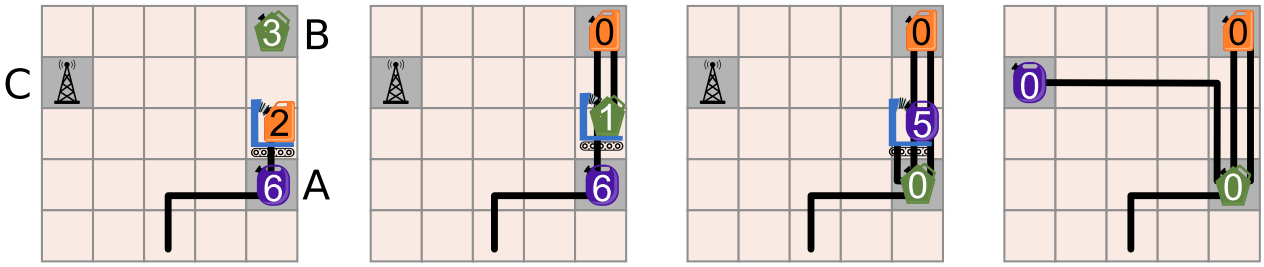
Voici la mission de Zérobot : il doit aller à la station de base  en vidant tous les réservoirs (niveau 0).

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`

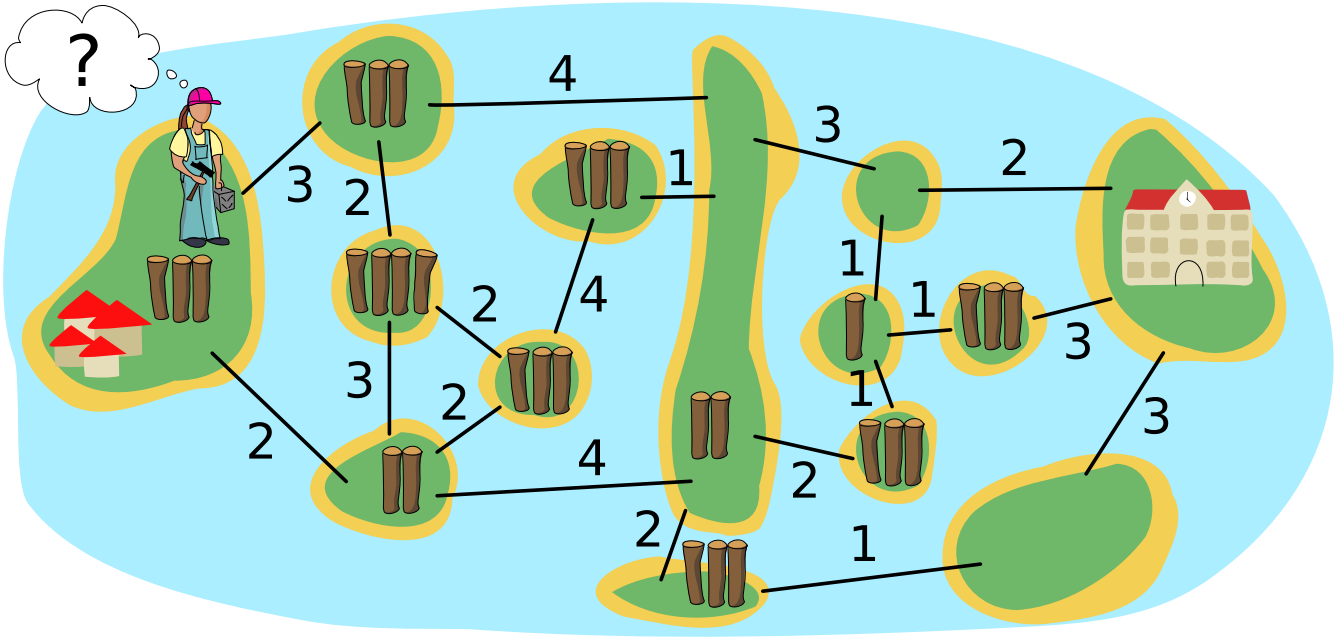
27. Raccordement

Des enfants ont emménagé sur l'île tout à gauche. Bianca doit construire des ponts permettant aux enfants d'aller à l'école sur l'île tout à droite.

La carte des îles montre combien il y a de troncs sur chaque île. Bianca peut prendre ces troncs pour construire des ponts sur les lignes. Le chiffre au-dessus d'une ligne indique combien de troncs sont nécessaires pour y contruire un pont. Dès qu'un pont entre deux îles est construit, Bianca peut le traverser en prenant avec les troncs qu'il lui reste. Elle ne peut bien sûr utiliser chaque tronc que pour un seul pont.

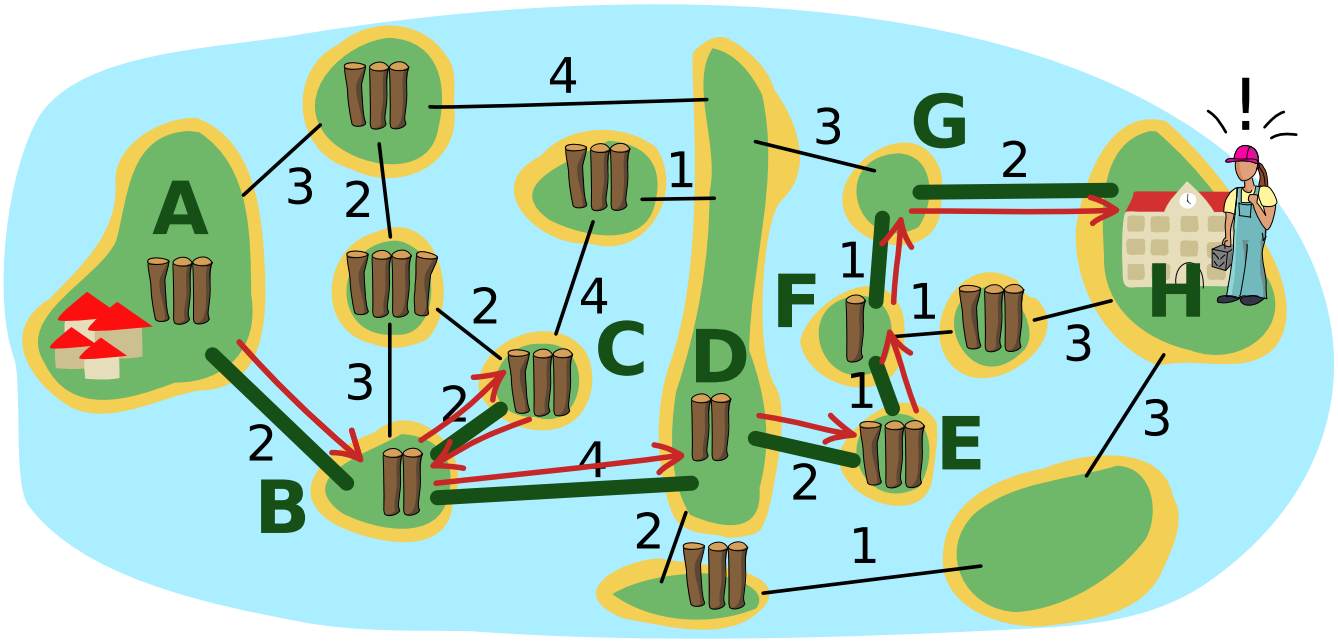
Bianca commence sur l'île de gauche. Son but est d'utiliser le moins de troncs possible.

Sur quelles lignes Bianca doit-elle construire des ponts pour atteindre son but ?



Solution

Voici la bonne réponse :



Les lignes vertes montrent les ponts que Bianca a construits. Les flèches rouges montrent comment elle a traversé les ponts :

- Sur l'île A, elle prend trois troncs et en utilise deux pour construire le premier pont. Elle traverse le pont avec le tronc restant et a $3 - 2 + 2 = 3$ troncs sur l'île B. Cela ne suffit pas pour construire un pont vers l'île D.
- Elle construit donc un pont vers l'île C avec deux troncs. Elle traverse le pont, prend les trois troncs de l'île C et revient. Elle a maintenant $3 - 2 + 3 = 4$ troncs.
- Avec ces quatre troncs, elle construit un pont vers l'île D, le traverse et prend les deux troncs de l'île D.
- Avec ces deux troncs, elle construit un pont vers l'île E, le traverse et prend les trois troncs. Elle construit des ponts vers les îles F et G. Sur l'île E, Bianca a trois troncs ; sur l'île F, $3 - 1 + 1 = 3$ troncs et sur l'île G ecore deux troncs.
- Ces deux troncs suffisent exactement pour construire un pont vers l'île H, où se trouve l'école.

Bianca a donc pu construire des ponts pour faire un chemin de l'île A à l'île H en utilisant 14 troncs en tout. Mais est-ce possible avec moins de troncs ? Pour le savoir, il faut étudier tous les chemins possibles. Comme tous les chemins passent par la longue île D, le problème peut être divisé en deux : de l'île A à l'île D, et de l'île D à l'île H :

- Bianca a utilisé 8 troncs pour les ponts entre l'île A et l'île D, et est arrivée sans tronc sur l'île D. On note son chemin ainsi : $2 - [2, 2] - 4$ (de l'île A par la ligne avec le 2 jusqu'à l'île B, puis aller-retour entre B et C par le ligne 2, puis par la ligne 4 jusqu'à D). Un chemin utilisant moins de troncs serait $3 - 4$, mais il ne peut être construit qu'avec un détour $(3 - [2, 2] - 4)$ et

nécessite donc neuf troncs ; Bianca arrive alors sur l'île D avec un tronc en réserve. Tous les autres chemins entre A et D utilisent neuf troncs ou plus.

- Bianca a utilisé six troncs pour les ponts entre les îles D et H. Elle ne peut pas construire le chemin direct $3 - 2$, même avec un tronc en réserve. Tous les autres chemins de l'île D à l'île H utilisent 6 troncs ou plus.

Ce n'est donc pas possible de construire des ponts permettant aux enfants de l'île-village A d'aller à l'île-école H avec moins de 14 troncs. Bianca a atteint son but.

C'est de l'informatique !

La carte des îles avec les lignes représentant des ponts possibles peut être représentée par un *graphe* : une structure mathématique qui relie par des arêtes des paires d'objets (aussi appelés *nœuds*). Dans un graphe, les îles sont représentées par des nœuds et les lignes par des *arêtes*. Dans ce cas, les arêtes ont des *poids* (le nombre de troncs nécessaires à la construction d'un pont), mais les nœuds également (le nombre de troncs disponibles sur l'île), ce qui est inhabituel. En informatique, on connaît plusieurs algorithmes efficaces pour calculer le plus court chemin (le chemin avec la plus petite somme des poids des arêtes) entre deux nœuds d'un graphe dont seules les arêtes ont un poids.

Le problème que Bianca veut résoudre de manière optimale dans cet exercice est plus compliqué : elle souhaite également trouver le plus court chemin, mais a encore une autre contrainte : la différence entre la somme des poids de tous les nœuds sur son chemin jusqu'ici (les troncs qu'elle a pu prendre) et la somme des poids des arêtes sur son chemin (les troncs utilisés pour les ponts) doit être plus grande que le poids de l'arête où Bianca souhaite construire le prochain pont. Pour trouver le chemin optimal, il faut ici essayer toutes les possibilité. C'est utile de diviser le problème en deux pour réduire le nombre de possibilités, et les contraintes nous permettent d'exclure beaucoup de chemins avant de les avoir complètement parcourus. En informatique, on appelle une telle méthode (essayer et exclure) le *retour sur trace* (voir aussi l'exercice « Jardin potager »).

Mots clés et sites web

- Graphe : `https://fr.wikipedia.org/wiki/Graphe_(mathématiques_discrètes)`
- Retour sur trace : `https://fr.wikipedia.org/wiki/Retour_sur_trace`

28. Notation postfixe

Une expression mathématique est constituée :

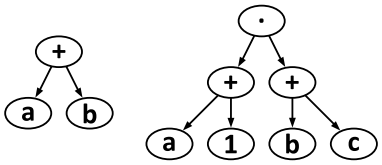
- d'un *opérateur* : $+$, $-$, $\cdot$ ou :
- et d'*opérandes* : des chiffres comme 1, 2, …, des lettres comme a, b, …, ou d'autres expressions comme $(1 + 2)$.

La structure d'une expression mathématique peut être représentée par un *arborescence*. Ce diagramme composé d'opérateurs et d'opérandes est dessiné comme cela : un cercle contenant un opérateur est relié à l'arborescence de ses opérandes. Dans le cas le plus simple, il s'agit de cercles contenant des chiffres ou des lettres.

On peut dériver la *notation postfixe* d'une expression mathématique d'une telle arborescence. Dans cette notation, on écrit chaque expression en commençant par les opérandes suivis des opérateurs.

Expression mathématique : a + b (a + 1) · (b + c)

Arborescence :

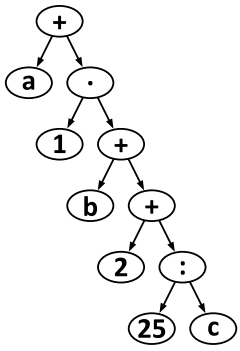


Notation postfixe : a b + a 1 + b c + ·

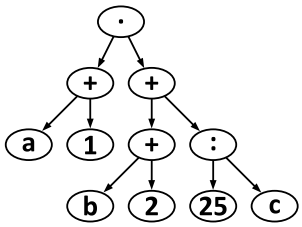
Voici la notation postfixe d'une autre expression :

a 1 + b 2 + · 25 c : +

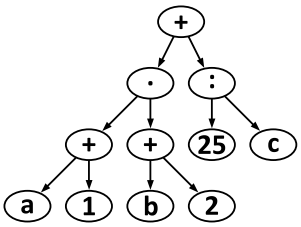
Quelle arborescence correspond à cette expression ?



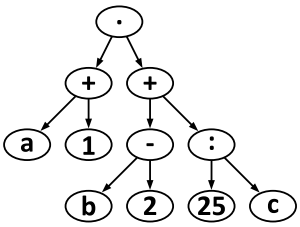
A)



B)

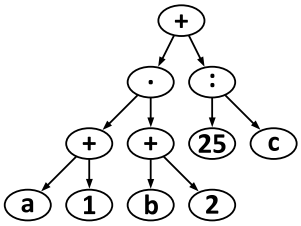


C)



D)

Solution



La bonne réponse est C :

Comme décrit dans la donnée de l'exercice, l'opérateur central d'une expression mathématique se trouve toujours tout en haut de l'arborescence (il forme sa *racine*) et tout à la fin de la notation postfixe. Si l'on veut trouver l'arborescence d'une expression en notation postfixe, il faut chercher le dernier signe de la notation postfixe tout en haut de l'arborescence, dans ce cas le +. Seules les arborescences des réponses A et C ont un + à leur racine.

L'opérateur + a deux opérandes, un à gauche et un à droite. En notation postfixe, on voit directement (à l'avant-dernier signe) que l'opérande à droite de l'expression est également une expression avec l'opérateur :. L'arborescence doit donc avoir le signe : à droite sous la racine. Ce n'est le cas que pour l'arborescence de la réponse C, qui doit donc être la bonne réponse.

On peut le vérifier en transformant toute l'arborescence de la réponse C en notation postfixe :

- les trois plus « petits » arbres, constitués de 3 éléments chacun, deviennent a 1 +, b 2 + et 25 c :.
- Les deux petits arbres de gauche deviennent les opérandes du + d'en haut ; l'arbre partiel de gauche devient donc a 1 + b 2 + ·. Le troisième petit arbre est l'opérande de droite.
- L'arborescence de la réponse C s'écrit donc a 1 + b 2 + · 25 c : + en notation postfixe, ce qui est exactment l'expression de la donnée de l'exercice.

C'est de l'informatique !

La *notation postfixe*, aussi appelée *notation polonaise inverse*, est souvent utilisée pour formuler des expressions, mathématiques ou autres (par exemple en programmation) de manière compacte et univoque. Si l'on écrivait l'expression de l'arborescence de la réponse C en notation normale (c'est à dire avec les opérateurs entre les opérandes, appelée aussi *notation infixe*), il faudrait utiliser des parenthèses : $(a+1)\cdot(b+2)+25:c$, dont on n'a pas besoin avec la notation postfixe. La notation postfixe a été introduite sous la forme de notation préfixe, avec les opérateurs devant les opérandes, par Jan Łukasiewicz (1878−1956). On utilise cette notation entre autres pour les fonctions mathématiques $f(x,y)$ et en programmation `functionname(argument1, argument2, argument3)`. En informatique, elle est utilisée entre autres pour l'*analyse syntaxique* (*parsing* en anglais) des expressions d'un langage de programmation.

Dans la passé récent, beaucoup de gens ont découvert la notation postfixe en utilisant les premières calculatrices scientifiques : elle permettait d'entrer et de calculer des expression mathématiques rapidement, de manière fiable et sans parenthèses. Il existe encore aujourd'hui une communauté de gens qui utilisent les calculatrices programmables (comme la HP-35s) avec la notation postfixe.

Mots clés et sites web

- Notation polonaise inverse :
  `https://fr.wikipedia.org/wiki/Notation_polonaise_inverse`
- Arbre syntaxique : `https://fr.wikipedia.org/wiki/Arbre_syntaxique`
- Jan Łukasiewicz : `https://fr.wikipedia.org/wiki/Jan_Łukasiewicz`
- HP 35s : `https://fr.wikipedia.org/wiki/HP-35s`

29. Cadenas

Bob a un cadenas sur la porte de sa maison. Pour l'ouvrir, il faut y entrer un code. Tous les chiffres du code doivent être différents. Actuellement, le code a cinq chiffres :

Bob a noté le code en le cachant un peu : $n >> c$ veut dire qu'il y a exactement n chiffres plus grands que c à gauche du chiffre c dans le code. Par exemple, en écrivant

$1 >> 3$

Bob note qu'il y a exactement un chiffre plus grand que 3 à sa gauche, à savoir le 4. Il a noté le code actuel comme cela :

$0 >> 0$; $3 >> 1$; $0 >> 2$; $1 >> 3$; $0 >> 4$

Bob ne trouve pas qu'un code à cinq chiffres est assez sûr. Il choisit donc un nouveau code avec les chiffres de 0 à 7. Il note le noveau code comme ceci :

$3 >> 0$; $2 >> 1$; $4 >> 2$; $4 >> 3$; $1 >> 4$; $1 >> 5$; $1 >> 6$; $0 >> 7$

Quel est le nouveau code ?

Solution

Voici la bonne réponse :

Pour déterminer le code, nous analysons la notation de Bob pour les chiffres de 0 à 7 les uns après les autres.

- 3 >> 0 : Il y a exactement trois chiffres plus grands que 0 à gauche du 0. Le 0 doit donc se trouver à la quatrième position du code.
- 2 >> 1 : Il y a exactement deux chiffres plus grands que 1 à gauche du 1. Le chiffre 1 doit donc être à la troisième position du code.
- 4 >> 2 : Il y a exactement quatre chiffres plus grands que 2 à gauche du 2. Comme les petits chiffres 0 et 1 sont déjà aux positions trois et quatre, les chiffres plus grands doivent être en première, deuxième, cinquième et sixième positions. Le chiffre 2 doit donc être à la septième position du code.
- 4 >> 3 : Il y a exactement quatre chiffres plus grands que 3 à gauche du 3. Le chiffre 3 doit donc être à la huitième et dernière position du code.
- 1 >> 4 : Il y a exactement un chiffre plus grand que 4 à gauche du 4. Le chiffre 4 doit donc être à la deuxième des positions restantes ; c'est la deuxième position du code.
- 1 >> 5 : Il y a exactement un chiffre plus grand que 5 à gauche du 5. Le chiffre 5 doit donc être à la deuxième des positions restantes ; c'est la cinquième position du code.
- 1 >> 6 : Il y a exactement un chiffre plus grand que 6 à gauche du 6. Le chiffre 6 doit donc être à la deuxième des positions restantes ; c'est la sixème position du code.
- 0 >> 7 : Il n'y a aucun chiffre plus grand que 7. Le chiffre 7 doit être à la dernière des positions libres, donc à la première position du code.

C'est de l'informatique !

Dans sa notation, Bob décrit le code par rapport à une suite ordonnée des chiffres (ou nombres) utilisés.

Regardons à nouveau le code à cinq chiffres : 0 2 4 3 1. Il est généré en prenant les chiffres ordonnés 0 1 2 3 4 et en changeant leurs positions. On appelle le résutlat une *permutation* (des chiffres de 0 à 4). Dans une permutation, l'ordre des chiffres est modifié. Par exemple, dans le code, le 4 précède le 3 alors que le 3 précède le 4 dans la suite ordonnée (car $3 < 4$). Le 3 est donc à la « mauvaise position » par rapport à l'ordre de la suite. En combinatoire, un domaine des mathématiques, on appelle cela une *inversion*.

Le code de Bob est donc une permutation, et sa notation indique le nombre de fois que chaque chiffre est inversé : Le 0 est à la bonne position, le 1 participe à 3 inversions (3 >> 1 : trois chiffres plus grands sont avant le 1), le 2 est à la bonne position, le 3 est inversé une fois, le 4 est à la

bonne position. Cette suite de nombre d'inversions s'appelle une *séquence d'inversions*. La somme du nombre d'inversions décrit également le degré de désordre d'une permutation – voir l'exercice « Train de marchandises ».

Nous avons à présent trois suites – le code (la permutation), le suite ordonnée et la séquence d'inversions – et les regroupons dans un tableau :

Code / Permutation	0	2	4	3	1
Suite ordonnée	0	1	2	3	4
Séquence d'inversions	0	3	0	1	0

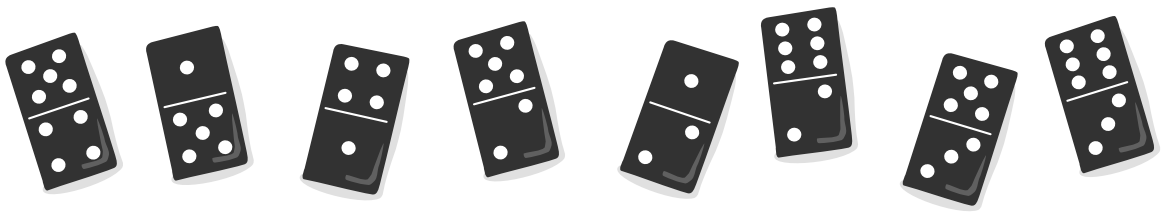
La description de la solution a montré qu'il existe un algorithme efficace pour déterminer la permutation en partant de la séquence d'inversion. Il suffit de parcourir une fois la séquence d'inversions. L'informatique traite souvent de problèmes combinatoires et il y a beaucoup d'algorithmes pour résoudre de tels problèmes. Ils peuvent être utilisés pour la résolution automatique de casses-tête (comme les sudokus), mais aussi pour des problèmes « sérieux ». En général, ils sont beaucoup plus compliqués que l'algorithme utilisé pour résoudre cet exercice du Castor.

Mots clés et sites web

- Permutation : `https://fr.wikipedia.org/wiki/Permutation`
- Combinatoire : `https://fr.wikipedia.org/wiki/Combinatoire`

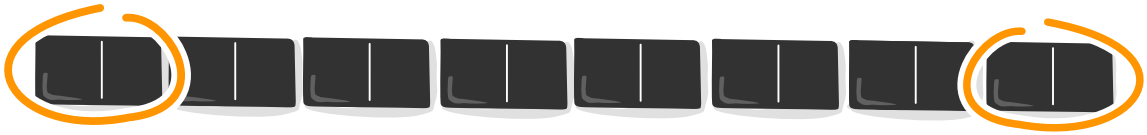
30. Dominos

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.

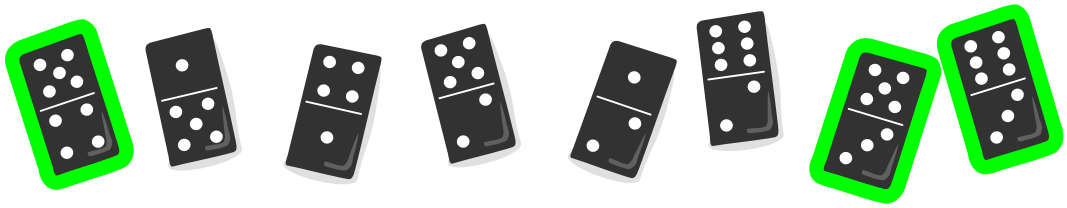
Il y a plusieurs manières d'aligner ces huit dominos, mais certaines pièces ne peuvent jamais se trouver au début ou à la fin de la ligne.



De quelles pièces s'agit-il ?

Solution

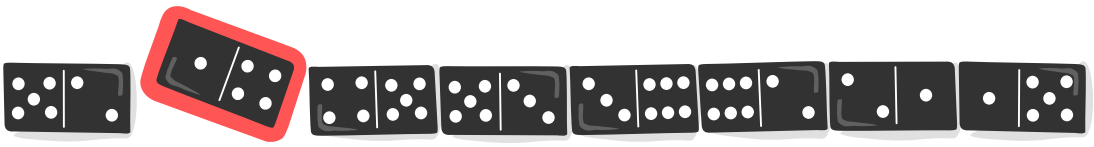
Trois des huits pièces ne peuvent pas être posées au début ou à la fin de la ligne :



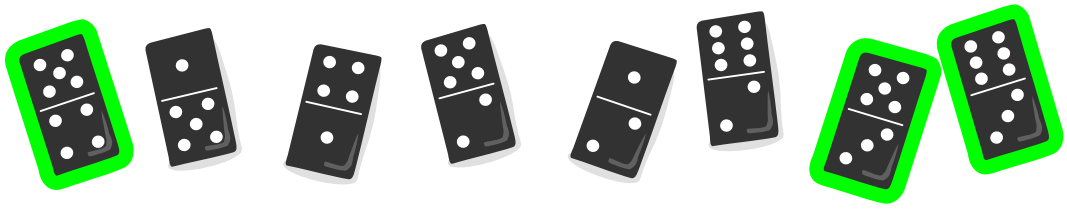
Pour résoudre l'exercice, on considère les nombres de points sur les 16 cases des dominos. Nous comptons combien de fois chaque nombre de points est présent et regardons si cette fréquence est paire ou impaire :

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.

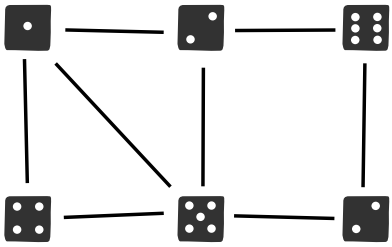


Comme il y a des cases présentes un nombre impair de fois parmi les huit dominos de cet exercice, ce sont eux qui doivent se trouver aux extrémités. Les dominos ayant deux cases avec fréquence paire ne peuvent donc pas être posés aux extrémités de la ligne. Ce sont les dominos suivants :

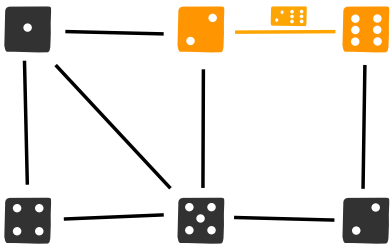


C'est de l'informatique !

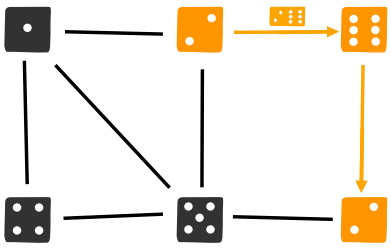
Il y a plusieurs possibilités de poser les dominos de cet exercice du Castor en une ligne correcte. Pour avoir une meilleure vue d'ensemble, les informaticiens et informaticiennes utilisent des *graphes* :



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.

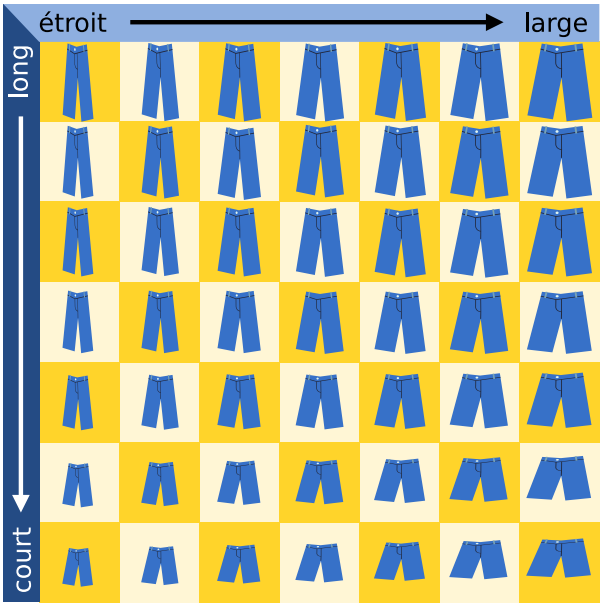
Mots clés et sites web

- 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)`

31. Nouveau pantalon

Christian a besoin d'un nouveau pantalon. Le magasin vend son pantalon préféré en sept longueurs et sept largeurs différentes. Les 49 tailles sont rangées dans les cases d'une étagère, classées par largeur et longueur.

Comme Christian ne connaît pas sa taille, il doit trouver la bonne taille en essayant les pantalons. À chaque essai, Christian note si le pantalon lui va ou s'il a besoin d'un pantalon plus large, plus étroit, plus court ou plus long. Pour qu'un pantalon lui aille, il faut que la largeur et la longueur soient bonnes.



Le vendeur gémit : ça risque de prendre du temps de trouver la bonne taille parmi 49.

Mais Christian a pensé à une méthode lui permettant de toujours trouver la bonne taille avec le moins d'essais possible.

Combien de pantalons Christian doit-il au maximum essayer avant d'identifier la bonne taille ?

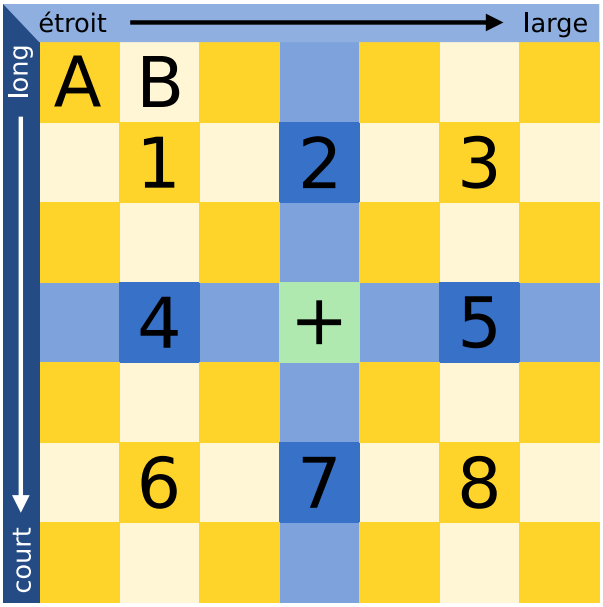
Solution

La bonne réponse est 2.

Christian pourrait bien sûr avoir de la chance et tomber sur la bonne taille de pantalon à son premier essai, mais il ne peut pas se fier au hasard et procède d'après la méthode suivante :

Il commence par essayer le pantalon du milieu (à la position + sur l'image) et en vérifie la longueur et la largeur.

- Si la longueur et la largeur vont, il a trouvé le pantalon à la bonne taille.
- Si le pantalon est trop court et trop large, le bon pantalon se trouve dans le secteur 1.
- Si le pantalon est trop court mais a la bonne largeur, le bon pantalon se trouve dans le secteur 2.
- Si le pantalon est trop court et trop étroit, le bon pantalon se trouve dans le secteur 3.
- Et ainsi de suite pour les secteurs 4 à 8.



Imaginons que le pantalon ayant la bonne taille soit dans le secteur 1. Pour son deuxième essai, Christian choisit le pantalon rangé au milieu du secteur 1. Il y a à nouveau plusieurs possibilités :

- Si la longueur et la largeur vont, il a trouvé le pantalon à la bonne taille.
- Si le pantalon est trop court et trop large, Christian sait que le bon pantalon se trouve en position A.
- Si le pantalon est trop court mais a la bonne largeur, Christian sait que le bon pantalon se trouve en position B.
- Et ainsi de suite pour les autres positions du secteur 1.

Comme la case centrale de chaque secteur numéroté ne possède qu'une case voisine dans chaque directions, aucun essai supplémentaire n'est nécessaire. Christian a donc besoin de deux essais au maximum pour trouver la bonne taille de pantalon.

C'est de l'informatique !

La méthode que Christian utilise pour ses essais de pantalons s'appelle *recherche dichotomique* en informatique. Lors de la recherche dichotomique d'un objet dans une liste d'objets triés, l'objet du milieu est comparé à l'objet recherché. Si l'objet du milieu ne correspond pas à celui que l'on recherche, on sait dans quelle moitié de la liste l'objet recherché se trouve et l'on peut y continuer la recherche dichotomique. Ainsi, la liste est est séparée en deux à chaque étape de la recherche – d'où « dichotomique ». De cette manière, on trouve rapidement l'objet recherché. Environ 10 étapes sont nécessaire pour rechercher dans une liste de 1000 objets, et 20 étapes pour une liste de 1 000 000 objets. De manière générale, on peut le formuler ainsi : il faut en moyenne $\log(n)$ étapes pour un tableau de n objets (la fonction log est le logarithme en base 2). La recherche dichotomique est souvent utilisée par les programmes informatiques pour les recherches dans les données triées en raison de sa rapidité.

Dans cet exercice du Castor, l'espace de recherche (les pantalons dans l'étagère) est séparé en deux dimensions (longueur et largeur). Christian peut donc appliquer la recherche dichotomique dans les deux dimensions en même temps, et l'espace de recherche n'est pas divisé en deux à chaque étape, mais directement en huit – pour autant que Christian ne soit pas directement tombé sur la bonne taille !

Mots clés et sites web

- Recherche dichotomique : `https://fr.wikipedia.org/wiki/Recherche_dichotomique`
- Algorithme de recherche : `https://fr.wikipedia.org/wiki/Algorithme_de_recherche`

32. Détecteur de conflit

Anna et Ben veulent construire un « détecteur de conflit » qui indique s'ils sont d'avis différents.

Pour cela, ils utilisent des éléments qui peuvent être dans deux états : Oui ou Non. Deux éléments peuvent être reliés par un câble.

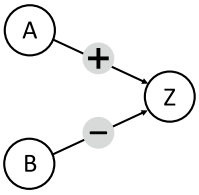
Lorsqu'un élément est

- dans l'état Oui : il transmet un signal par tous ses câbles sortants ;
- dans l'état Non : il ne transmet aucun signal.

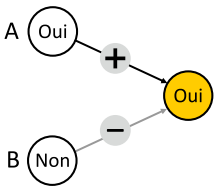
On peut régler chaque câble pour qu'un signal transmis devienne positif (+) ou négatif (–) pour l'élément de droite auquel il est relié. Un élément qui reçoit des signaux passe à l'état Oui s'il reçoit plus de signaux positifs que de signaux négatifs, et reste à l'état Non sinon.

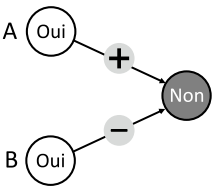
Anna fixe l'état de l'élément A et Ben l'état de l'élément B ; ce sont les entrées du détecteur.

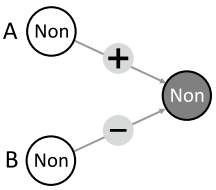
Anna et Ben commencent par construire cette machine :

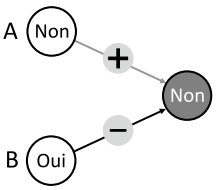


Ils remarquent que l'élément Z n'est dans l'état Oui que lorsque A est dans l'état Oui et B est dans l'état Non. Ce n'est pas ce qu'ils veulent.



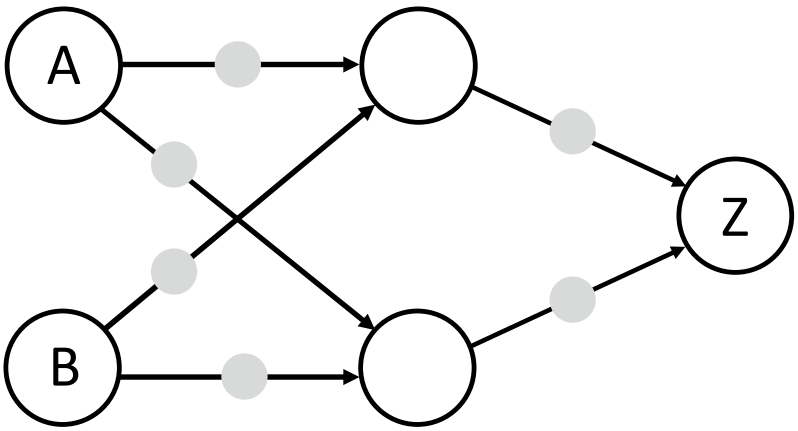






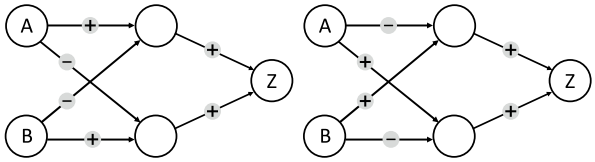
Anna et Ben construisent alors une plus grande machine (image ci-dessous) et sont sûrs qu'elle peut être un détecteur de conflit : l'état de Z ne doit être Oui que lorsque les état de A et B sont différents (Oui et Non ou Non et Oui). Sinon, Z doit être dans l'état Non. Il ne reste plus qu'à régler les câbles correctement.

Règle le type de signal, positif ou négatif, transmis par chaque câble afin que le détecteur de conflit fonctionne correctement.

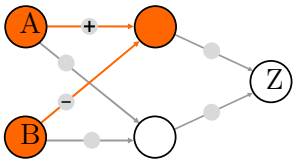


Solution

Les deux réponses suivantes sont justes :

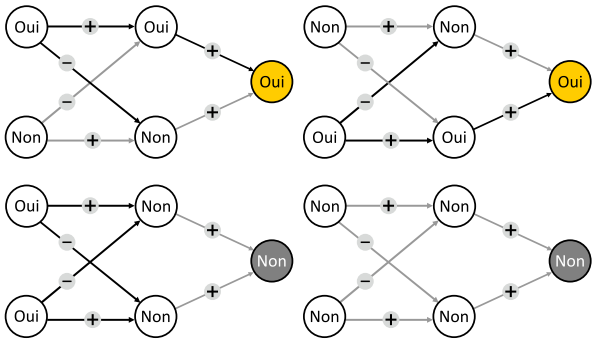


L'élement de sortie Z du détecteur de conflit doit être dans l'état Oui seulement en présence de deux entrées différentes (A = Oui et B = Non ainsi que A = Non et B = Oui). Z ne peut être en l'état Oui que s'il reçoit plus de signaux positifs que négatifs par ses deux câbles entrant. Au moins un des câbles doit donc transmettre un signal positif (+). Imaginons que seul le câble du haut menant à Z soit réglé sur +. L'élément en haut au centre doit alors pouvoir reconnaître les deux combinaisons d'entrées en conflit, donc être en l'état Oui dans les deux cas. Mais cet élément forme, avec les deux éléments d'entrée A et B, exactement la machine qu'Anna et Ben avait construite au début. Cet élément ne peut donc être en l'état Oui que dans un des deux cas de conflit, et l'un des câbles doit être réglé sur + et l'autre sur – pour cela :



Il faut donc un élément central pour chacun des cas de conflit, un pour A = Oui et B = Non et un pour A = Non et B = Oui. Les câbles entrant dans le premier élément doivent être réglés sur + (câble sortant de A) et – (câble sortant de B), les câbles entrant dans l'autre élément sur – (A) et + (B). Lequel des deux éléments centraux réagit à quel cas n'a pas d'importance, c'est pour cela qu'il y a deux possibilités de régler les câbles allant de A et B au milieu. Comme chaque élément central est dans l'état Oui dans exactement un des deux cas de conflit, les deux câbles sortant du milieu et entrant en Z doivent être réglés sur + afin que Z soit dans l'état Oui exactement dans ces deux cas.

L'image ci-dessous montre le fonctionnement du détecteur de conflit pour la première bonne réponse. On voit que l'élément du haut au milieu reconnaît le cas A = Oui et B = Non et celui du bas le cas A = Non et B = Oui. L'élement reconnaissant le conflit transmet un signal positif à Z, et Z passe donc en l'état Oui. Pour les autres entrées (A = Oui et B = Oui ainsi que A = Non et B = Non), les deux éléments du milieu sont en l'état Non, Z ne reçoit donc pas de signal positif et passe en l'état Non.



C'est de l'informatique !

Le détecteur de conflit traite deux valeurs d'entrée (Oui et Non) et retourne la sortie Oui lorsque les deux valeurs d'entrée sont différentes. Cette fonction logique s'appelle un OU exclusif (XOR, disjonction). La première machine d'Anna et Ben décrite dans cet exercice est une version simplifiée d'un *perceptron* comme décrit par Frank Rosenblatt en 1957. L'élément de sortie simule une cellule nerveuse (un neurone) qui peut traiter des signaux d'entrée et générer un signal de sortie. On peut implémenter les fonctions logiques ET et OU à l'aide d'un perceptron, mais pas le OU exclusif. Pour cela, une couche d'éléments supplémentaire est nécessaire, comme décrit dans cet exercice. C'est uniquement dans les années 80 que ceci a été découvert (Rumelhart, Hinton & Williams, 1986) et qu'on a par la suite été en mesure de programmer des réseaux de neurones artificiels qui fonctionnent de manière similaire au cerveau humain et sont capables, par exemple, d'analyser des images et d'y reconnaître des objets. On a développé des méthodes informatiques permettant à de grands réseaux de neurones comprenant beaucoup de couches et d'éléments d'effectuer leurs calculs de manière efficace. Ces réseaux forment la base de beaucoup de systèmes d'intelligence artificielle actuels.

Mots clés et sites web

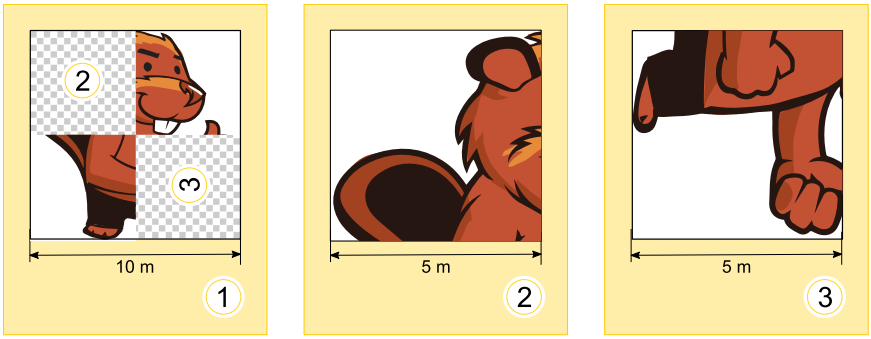
- Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning representations by back-propagating errors. Nature, 323(6088), 533−536 :
  `http://www.cs.toronto.edu/~hinton/absps/naturebp.pdf`
- Perceptron : `https://fr.wikipedia.org/wiki/Perceptron`
- Fonction OU exclusif : `https://fr.wikipedia.org/wiki/Fonction_OU_exclusif`

33. Peinture récursive

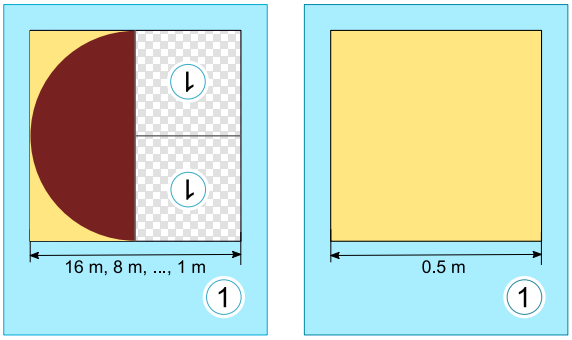
Tina et Ben aident à préparer une exposition temporaire au musée de l'informatique. Ils doivent peindre une image de 16×16 mètres sur le sol d'une salle d'exposition. L'artiste leur donne un set de cartes d'instruction de peinture avec des indications sur les éléments des images, leurs dimensions et leurs orientations.

Certaines cartes d'instruction ont des cases numérotées qui font référence à d'autres cartes.

Voici un exemple d'un précédent projet de peinture par carte. Si on effectue les instructions des trois cartes de la bonne manière, on obtient l'image d'un castor :

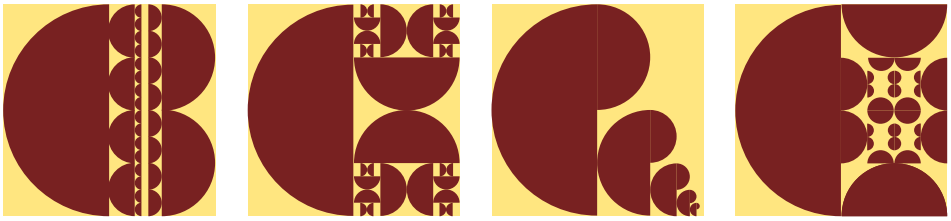


Tina et Ben reçoivent ces deux cartes pour l'exposition temporaire :



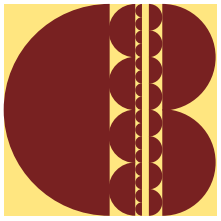
Ben fronce les sourcils. « Comment ça marche ? la carte de gauche réfère à elle-même, et en plus les deux cartes ont le même numéro ! » Tina rigole : « On va y arriver ! Commençons par la carte de gauche, la carte de droite nous dira plus tard quand nous devons arrêter de peindre. »

De quoi aura l'air le sol de la salle d'exposition ?



A) B) C) D)

Solution

La réponse A est juste : 

La carte d'instruction de gauche montre qu'un demi-cercle doit être peint sur la moitié gauche du sol, son côté arrondi tourné vers la gauche. La même carte d'instruction doit être utilisée deux fois pour le côté droit du sol. L'orientation de images sur le sol doit être la même que l'orientation des « 1 » sur la carte.

Les deux « 1 » sur la carte sont tourné de 180 degrés, la tête en bas. Les éléments d'images doivent donc également être tournés de façon à ce que le côté arrondi des demi-cercles soit tourné du côté opposé. Lors de la première application de la carte 1 (pour une largeur de 16 m), le côté arrondi du demi-cercle est tourné vers la gauche ; pour 8 m, vers la droite ; pour 4 m, à nouveau vers la gauche ; et ainsi de suite. Pour 0,5 m, la deuxième carte 1 est utilisée : Ben et Tina finissent de peindre la surface restante et peuvent s'arrêter.

De cette manière, c'est exactement l'image de la réponse A qui est peinte sur le sol.

C'est de l'informatique !

La première des deux cartes d'instruction 1 dans cet exercice du Castor fait référence à elle-même. Elle appelle, pour ainsi dire, Ben et Tina à s'appliquer elle-même une fois de plus avec une largeur différente. En informatique, les instructions qui font référence à elles-mêmes sont dites *récursives*. Ce terme vient du latin *recurrere* (« revenir » en français). La récursivité est un concept puissant. Certains problèmes complexes peuvent être résolus à l'aide d'une instruction récursive courte et simple.

Une instruction récursive doit contenir une condition définissant quand la récursivité doit être terminée. Sinon, le récursivité continue jusqu'à ce qu'une des ressources nécessaires soit épuisée, comme la mémoire de l'ordinateur ou la patience de l'utilisateur. Dans cet exercice, c'est la deuxième carte 1 qui a cette fonction : elle doit être utilisée lorsque la condition qu'une surface de 0,5 m de largeur doit encore être peinte est remplie. Comme elle ne fait référence à aucune carte, elle termine la récursivité.

Mots clés et sites web

- Programmation : `https://fr.wikipedia.org/wiki/Programmation_informatique`
- Récursivité : `https://fr.wikipedia.org/wiki/Récursivité`
- Algorithme récursif : `https://fr.wikipedia.org/wiki/Algorithme_récursif`

34. Décryptage

Un code spécial pour les textes remplace chaque lettre par un mot composé de chiffres entre 0 et 9. La règle suivante s'applique : aucun mot du code ne peut commencer par un mot du code chiffrant une autre lettre.

Le lettre `X`, par exemple, est chiffrée par 12. `Y` peut donc être chiffré par 2, car 12 ne commence pas par 2 (et 2 ne commence pas par 12). `Z` peut être chiffré par 11, car ni 12, ni 2 ne commence par 11 et 11 ne commence ni par 12, ni par 2. `Z` ne pourrait par contre par être chiffré par 21, car 21 commence par 2, qui est le code de `Y`.

Le mot `MEMORY` est chiffré par la suite de chiffres 12112233321.

Sépare la suite de chiffres en mots représentant chacune des lettres.

Solution

La bonne réponse est 1 21 1 22 33 321.

On commence par la gauche de la suite de chiffres. Si `M` était chiffré par le mot 12, `E` devrait être chiffré par 1, car 12 revient tout de suite après pour le deuxième `M`. Ceci serait contraire à la règle : le code pour `M` commencerait par 1, qui est le code pour `E`. De plus longs mots (121, 1211, 12112, etc.) ne peuvent pas coder le `M`, car ce mot chiffré doit apparaître deux fois dans le cryptogramme, ce qui n'est pas le cas de ces mots. Le mot chiffré pour `M` doit donc être le 1.

Celui-ci doit être suivi du mot chiffré pour le `E`, puis à nouveau du `M` (donc du 1). Le mot chiffré pour `E` doit donc être soit 2, soit 21, soit 211223332. Le 2 n'est pas possible, car le mot en clair commencerait par `MEMM`. 211223332 n'est pas possible non plus, car le mot en clair serait alors `MEM`. Le mot chiffré pour `E` doit donc être 21. 1 21 1 est donc le code pour `MEM`.

Le reste de la suite de chiffres, c'est à dire 2233321, code les lettres `ORY`. Le 2 ne peut pas coder le `O`, sinon `MEM` serait suivi de `OO`. Le mot chiffrant le `O` doit donc contenir au moins 22. À la fin, 1 et 21 sont déjà assignés à `M` et `E`, respectivement ; le mot chiffré pour `Y` doit donc au moins être 321. Entre 22 et 321 se trouve 33, ce qui doit être le mot chiffré pour `R` : la seule autre possibilité serait le 3. Le mot chiffré pour `Y` devrait alors être 3321, et commencerait par 3, le mot chiffré pour `R`, ce que la règle interdit. La séparation de la deuxième partie est donc 22 33 321.

C'est de l'informatique !

Le code utilisé dans cet exercice est un exemple de *code préfixe*. Un préfixe est une suite de symboles précédent une autre suite de symboles. Dans un code préfixe, aucun mot du code ne peut être le préfixe d'un autre mot du code. Cela veut dire qu'aucun mot du code ne peut commencer par un autre mot du code.

Les mots des codes préfixes ont des longueurs différentes. L'avantage de la règle des préfixes est qu'aucun symbole séparateur entre les mots du code n'est nécessaire : on peut toujours reconnaître à quelle position le prochain mot commence. En choisissant des mots courts pour les lettres fréquentes, on peut chiffrer des textes de manière efficace sans utiliser trop de place.

Le codage de Huffman est une méthode permettant de trouver un code préfixe idéal. Elle est très répandue et est utilisée, entre autres, pour les formats JPEG et MP3.

Mots clés et sites web

- Code préfixe : `https://fr.wikipedia.org/wiki/Code_préfixe`
- Codage de Huffman : `https://fr.wikipedia.org/wiki/Codage_de_Huffman`
- Cryptographie : `https://fr.wikipedia.org/wiki/Cryptographie`
- Cryptanalyse : `https://fr.wikipedia.org/wiki/Cryptanalyse`

A. Auteur·e·s des exercices

Eslam AbdElAal
Akram Ahmed
Nursultan Akhmetov
Somayah Albaradei
Laila Alharthi
Esraa Almajhad
Khairul Anwar
Aldrich Ellis Catapang Asuncion
James Atlas
Masiar Babazadeh
Leonardo Barichello
Liam Baumann
Wilfried Baumann
Tim Bell
Javier Bilbao
Leonardo Cavalcante
Špela Cerar
Diego César
Sarah Chan
Zaheer Chothia
Marios Omar Choudary
Gunnar Collier
Eimear Colreavy
Raluca Constantinescu
Kris Coolsaet
Lucia Crivelli
María Eugenia Curi
Valentina Dagienė
Darija Dasović
Christian Datzko
Justina Dauksaite
Nora A. Escherle
Georgios Fesakis
Gerald Futschek
Bence Gaál
Emily Gates
Anaclara Gerosa
Adam Grodeck
Štefan Gura
Juan Gutiérrez
Hans-Werner Hein
Tracy Henderson
Josefine Hiebler
Mathias Hiron
Alisher Ikramov
Thomas Ioannou
Hyun-seok Jeon
Filiz Kalelioğlu
Merel Kämper
David Khachatryan
Gohar Khachatryan
Jihye Kim

Vaidotas Kinčius
Mhairi King
Jia-Ling Koh
Sophie Koh
Víctor Koleszar
Taina Lehtimäki
Marielle Léonard
Angélica Herrera Loyo
Carlos Luna
Michael Weigend
Dario Malchiodi
Yong Mao
Yoshiaki Matsuzawa
Anna Morpurgo
Madhavan Mukund
Natalia Natalia
Tom Naughton
Jalil Nedaeepour
Graciela Oyhenard
Özgür Özdemir
Marika Parviainen
Elsa Pellet
Jean-Philippe Pellet
Zsuzsa Pluhár
Wolfgang Pohl
Ilya Posov
Sergey Pozdniakov
JP Pretti
Estela Ramić
Omar Colon Reyes
Chris Roffey
Karima Sayeh
Kirsten Schlüter
Margareta Schlüter
Eljakim Schrijvers
Rosario Schunk
Giovanni Serafini
Kim Seulki
Vipul Shah
Rostyslav Shpakovych
Jacqueline Staub
Alieke Stijf
Gabrielė Stupurienė
Marianne Thut
Monika Tomcsányiová
Ahto Truu
Laura Ungureanu
Svetlana Unković
Jiří Vaníček
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

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.