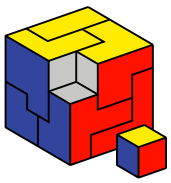
Prologin 2016



Concours national d'informatique

Épreuve écrite d'algorithmique
Lille

27/02/2016

Résolution de l'intégramme

1 Préambule

Bienvenue à **Prologin**. Ce sujet est l'épreuve écrite d'algorithmique et constitue la première des trois parties de votre épreuve régionale. Sa durée est de 3 heures. Par la suite, vous passerez un entretien (20 minutes) et une épreuve de programmation sur machine (4 heures).

Conseils

— Lisez bien tout le sujet avant de commencer.
— **Soignez la présentation** de votre copie.
— N'hésitez pas à poser des questions.
— Si vous avez fini en avance, relisez bien.
— N'oubliez pas de passer une bonne journée.

Remarques

— Le barème est donné à titre indicatif uniquement.
— Indiquez lisiblement vos nom et prénom, la ville où vous passez l'épreuve et la date en haut de votre copie.
— Tous les langages sont autorisés, veuillez néanmoins préciser celui que vous utilisez.
— Ce sont des humains qui lisent vos copies : laissez une marge, aérez votre code, ajoutez des commentaires (**seulement** lorsqu'ils sont nécessaires) et évitez au maximum les fautes d'orthographe.
— Le barème récompense les algorithmes les plus efficaces : écrivez des fonctions qui trouvent la solution le plus rapidement possible.
— Si vous trouvez le sujet trop simple, relisez-le, réfléchissez bien, puis dites-le-nous, nous pouvons ajouter des questions plus difficiles.

2 Sujet

Introduction

Le but du sujet est de résoudre un très célèbre casse-tête logique : l'*intégramme*, ou *zebra puzzle* en anglais. Il existe des relations entre des éléments, et à partir d'un certain nombre d'indices il s'agit de retrouver l'intégralité des relations.

Par exemple on dispose de personnages, disons Aramis, Princesse et Gribouille, chaque personnage aime faire une activité parmi jouer, dormir et manger. Et chaque personnage utilise un outil pour son activité : la salade, le coussin ou la balle. Les outils et activités de chacun sont différents.

On dispose des indices suivants :

— Princesse aime manger
— On ne joue pas avec le coussin
— L'outil d'Aramis n'est ni la salade ni le coussin
— On n'utilise pas la salade pour dormir
— L'outil de Gribouille est le coussin

On peut représenter l'énigme comme une grille de la manière suivante :

		Qui			Quoi		
		Aramis	Princesse	Gribouille	Jouer	Dormir	Manger
Avec quoi	Salade						
	Coussin				X		
	Balle						
Quoi	Jouer		X				
	Dormir		X				
	Manger	X	O	X			

La grille se remplit comme suit :

— On place un rond dans une case si les propriétés en ligne et en colonne de la case correspondent à une même personne
— On place une croix si ce n'est pas le cas
— Tant qu'on est pas certain on laisse la case vide

Par exemple, Princesse aime manger donc on place un rond à l'intersection de Princesse et Manger.

On ne joue pas avec le coussin, donc on place une croix à l'intersection de Jouer et Coussin.

Résoudre l'énigme revient donc à compléter la grille.

Pour trouver la valeur d'une case il va falloir la déduire de celles que l'on sait.

Par exemple, sachant que Princesse aime manger, on sait que l'activité d'Aramis ou Gribouille n'est pas de manger. On en déduit également que Princesse ne pratique pas les activités jouer et dormir. On place alors des croix dans les cases correspondantes.

Voici la grille une fois résolue :

		Qui			Activité		
		Aramis	Princesse	Gribouille	Jouer	Dormir	Manger
Outil	Salade	X	O	X	X	X	O
	Coussin	X	X	O	X	O	X
	Balle	O	X	X	O	X	X
Activité	Jouer	O	X	X			
	Dormir	X	X	O			
	Manger	X	O	X			

Partie 1

Dans cette première partie nous allons commencer par faire un algorithme qui modélise le problème et comprend les contraintes en entrée.

Pour commencer, il faut modéliser la grille. Pour simplifier on représentera la grille d'une manière à ce que les attributs des personnages soient les indices d'un tableau et chaque élément du tableau contient la table de vérité d'appartenance pour chacun des personnage.

Plus simplement `grille[i]` représentera la table pour l'attribut i et `grille[i][j]` est un booléen qui vaut `Vrai` si le personnage j possède l'attribut i. Ainsi seules les relations directes entre les personnages et leurs attributs sont stockés.

Les tables seront toutes initialisées à `Vrai`, ce qui signifie que chaque assignation d'un attribut à un personage est encore possible.

La grille de l'exemple initialisée serait donc représentée comme suit :

	Aramis	Princesse	Gribouille
Salade	O	O	O
Coussin	O	O	O
Balle	O	O	O
Jouer	O	O	O
Dormir	O	O	O
Manger	O	O	O

Question 1 (1 point)

> Proposer une structure de données pour représenter `grille` que vous allez utiliser tout au long du sujet. Cette structure sera supposée définie globalement.
>
> Écrire une fonction `initialiser_grille` qui prend comme paramètres deux entiers n et m et qui initialise `grille` pour une énigme consistant de n personnages et m groupes d'attributs. Votre fonction ne devra rien renvoyer.

Par rapport au modèle de la grille qu'on a établit, on peut constater qu'il existe plusieurs types de contraintes, certaines lient directement un attribut à un personnage (ex : "Princesse aime manger", "L'outil d'Aramis n'est ni la salade ni le coussin"), qu'on appelera *contraintes unaires* (car n'utilisant qu'un seul attribut), tandis que d'autres lient des attributs entre eux (ex : "On ne joue pas avec le coussin"), qu'on appelera *contraintes binaires* (car utilisant deux attributs).

Formellement les contraintes seront de l'un des types suivants.

— *contrainte unaires* : L'attribut i appartient/n'appartient pas à la personne j.
— *contrainte binaire* : Les attributs i_1 et i_2 appartiennent/n'appartiennent pas à la même personne

Commençons avec les contraintes unaires. Comme toute la grille est initialisée à `Vrai` il va falloir traduire les contraintes de manière à ajouter des `Faux`.

On rappelle au passage que si un attribut i appartient à une personne j, alors il ne peut pas appartenir à une autre personne.

Question 2 (2 points)

> Écrire une fonction `ajouter_contrainte_unaire` qui prend en entrée deux entiers i et j ainsi qu'un booléen b et modifie `grille` en prenant en compte la contrainte que le personnage j possède l'attribut i avec la valeur de vérité b. Attention vous n'avez le droit ici de modifier qu'uniquement la table de vérité de l'attribut i [1].

Attaquons-nous maintenant aux contraintes binaires. Comme elles ne peuvent pas être stockées directement dans notre structure de grille, il va falloir les placer dans une structure à part dont on se servira pour résoudre la grille.

Question 3 (1 point)

> Proposez une structure qu'on appelera `contraintes` qui vous servira dans la suite (elle sera supposée définie globalement). Il faudra que votre structure permette de rajouter une contrainte et que, pour deux attributs i et j, il soit possible de tester s'il existe une contrainte entre i et j. Si elle existe il faut qu'il soit possible de dire si c'est une contrainte `Vrai` (qui doit être satisfaite) ou `Faux` (qui doit ne pas l'être).

Question 4 (1 point)

> Écrire une fonction `ajouter_contrainte_binaire` qui prend en entrée deux attributs i et j ainsi qu'un booléen b et qui ajoute à `contraintes` la contrainte entre i et j de valeur de vérité b.

Il existe également des contraintes binaires implicites, car les attributs d'un même sous-ensemble d'attributs (ex : "outils" ou "activités") doivent appartenir à des personnes différentes. On supposera que l'indexation des attributs fait que les sous-ensembles sont indexés les uns à la suite des autres. Les n premiers attributs appartiennent au même sous-ensemble, de même pour les n suivants, etc..

Question 5 (4 points)

> Écrire une fonction `ajouter_contraintes_implicites` qui prend en entrée deux entiers n et m, le nombre de personnes et de groupes d'attributs, et qui ajoute ces contraintes dans `contraintes`.

1. les déductions viendront ensuite, ne soyez pas trop pressés ;p

Pour l'exemple, on indexe les personnes et les attributs comme suit :

— Aramis → 0
— Princesse → 1
— Gribouille → 2
— Salade → 0
— Coussin → 1
— Balle → 2
— Jouer → 3
— Dormir → 4
— Manger → 5

Question 6 (2 points)

> Écrire une fonction `initialisation_exemple` qui utilise les fonctions précédentes pour initialiser toutes les structures et ajoute l'ensemble des contraintes pour l'exemple donné.

Partie 2

Enfin on arrive à la résolution !

Dans cette partie on suppose que la grille qui a été initialisée est stockée sous le nom `grille_initialisée`.

Question 7 (3 points)

> Écrire une fonction `test` qui prend en entrée une grille et qui renvoie `Vrai` si et seulement si la grille respecte les contraintes stockées dans `contraintes`

Question 8 (4 points)

> Écrire une fonction `résoudre` qui prend en entrée une grille et qui, si la grille est achevée l'affiche si elle respecte les contraintes, sinon choisit un des attributs pour lesquels il reste plusieurs cases à `Vrai` dans la table de vérité et qui pour chaque case à `Vrai` s'exécute récursivement en prenant la décision que ce sera définitivement celle là qui sera vraie.

Question 9 (1 point)

> Quel appel reste-t-il à faire pour résoudre l'énigme ?
> Que pensez-vous de l'efficacité d'une telle résolution ?

La fonction que vous venez d'écrire permet en effet de résoudre l'énigme, bravo ! Cependant c'est un peu dommage de n'utiliser les contraintes qu'à la fin alors qu'on s'imagine bien qu'il doit être possible de déduire des choses sur la grille avant de se lancer dans une bête énumération des solutions.

Question 10 (3 points)

Écrire une fonction `est_possible` qui prend en entrée deux attributs i_1 et i_2 et une personne j et qui renvoie `Vrai` si et seulement si il existe une personne k telle que si j possède l'attribut i_1 alors l'attribut i_2 peut être possédé par k en respectant les contraintes.

Question 11 (5 points)

Écrire une fonction `est_modifiable` qui prend en entrée un attribut i et qui, pour chaque personne j telle que la case (i, j) de la grille est encore à `Vrai`, vérifie si une telle assignation à `Vrai` ne vas pas empêcher, sous les contraintes du problème, de trouver une assignation pour un autre attribut. Si c'est le cas elle modifie la valeur de la case. De plus `est_modifiable` renverra `Vrai` si et seulement si une modification de la grille a été effectuée.

Par exemple, avec l'état de la grille suivant, l'attribution `Vrai` de *dormir* à Princesse empêche à *manger* de trouver une attribution. En effet la grille permet déjà de dire que Aramis et Gribouille ne font pas l'activité *manger*. De plus *manger* et *dormir* ne peuvent pas être attribués à la même personne (contrainte implicite). Donc on sait que Princesse ne peut pas être en train de *manger*.

	Aramis	Princesse	Gribouille
Salade	O	O	O
Coussin	O	O	O
Balle	O	O	O
Jouer	O	O	O
Dormir	O	O	O
Manger	X	O	X

Question 12 (2 points)

Écrire une fonction `filtrage` qui prend en entrée une grille et qui modifie la grille à l'aide des fonctions précédentes jusqu'à ce que ce ne soit plus possible. Elle renverra la grille modifiée.

Question 13 (1 point)

Reprendre l'écriture de la fonction `résoudre` de la question 8 en filtrant la grille à chaque étape.

Partie bonus

On pourrait encore améliorer cette solution en choisissant intélligemment la prochaine variable dont on décide l'attribution.

Question bonus 14 (5 points)

> Écrire une fonction `choix_variable` qui prend en entrée une grille et renvoie un attribut choisit selon les trois critères suivants :
>
> 1. plus grand nombre de valeurs à `Vrai` dans la table de vérité
> 2. plus grand nombre de contraintes binaires dans lesquelles il rentre en jeu
> 3. indice le plus petit
>
> Le choix s'effectue dans l'ordre lexicographique (si plusieurs vérifient 1. alors 2. les départage et sinon 3.)

Question bonus 15 (1 point)

> Écrire la dernière version de `résoudre` qui prend en compte ce choix.

Question bonus 16 (1 point)

> Qui est le présupposé auteur de ce casse-tête ?

Question bonus 17 (4 points)

> L'énigme devient plus dure si on rajoute des contraintes de comparaisons temporelles. Saurez-vous résoudre l'énigme suivante ?
>
> — Edwin a passé son audition avant la personne classée 5eme et après le jazz, que n'a pas chanté Camille.
> — La salsa, chantée ni par Camille ni par Leeloo, a été mieux classée que Maya et moins bien que la personne auditionnée mercredi.
> — L'audition de lundi a été classée après celle de Zacharie et avant le rock.
> — Le chant classique a eu lieu après celui classé 2eme et 1 jour avant le rock.
> — Maya a chante après le rap et avant la personne classée 3eme, qui n'est pas Zacharie.
>
> Vous pourrez vous aider de la grille suivante.

		Classement					Style					Jour				
		1er	2e	3e	4e	5e	Cl	Ja	Ra	Sa	Ro	Lu	Ma	Me	Je	Ve
Chanteur	Zacharie															
	Camille															
	Edwin															
	Leeloo															
	Maya															
Jour	Lundi															
	Mardi															
	Mercredi															
	Jeudi															
	Vendredi															
Style	Classique															
	Jazz															
	Rap															
	Salsa															
	Rock															

Fin

Le sujet comporte 7 pages (sans compter la page de garde) et 17 questions, parmi lesquelles 4 questions bonus. Les questions normales sont notées sur 30 points, et les questions bonus rapportent au total 11 points, plus 1 points de présentation.