INFORMATIK-BIBER SCHWEIZ
CASTOR INFORMATIQUE SUISSE
CASTORO INFORMATICO SVIZZERA

Exercices et solutions 2023

Années HarmoS 13/14/15

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

Éditeurs :

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

01010011010101100100100
10000010010110101010011
01010011010010010100010
10010110101010011010011
01000100101001001000001

SSIE

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

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

Ont collaboré au Castor Informatique 2023

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

Cheffe de projet : Nora A. Escherle

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

Le choix des exercices a été fait en collaboration avec les organisateur de Bebras en Allemagne, Autriche, Hongrie, Slovaquie et Lithuanie. Nous remercions en particulier :
Valentina Dagienė, Vaidotas Kinčius : Bebras.org, Lituanie
Wolfgang Pohl, Jakob Schilke : Bundesweite Informatikwettbewerbe (BWINF), Allemagne
Hannes Endreß : Materna Information & Communications SE, Allemagne
Ulrich Kiesmüller : Simon-Marius-Gymnasium Gunzenhausen, Allemagne
Kirsten Schlüter : Bayerisches Staatsminiserium für Unterricht und Kultus, Allemagne
Margareta Schlüter : Universität Tübingen, Allemagne
Jacqueline Staub : Universität Trier, Allemagne
Michael Weigend : WWU Münster, Allemagne
Wilfried Baumann, Liam Baumann, Josefine Hiebler : Österreichische Computer Gesellschaft, Autriche
Gerald Futschek : Technische Universität Wien, Autriche
Zsuzsa Pluhár : ELTE Informatikai Kar, Hongrie

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

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

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

© Castor Informatique 2023, SSIE

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.

Les exercices sont protégés par une licence Creative Commons Paternité – Pas d'Utilisation Commerciale – Partage dans les Mêmes Conditions 4.0 International. Les auteur·e·s sont cité·e·s en p. 57.

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.

© Castor Informatique 2023, SSIE

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/`

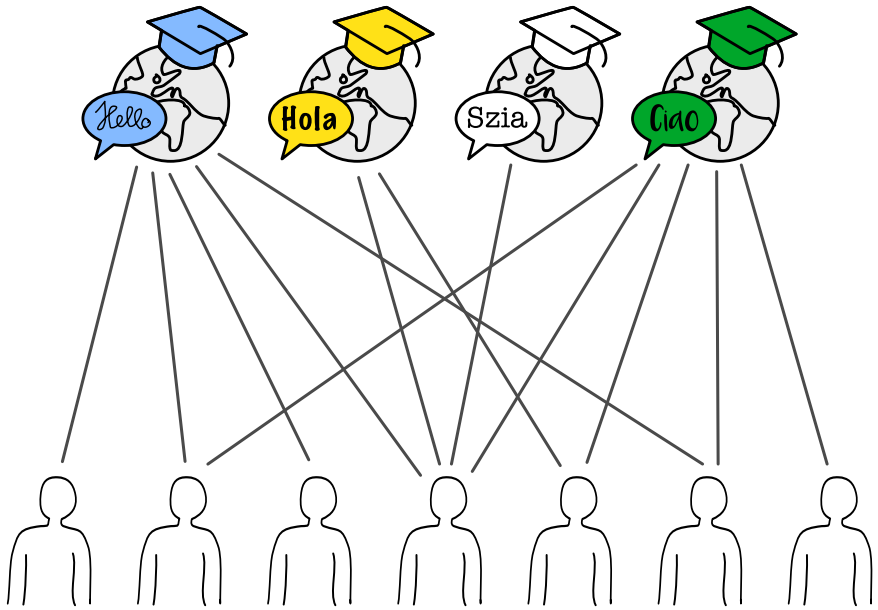
 © Castor Informatique 2023, SSIE

Table des matières

© Castor Informatique 2023, SSIE

1. Répartition des tâches

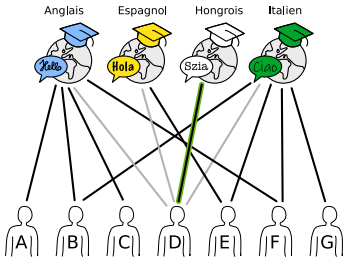
Une école de langue organise quatre cours d'été. Sur l'image ci-dessous, les lignes montrent quel enseignant de l'école est capable de donner quel cours.



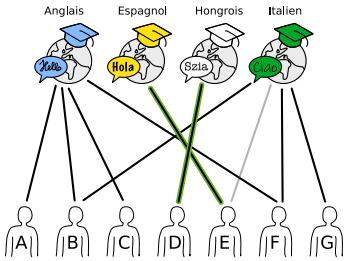
Chaque enseignant ne peut donner qu'un seul cours. Il y a quand même plusieurs possibilités d'assigner un enseignant capable à chaque cours.

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

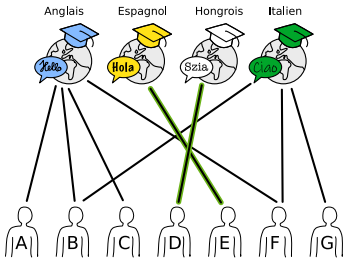
Solution



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.

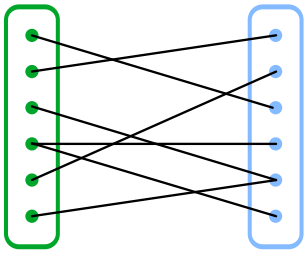
Il y a ainsi dix possibilités en tout d'assigner un enseignant capable à chaque cours :

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.

 © Castor Informatique 2023, SSIE



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`

© Castor Informatique 2023, SSIE

2. Chantier castor

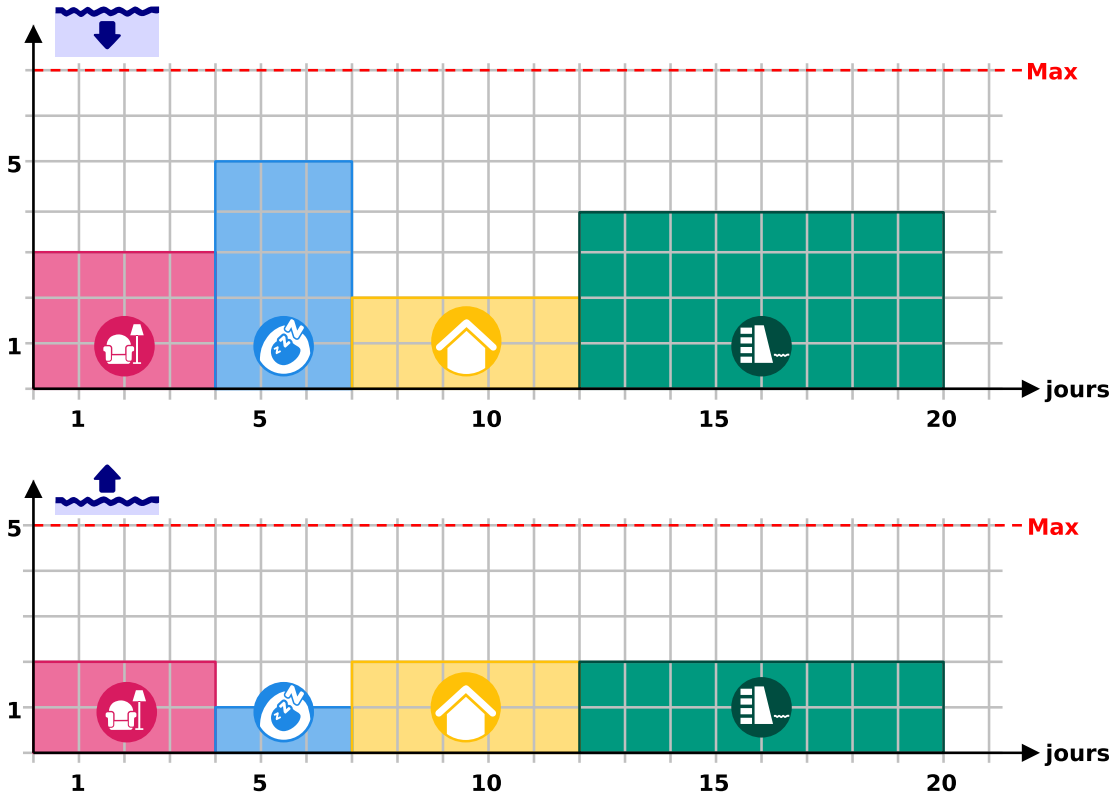
Une hutte de castor est composée de quatre parts qui sont toutes en partie sous l'eau et en partie au-dessus de l'eau.

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 :

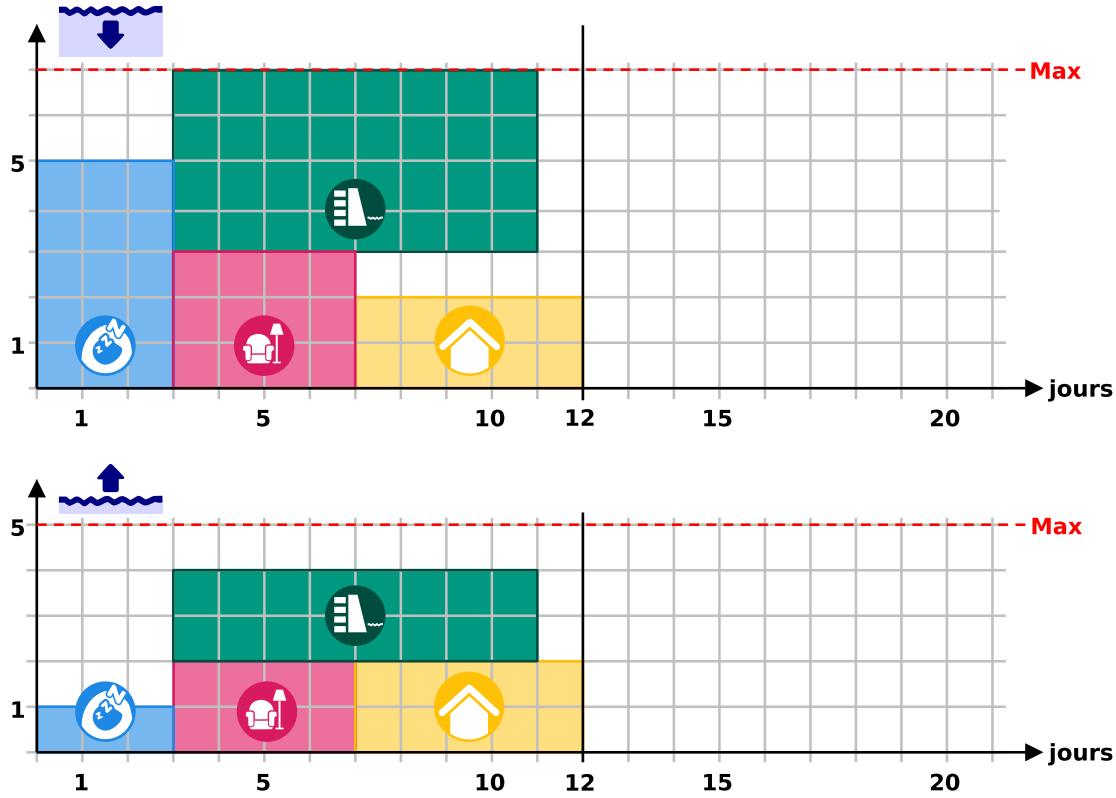


Développe un plan de travail permettant de finir la hutte en le moins de jours possible. Combien de jours faut-il ?

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

 © Castor Informatique 2023, SSIE

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`

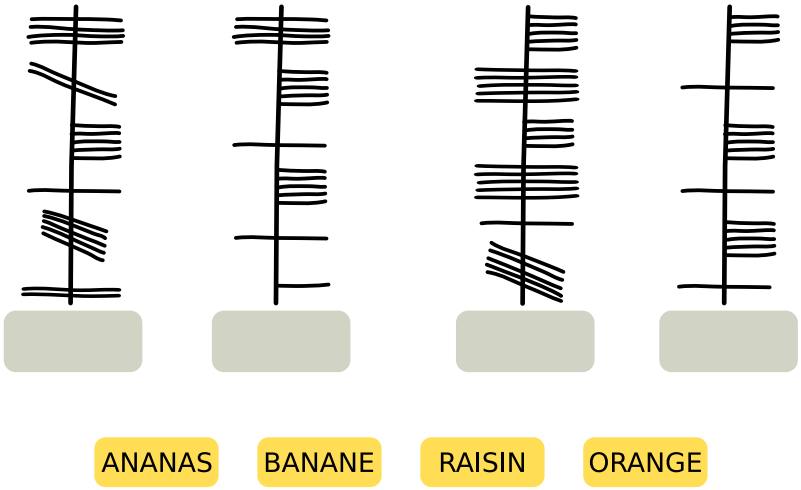
© Castor Informatique 2023, SSIE

3. Ogham

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

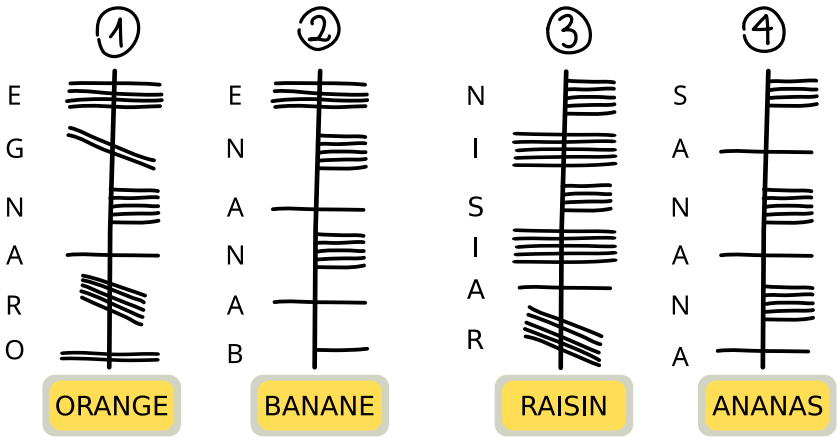
Sue utilise l'écriture oghamique comme code secret. Elle écrit ainsi quatre mots – ses fruits préférés : ANANAS, BANANE, RAISIN et ORANGE.

Quel mot correspond à quel code en Ogham ?



Solution

Voici la bonne réponse :



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

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

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

C'est de l'informatique !

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

 © Castor Informatique 2023, SSIE

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.

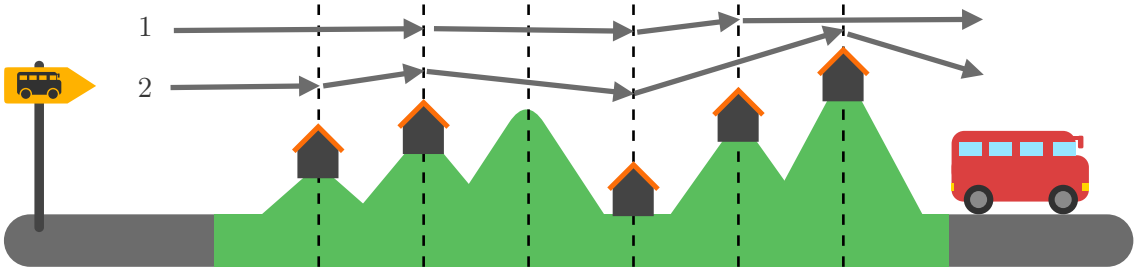
Mots clés et sites web

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

© Castor Informatique 2023, SSIE

4. 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

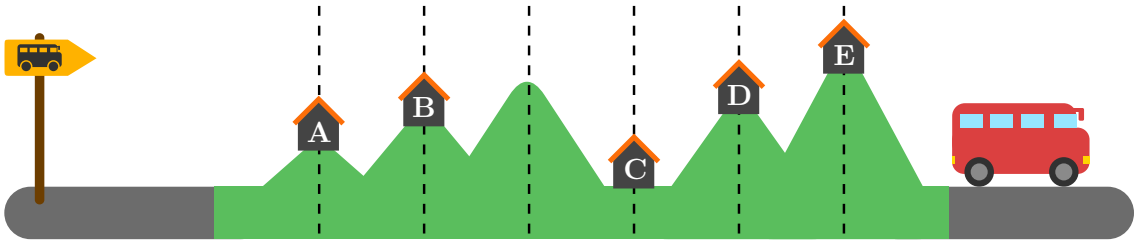
E) 6 randonnées

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

 © Castor Informatique 2023, SSIE

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.

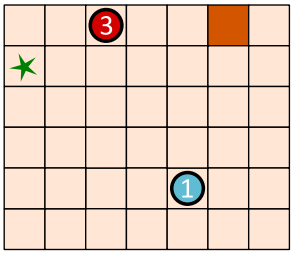
Mots clés et sites web

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

© Castor Informatique 2023, SSIE

5. Robots tamponneurs

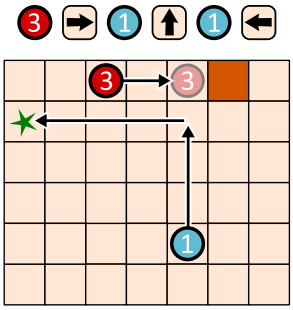
Les robots tamponneurs sont des robots très simples. Ils roulent sur un plateau de jeu divisé en cases.



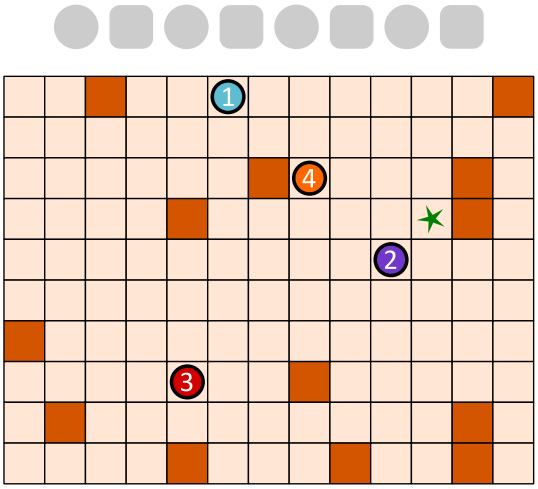
Pour les diriger, on commence par sélectionner un robot tamponneur. On l'envoie ensuite dans une certaine direction à l'aide d'une flèche-commande : en haut, en bas, à gauche ou à droite. Le robot tamponneurs roule ensuite tout droit dans la direction de la commande jusqu'à ce qu'il rencontre un obstacle 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 1 atteigne le but et y reste.

Le robot tamponneur 1 atteint le but en suivant la suite de commandes suivante :



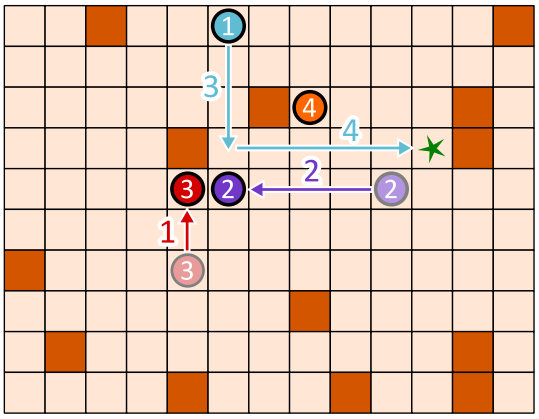
Crée une suite de commandes avec quatre flèches permettant au robot tamponneur 1 d'atteindre le but .



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

 © Castor Informatique 2023, SSIE

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`

© Castor Informatique 2023, SSIE

6. Les courses d'Emma

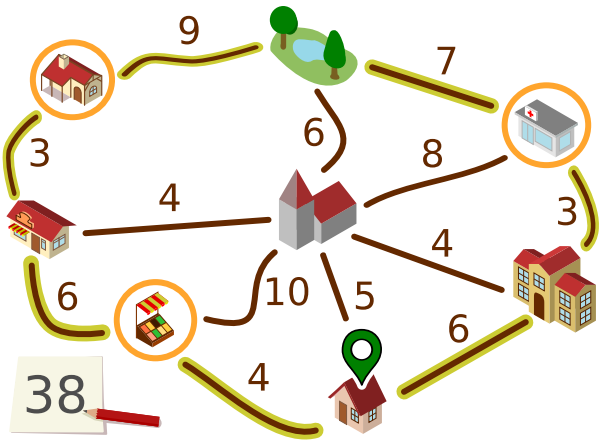
Emma est à la maison . Elle doit faire trois courses et revenir à la maison :

- 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.

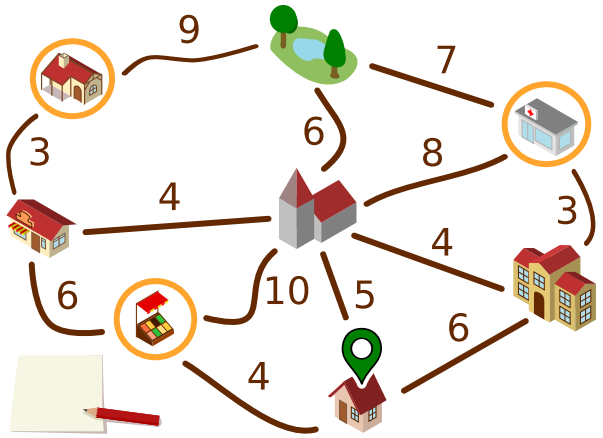
Emma a noté sur un plan de combien de minutes elle a besoin pour parcourir les chemins entre différents endroits de sa ville. Elle a aussi noté quels chemins elle prend pour faire ses courses.

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



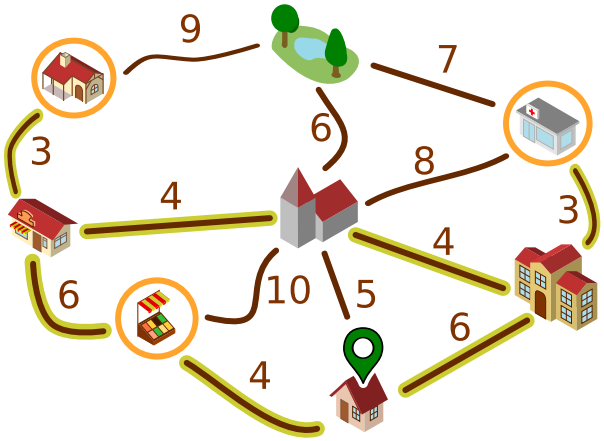
Emma se demande si elle pourrait être plus rapide. Peut-être en faisant l'aller-retour sur certains chemins ?

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

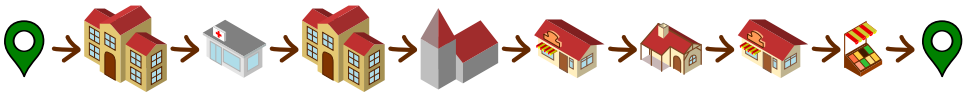


Solution

Voici la bonne réponse :



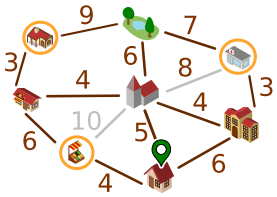
Emma peut faire le trajet suivant le long des chemins sélectionnés (ou dans la direction opposée) :



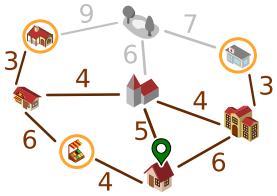
Pour ce trajet, elle a besoin de $6 + 3 + 3 + 4 + 4 + 3 + 3 + 6 + 4 = 36$ minutes.

 © Castor Informatique 2023, SSIE

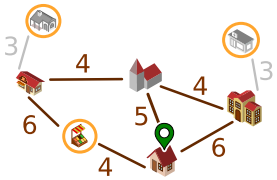
Nous voulons maintenant démontrer qu'il ne peut pas y avoir de trajet plus court. Pour cela, nous utilisons une version simplifiée du plan.



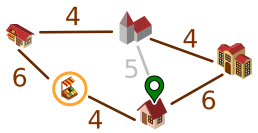
Nous pouvons ignorer les chemins en gris. Il existe des chemins plus courts passant par d'autres endroits entre les endroits qu'ils relient.



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.

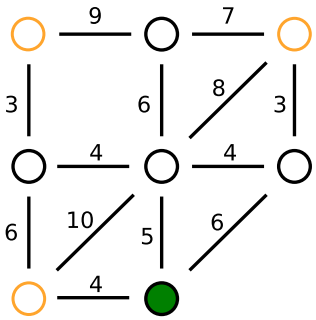


Emma doit aller à la pharmacie et au kiosque. Elle ne peut y aller que depuis la boulangerie et l'école, respectivement. Elle doit faire l'aller-retour entre ces endroits, ce qui dure $3 + 3 = 6$ minutes pour chacun, donc 12 minutes en tout. Nous en prenons note et simplifions la plan en enlevant les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

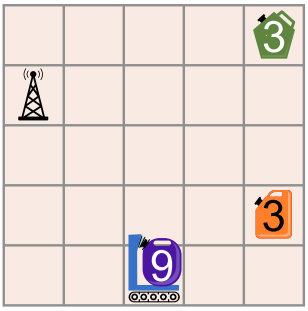
 © Castor Informatique 2023, SSIE

7. 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 !

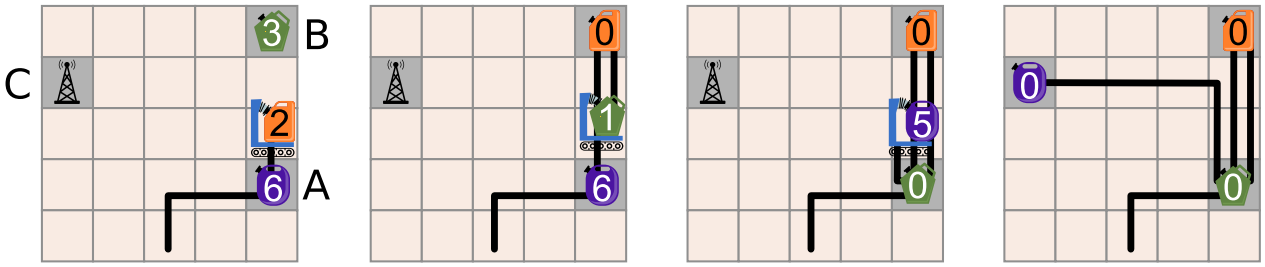
Voici la mission de Zérobot : il doit aller à la station de base  en vidant tous les réservoirs (niveau 0).

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).

 © Castor Informatique 2023, SSIE

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`

© Castor Informatique 2023, SSIE

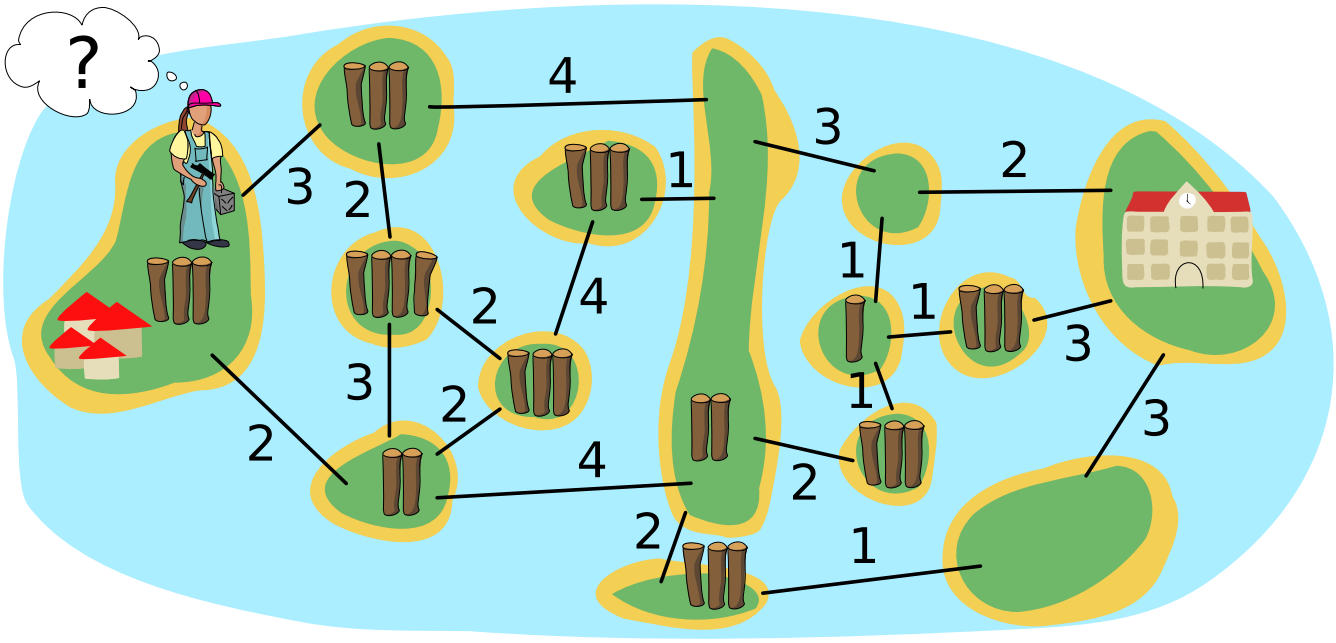
8. Raccordement

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.

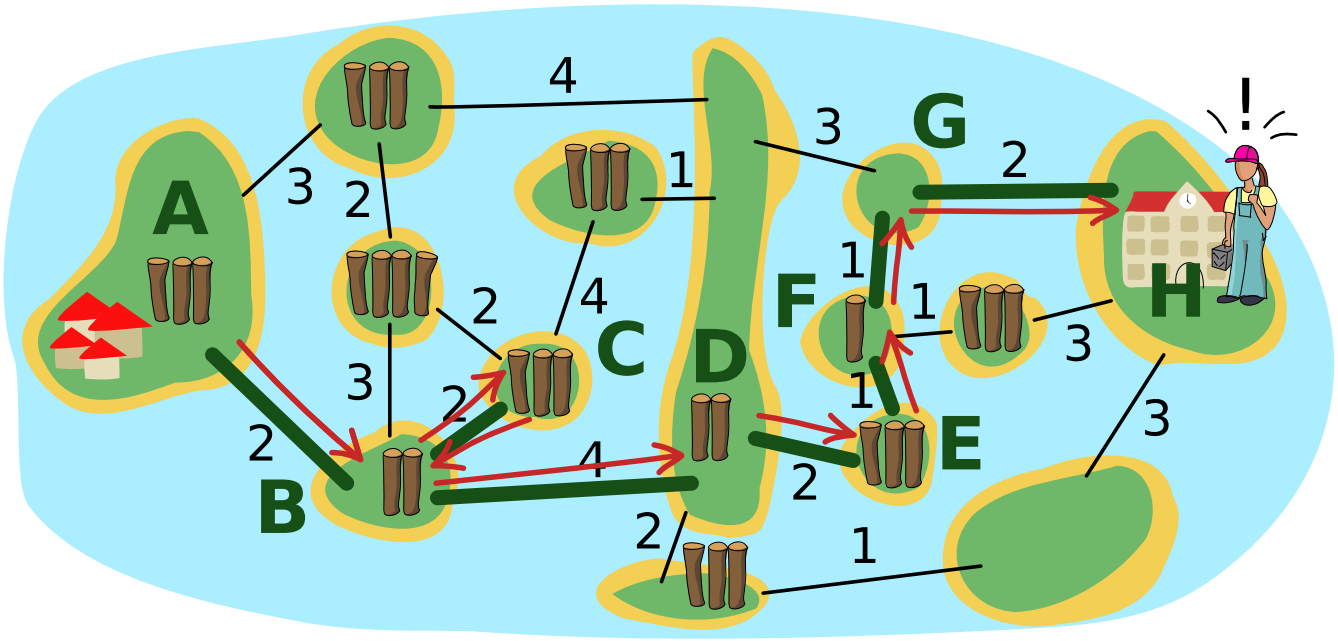
Bianca commence sur l'île de gauche. Son but est d'utiliser le moins de troncs possible.

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

 © Castor Informatique 2023, SSIE

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`

© Castor Informatique 2023, SSIE

9. Notation postfixe

Une expression mathématique est constituée :

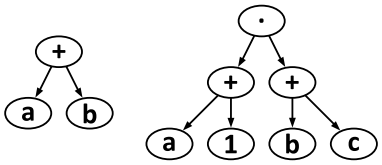
- d'un *opérateur* : $+$, $-$, $\cdot$ ou :
- et d'*opérandes* : des chiffres comme 1, 2, …, des lettres comme a, b, …, ou d'autres expressions comme $(1 + 2)$.

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 :

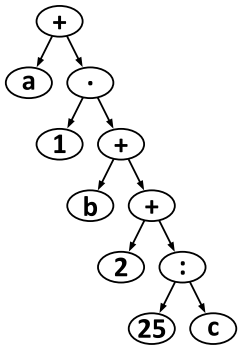


Notation postfixe : a b + a 1 + b c + ·

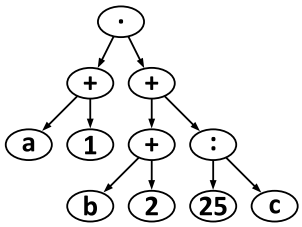
Voici la notation postfixe d'une autre expression :

a 1 + b 2 + · 25 c : +

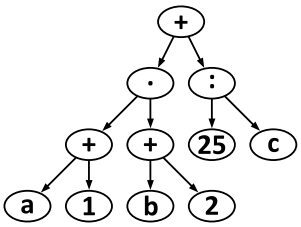
Quelle arborescence correspond à cette expression ?



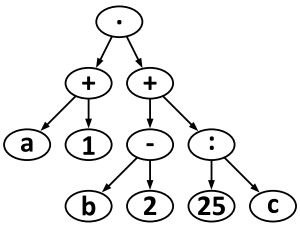
A)



B)

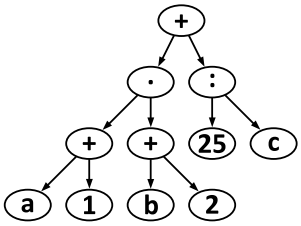


C)



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.

 © Castor Informatique 2023, SSIE

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`

© Castor Informatique 2023, SSIE

10. 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

 © Castor Informatique 2023, SSIE

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.

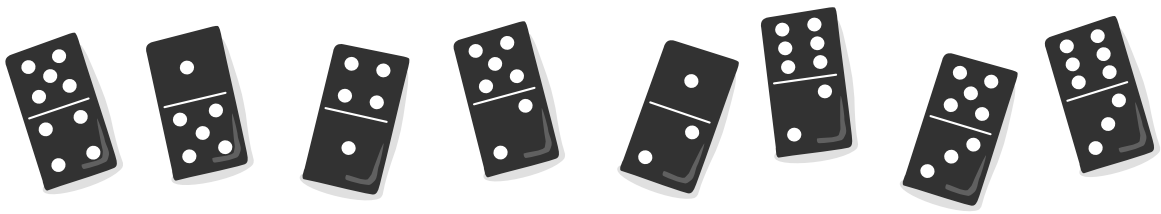
Mots clés et sites web

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

© Castor Informatique 2023, SSIE

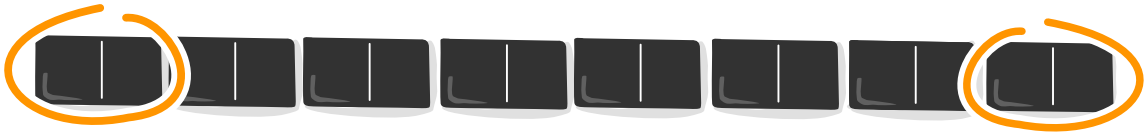
11. Dominos

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.

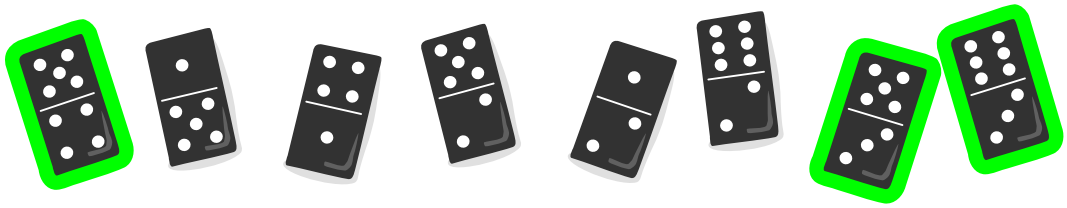
Il y a plusieurs manières d'aligner ces huit dominos, mais certaines pièces ne peuvent jamais se trouver au début ou à la fin de la ligne.



De quelles pièces s'agit-il ?

Solution

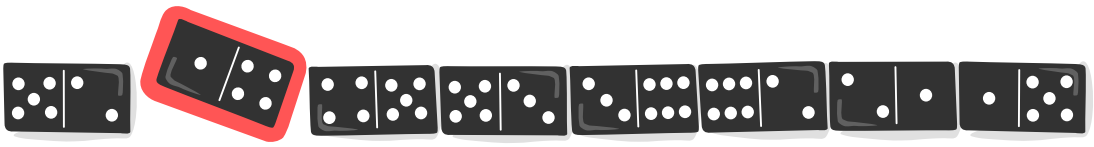
Trois des huits pièces ne peuvent pas être posées au début ou à la fin de la ligne :



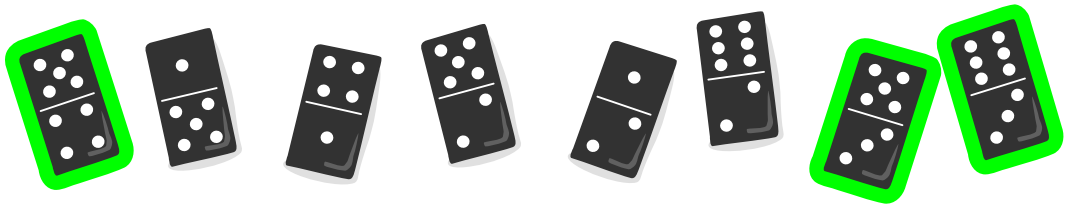
Pour résoudre l'exercice, on considère les nombres de points sur les 16 cases des dominos. Nous comptons combien de fois chaque nombre de points est présent et regardons si cette fréquence est paire ou impaire :

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.



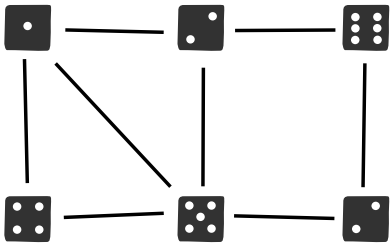
Comme il y a des cases présentes un nombre impair de fois parmi les huit dominos de cet exercice, ce sont eux qui doivent se trouver aux extrémités. Les dominos ayant deux cases avec fréquence paire ne peuvent donc pas être posés aux extrémités de la ligne. Ce sont les dominos suivants :



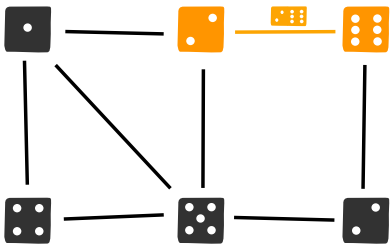
 © Castor Informatique 2023, SSIE

C'est de l'informatique !

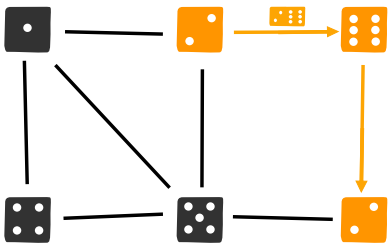
Il y a plusieurs possibilités de poser les dominos de cet exercice du Castor en une ligne correcte. Pour avoir une meilleure vue d'ensemble, les informaticiens et informaticiennes utilisent des *graphes* :



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

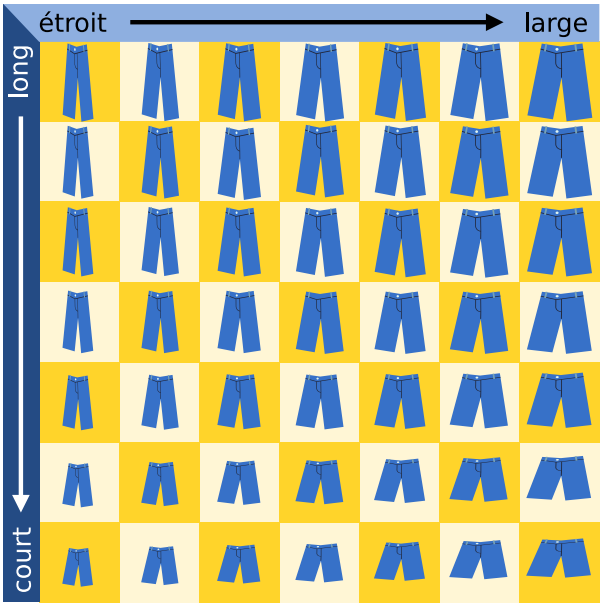
- Graphe : `https://fr.wikipedia.org/wiki/Graphe_(mathématiques_discrètes)`
- Nœud : `https://fr.wikipedia.org/wiki/Sommet_(théorie_des_graphes)`
- Arête : `https://fr.wikipedia.org/wiki/Arête_(théorie_des_graphes)`
- Chemin : `https://fr.wikipedia.org/wiki/Chemin_(théorie_des_graphes)`
- Chemin eulérien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

 © Castor Informatique 2023, SSIE

12. 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 ?

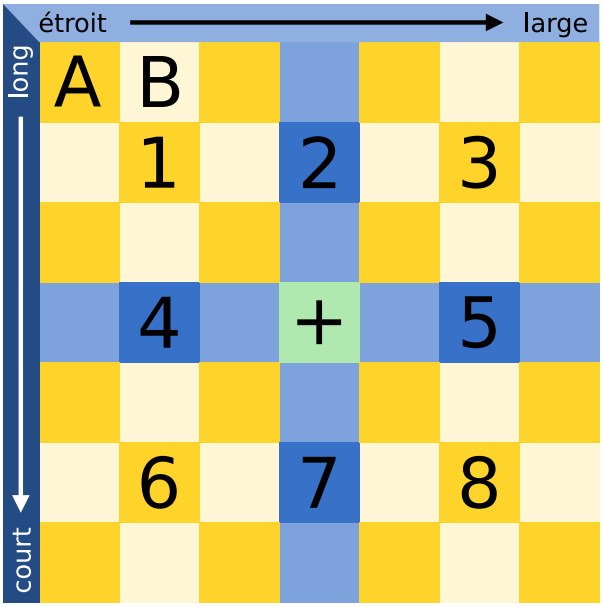
Solution

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.

 © Castor Informatique 2023, SSIE

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`

© Castor Informatique 2023, SSIE

13. 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.

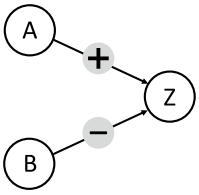
Lorsqu'un élément est

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

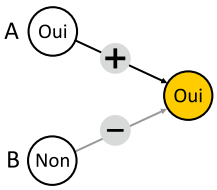
On peut régler chaque câble pour qu'un signal transmis devienne positif (+) ou négatif (–) pour l'élément de droite auquel il est relié. Un élément qui reçoit des signaux passe à l'état Oui s'il reçoit plus de signaux positifs que de signaux négatifs, et reste à l'état Non sinon.

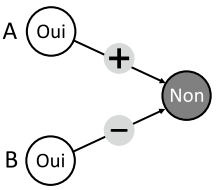
Anna fixe l'état de l'élément A et Ben l'état de l'élément B ; ce sont les entrées du détecteur.

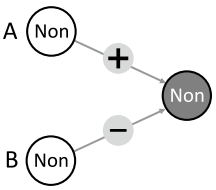
Anna et Ben commencent par construire cette machine :

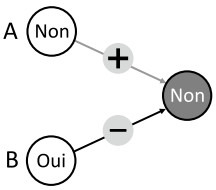


Ils remarquent que l'élément Z n'est dans l'état Oui que lorsque A est dans l'état Oui et B est dans l'état Non. Ce n'est pas ce qu'ils veulent.



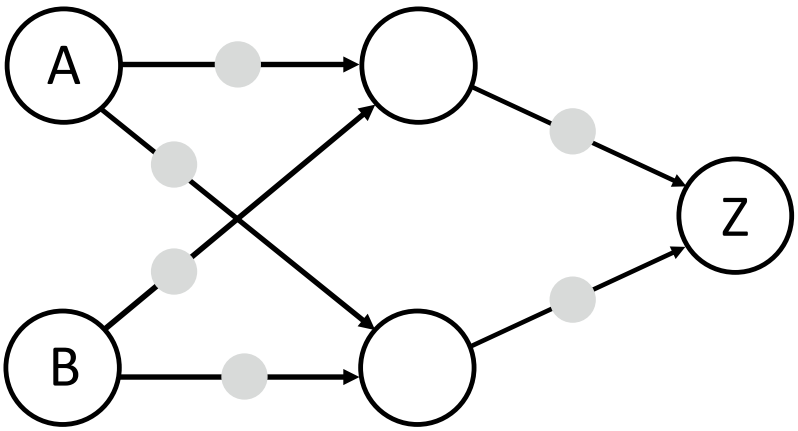






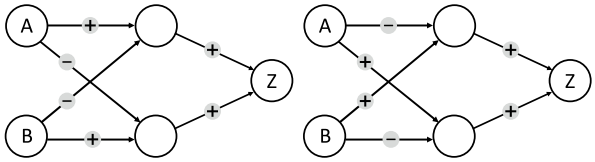
Anna et Ben construisent alors une plus grande machine (image ci-dessous) et sont sûrs qu'elle peut être un détecteur de conflit : l'état de Z ne doit être Oui que lorsque les état de A et B sont différents (Oui et Non ou Non et Oui). Sinon, Z doit être dans l'état Non. Il ne reste plus qu'à régler les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

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

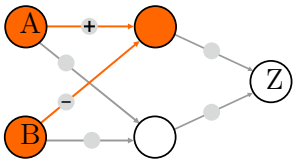


Solution

Les deux réponses suivantes sont justes :

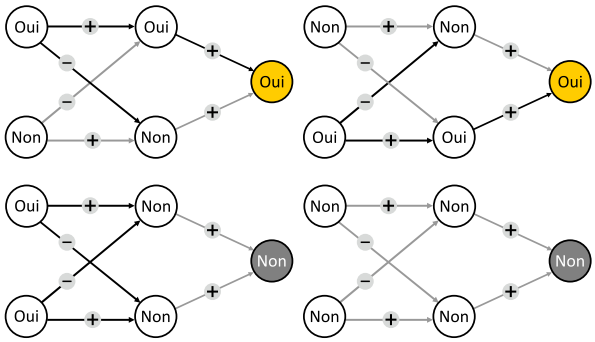


L'élement de sortie Z du détecteur de conflit doit être dans l'état Oui seulement en présence de deux entrées différentes (A = Oui et B = Non ainsi que A = Non et B = Oui). Z ne peut être en l'état Oui que s'il reçoit plus de signaux positifs que négatifs par ses deux câbles entrant. Au moins un des 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.



 © Castor Informatique 2023, SSIE

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`

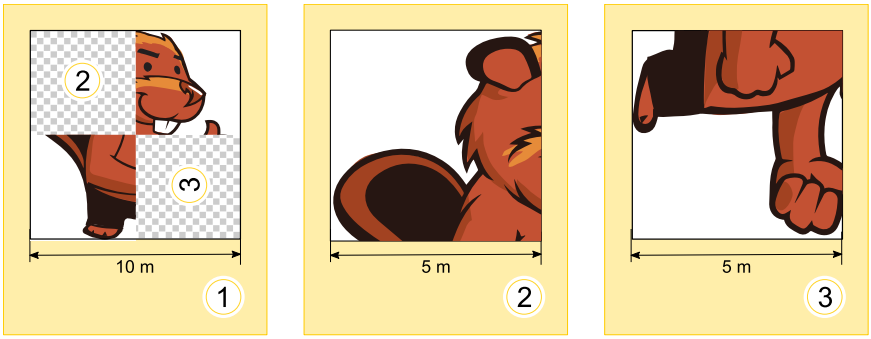
© Castor Informatique 2023, SSIE

14. Peinture récursive

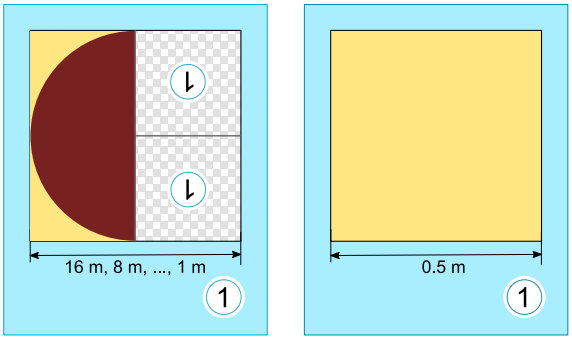
Tina et Ben aident à préparer une exposition temporaire au musée de l'informatique. Ils doivent peindre une image de 16 × 16 mètres sur le sol d'une salle d'exposition. L'artiste leur donne un set de cartes d'instruction de peinture avec des indications sur les éléments des images, leurs dimensions et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

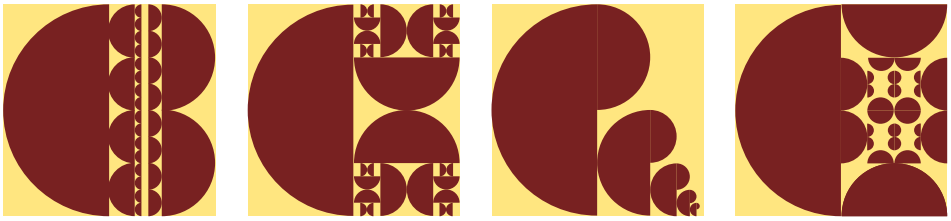


Tina et Ben reçoivent ces deux cartes pour l'exposition temporaire :



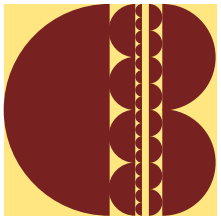
Ben fronce les sourcils. « Comment ça marche ? la carte de gauche réfère à elle-même, et en plus les deux cartes ont le même numéro ! » Tina rigole : « On va y arriver ! Commençons par la carte de gauche, la carte de droite nous dira plus tard quand nous devons arrêter de peindre. »

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`

 © Castor Informatique 2023, SSIE

15. 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`

 © Castor Informatique 2023, SSIE

A. Auteur·e·s des exercices

Eslam AbdElAal
Akram Ahmed
Nursultan Akhmetov
Khairul Anwar
James Atlas
Leonardo Barichello
Wilfried Baumann
Tim Bell
Leonardo Cavalcante
Zaheer Chothia
Eimear Colreavy
Raluca Constantinescu
Kris Coolsaet
Valentina Dagienė
Christian Datzko
Justina Dauksaite
Nora A. Escherle
Georgios Fesakis
Gerald Futschek
Hans-Werner Hein
Tracy Henderson
Thomas Ioannou
Filiz Kalelioğlu
Merel Kämper
Gohar Khachatryan
Jihye Kim
Víctor Koleszar
Taina Lehtimäki
Michael Weigend
Dario Malchiodi
Yong Mao
Anna Morpurgo
Jalil Nedaeepour
Özgür Özdemir
Elsa Pellet
Zsuzsa Pluhár
Wolfgang Pohl
Ilya Posov
Sergey Pozdniakov
JP Pretti
Omar Colon Reyes
Chris Roffey
Karima Sayeh
Margareta Schlüter
Eljakim Schrijvers
Vipul Shah
Rostyslav Shpakovych
Jacqueline Staub
Alieke Stijf
Ahto Truu
Laura Ungureanu
Jiří Vaníček

Michael Weigend

Kyra Willekes

© Castor Informatique 2023, SSIE

B. Partenaires académiques

`http://www.abz.inf.ethz.ch/`
Ausbildungs- und Beratungszentrum für Informatikunterricht der ETH Zürich.

`http://www.hepl.ch/`
Haute école pédagogique du canton de Vaud

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

C. Sponsoring

http://www.haslerstiftung.ch/

Standortförderung beim Amt für Wirtschaft und Arbeit Kanton Zürich

http://www.ubs.com/

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

i-factory (Musée des transports, Lucerne)

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

© Castor Informatique 2023, SSIE

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.

© Castor Informatique 2023, SSIE