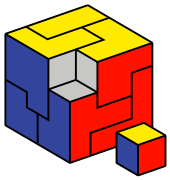
Prologin 2016



Concours national d'informatique

Épreuve écrite d'algorithmique
Lyon I

Samedi 6 février 2016

Keep coding and nobody explodes

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

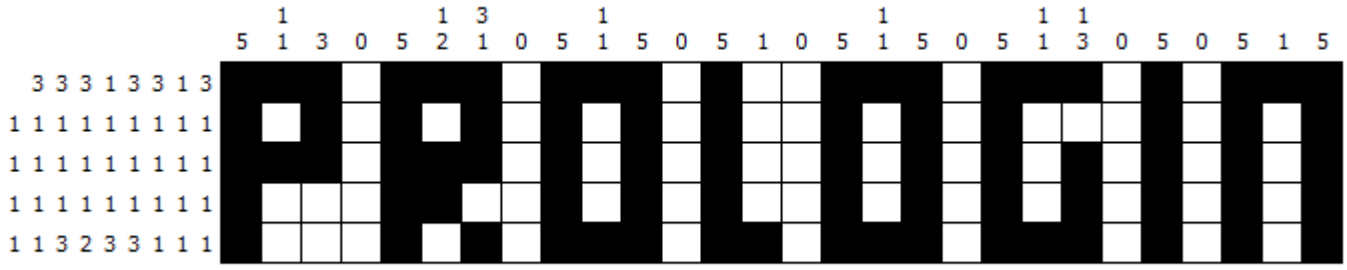
Introduction

Agent 404, nous avons besoin de vous pour une ultime mission. Nos espionnes Alice et Carole nous ont informés que des explosifs ont été introduits dans notre quartier général. Ils exploseront dans exactement 3 heures et 15 minutes.

Votre mission [1] sera de retrouver la localisation de ces mines avant qu'elles n'explosent. Vous avez toute notre confiance. L'avenir de notre organisation repose sur vos épaules. Bonne chance, Bob.

Ces explosifs ont été introduits par Ève, qui a dessiné des plans indiquant leurs positions. Grâce à nos espionnes [2], nous avons récupéré ces plans, mais ils sont évidemment codés. Cependant, nous avons fini par découvrir le système utilisé par Ève pour chiffrer ces plans. Ce dernier consiste à indiquer pour chaque ligne et chaque colonne le nombre, la taille, et la position relative des blocs contigus de mines. Dans les plans que l'on va obtenir après déchiffrage, une case contenant une mine est coloriée en noir, tandis qu'une case sans mine est laissée blanche. [3]

FIGURE 1 – Sur cet exemple de plan, le codage des colonnes indiqué en haut et le codage des lignes est indiqué à gauche



Par exemple : `[2,3]` signifie qu'il y d'abord éventuellement des cases blanches, puis un bloc de 2 cases noires, puis un nombre indéterminé et non nul de cases blanches, puis un bloc de 3 cases noires, puis encore éventuellement des cases blanches. Si la longueur d'une ligne est 8, les possibilités de lignes dont le code est `[2,3]` sont les suivantes :

FIGURE 2 – Possibilités de décodage pour `[2,3]`

1. ~~Si vous l'acceptez.~~
2. Alice et Carole, vous vous souvenez ?
3. En fait, Ève est une cruciverbiste. Si vous ne connaissez pas le sens de ce mot, demandez-le lors de l'entretien.

Partie I : encodage

Question 1 (3 points)

> Quelles structures de données utiliserez-vous pour représenter :
> — un plan formé de cases en noir et blanc ?
> — une ligne ou une colonne d'une telle image ?
> — les informations de taille des blocs noirs sur une rangée (ligne ou colonne) ?

Question 2 (2 points)

> Écrivez une fonction `encode_ligne` qui prend en entrée une ligne et renvoie en sortie le code de cette ligne, c'est-à-dire les tailles des blocs sur la ligne.

Question 3 (2 points)

> Écrivez une fonction `encode_image` qui calcule les tailles des blocs sur chaque ligne et chaque colonne d'une image fournie en entrée.

Partie II : décodage

Ève revient d'une semaine de vacances et trouve un ensemble de codages réalisés par ses stagiaires. En vérifiant le travail de ces derniers, elle se rend compte de deux problèmes : un des codages produits ne peut correspondre à aucune image, et un autre peut au contraire être décodés en plusieurs images différentes.

Question 4 (2 points)

> Mettez-vous dans la peau d'un stagiaire incompétent. Donnez un exemple de codage sans décodage possible et un exemple de codage avec plusieurs décodages valables.

Nous allons tout d'abord nous intéresser au décodage d'une seule ligne. À partir d'une suite de taille de blocs (`[2,3]` par exemple), et d'un entier n (8 par exemple) correspondant à la taille de la ligne à remplir, on cherche à calculer l'ensemble des décodages possibles. Par exemple, pour le code `[2,3]` et une ligne de taille 8, les possibilités sont (l'ordre n'ayant pas d'importance) : `[1,1,0,1,1,1,0,0]`, `[1,1,0,0,1,1,1,0]`, `[1,1,0,0,0,1,1,1]`, `[0,1,1,0,1,1,1,0]`, `[0,1,1,0,0,1,1,1]`, `[0,0,1,1,0,1,1,1]` (1 indiquant une case noire, et 0 une case blanche). Vous pouvez vérifier que cela correspond bien aux possibilités montrées en figure 2.

Question 5 (6 points)

> (a) Montrez que si le code d'une ligne de taille n est une liste `t` de taille au moins deux, et que le premier bloc est placé à la position i, alors placer les autres blocs revient à décoder une ligne de taille $n - i - \texttt{t}[0] - 1$.
> (Observez par exemple sur la moitié gauche de la figure 2 que pour décoder `[2,3]`, une fois le bloc de taille 2 placé tout à gauche, les emplacements possibles pour le bloc de taille 3 sont limités aux $5 = 8 - 0 - 2 - 1$ cases à droite.)
>
> (b) Déduisez-en un algorithme (par exemple récursif) pour obtenir la liste de tous les décodages possibles d'une ligne ou d'une colonne.

Question 6 (3 points)

> À partir de la question précédente, écrire une fonction qui décode une image en testant toutes les lignes possibles tout en vérifiant la compatibilité avec les codes des colonnes. *(Si vous n'avez pas répondu à la question précédente, vous pouvez quand même admettre que vous disposez d'une fonction pour générer tous les décodages possibles d'une ligne ou d'une colonne.)*

Pour déchiffrer le code d'Ève plus efficacement, nous allons utiliser une méthode de *backtracking*. L'idée est de choisir, pour la première ligne, un décodage possible pour celle-ci, puis de continuer avec la suivante et ainsi de suite… Mais plutôt que d'attendre d'avoir fait un choix pour *toutes* les lignes avant de le vérifier par rapport au code des colonnes, on peut se rendre compte plus tôt qu'on est sur une mauvaise piste. Autrement dit, quelle que soit la façon dont on s'y prend, on ne pourra pas colorier le reste des lignes de manière à obtenir un décodage valide. Dans ce cas, on revient en arrière immédiatement pour changer le dernier choix de décodage de ligne.

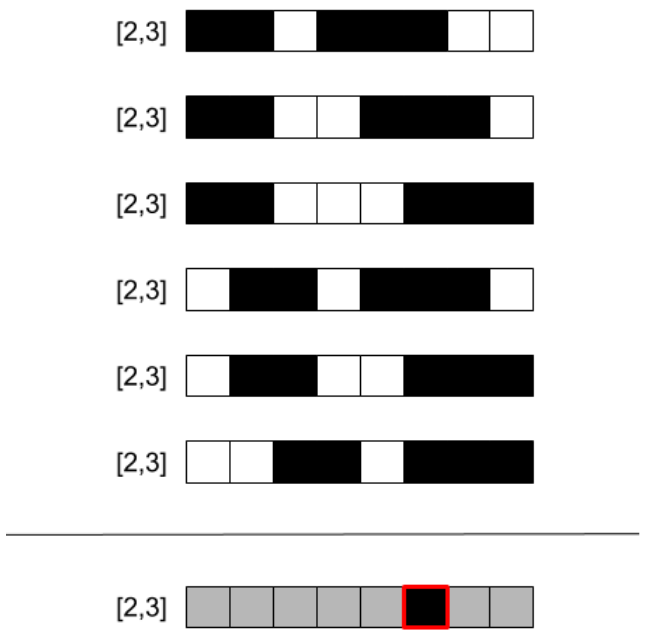
Question 7 (6 points)

> (a) Proposez un critère permettant de détecter de telles incohérences futures relativement aux colonnes.
>
> (b) Implémentez la résolution par backtracking, en suivant l'explication ci-dessus.

Partie III : déductions

Le temps presse, les mines vont bientôt exploser ! Pour accélérer encore le décodage, nous allons chercher à remplir certaines cases avant d'utiliser le backtracking, afin de réduire le nombre de possibilités à tester.

FIGURE 3 – Exemple de combinaison sur une ligne de taille 8 et code [2,3]



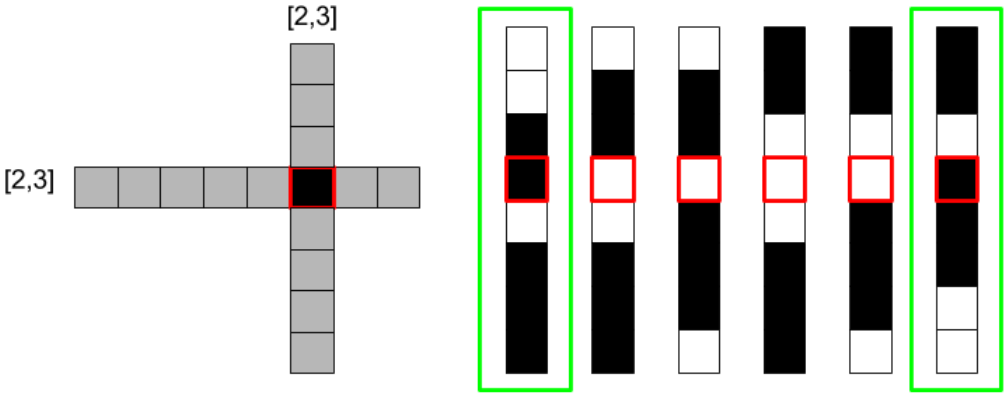
Pour cela, nous allons effectuer ce que nous appelerons des *combinaisons* et des *exclusions*.

Le principe des combinaisons est le suivant. Considérons toutes les solutions possibles pour une ligne ou colonne donnée. Si une case est présente de la même couleur sur toutes les solutions possibles, alors elle sera forcément de cette couleur sur la grille finale.

Sur l'exemple de la figure 3, nous savons que la 6ème case de la ligne contient forcément une mine, car cette mine est présente dans toutes les combinaisons possibles vérifiant les conditions.

Quant aux exclusions, elles consistent à utiliser les informations obtenues par combinaisons pour éliminer certaines possibilités de lignes ou de colonnes. Reprenons l'exemple précédent. Après avoir effectué une combinaison sur la ligne numéro 4, nous avons réussi à déduire la position d'une mine. On peut alors se servir de cette information sur les colonnes, et éliminer les solutions de la colonne qui ne contiennent pas de mines à l'emplacement trouvé.

FIGURE 4 – Exemple d'exclusion



Ainsi, les deux seules décodages possibles pour les colonnes sur cet exemple sont, après exclusion : `[0,0,1,1,0,1,1,1]` et `[1,1,0,1,1,1,0,0]`.

Question 8 (2 points)

> (a) Implémentez les combinaisons en écrivant une fonction, qui, à partir des décodages possibles d'une ligne, détermine quelles cases sont sont noires sur tous ceux-ci, et de même pour les cases blanches. Votre fonction devra renvoyer un tableau indiquant pour chaque indice si la case sur cette colonne est forcément noire, forcément blanche, ou de couleur indéterminée.
>
> (b) Écrivez une fonction qui, en effectuant cette opération sur les lignes et les colonnes, préremplit l'image.

Question 9 (2 points)

> Implémentez les exclusions en écrivant une fonction qui, grâce à un plan partiellement rempli, filtre la liste des décodages possibles d'une ligne et ne garde que ceux qui sont compatibles avec le plan partiel.

Question 10 (4 points)

> Écrivez un programme qui déchiffre le code d'Ève en effectuant d'abord des combinaisons et des exclusions sur toutes les lignes et les colonnes, jusqu'à ne plus obtenir d'informations supplémentaires, et enchaîne ensuite sur un backtracking pour finir de remplir l'image.

L'objet des deux questions suivantes est de parvenir à faire cette étape de préremplissage plus efficacement.

Question 11 (8 points)

> On souhaite ici connaître toutes les informations (cases forcément blanches ou noires) qu'on peut déduire sur une seule ligne à partir de son code par combinaison, sans générer explicitement tous les décodages possibles.
>
> (a) En altérant le code de la question 5(b), proposez une solution simple en *espace* polynomial [4].
>
> (b) Donnez un algorithme qui résout ce problème en *temps* polynomial.

Question 12 (10 points)

> (a) Décrivez un algorithme en temps polynomial qui, étant donnée une ligne dont les cases sont noires, blanches ou de couleur indéterminée, ainsi que son code, en déduit les informations supplémentaires qu'on aurait pu obtenir par une exclusion suivie d'une combinaison.
>
> (b) Appliquez ceci pour effectuer le préremplissage du plan avec toutes les informations déductibles par combinaison et exclusion, sans avoir à générer explicitement les décodages possibles.

Partie bonus

Question bonus 13 (4 points)

> (a) Montrez que les différents décodages possibles d'une ligne de longueur n et de code `t` peuvent être représentés par les lignes de longueur n' dont exactement k' cases sont noires, pour des valeurs de n' et k' bien choisies.
>
> (b) Donnez une formule pour le nombre de décodages possibles d'une ligne.

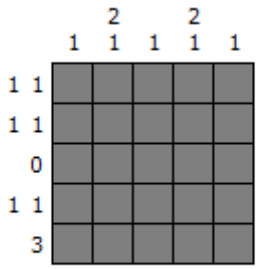
Question bonus 14 (4 points)

> Générer et stocker tous les décodages possibles de toutes les lignes est coûteux en mémoire. Comment mettre en œuvre le backtracking en évitant ce problème ?

4. Si vous ne connaissez pas le sens de ce mot dans ce contexte, cette fois-ci, n'attendez pas l'entretien pour le demander, faites-le durant l'épreuve !

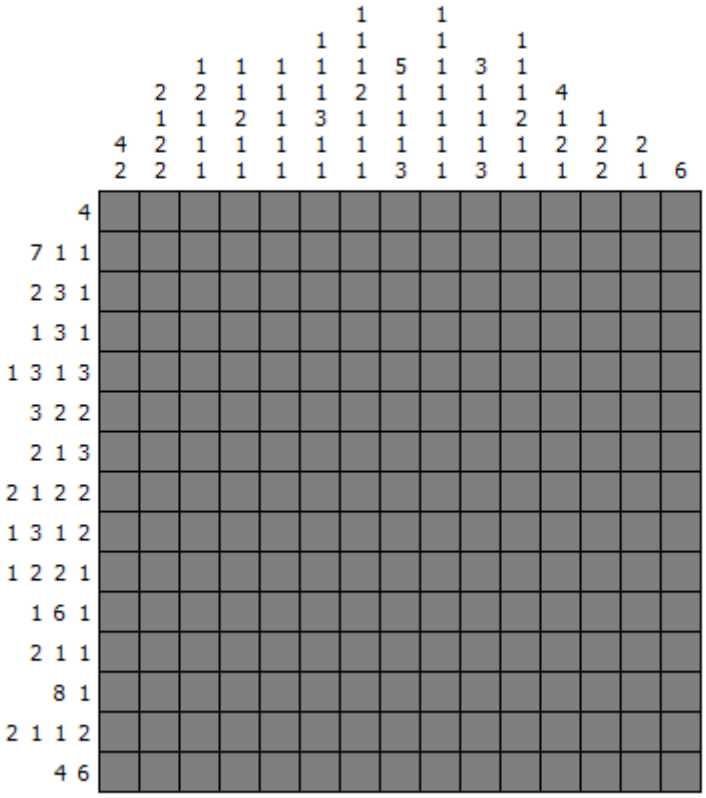
Question bonus 15 (1 point)

Déchiffrez le code suivant :



Question bonus 16 (1 point)

Vous en voulez encore ? Voici :



FIN [5]

Le sujet comporte 7 pages (sans compter la page de garde) et 16 questions, parmi lesquelles 4 questions bonus. Les questions normales sont notées sur 50 points, et les questions bonus rapportent au total 10 points, plus 1 point de présentation.

5. Félicitations ! Vous avez sauvé le monde