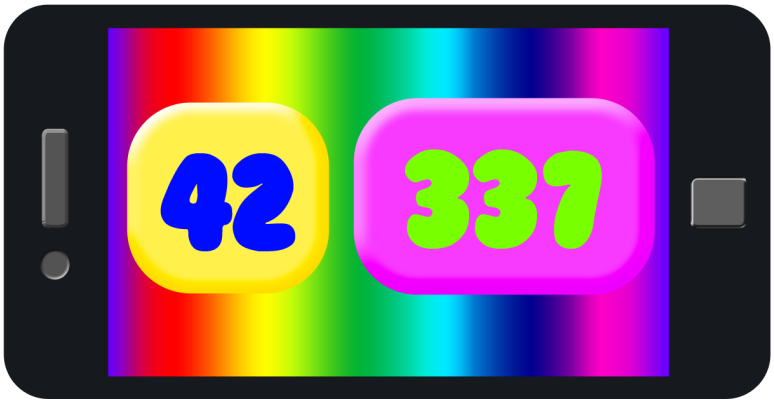
Prologin 2015

# Concours national d'informatique

## Épreuve écrite d'algorithmique
## Paris I

Dimanche 8 février 2015

# Coprime Crush




Interface utilisateur approuvée par Rainbow Dash

## 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 Coprime Crush est la nouvelle application sur smartphone faisant fureur. Le principe est simple : on part du couple de nombres $(1, 1)$ et on peut obtenir d'autres couples $(n, m)$ en appuyant sur l'écran. (Sur l'image de la page précédente, $n = 42$ et $m = 337$.) Le jeu [1] étant le reflet postmoderne de notre époque nihiliste et $\mathbb{N}$ étant [2] infini, il ne s'arrête jamais, il n'y a pas d'objectif précis à accomplir.

Les deux coups possibles consistent à appuyer sur un bouton pour faire augmenter son contenu :

- appuyer sur le bouton gauche fait passer de $(n, m)$ à $(n + m, m)$ ;
- appuyer sur la bouton droit fait passer de $(n, m)$ à $(n, m + n)$.

Ainsi, si on appuyait à droite sur l'exemple, on passerait de $(42, 337)$ à $(42, 379)$.

### Précisions

Tout au cours du sujet, on emploiera le vocabulaire suivant :

- un *coup* est soit « appuyer à gauche », soit « appuyer à droite » ;
- une *position* est un couple de nombres que l'on peut obtenir en exécutant 0, 1 ou plusieurs coups en partant de la *position initiale* : $(1, 1)$.

Pour simplifier les choses, vous pouvez étendre votre langage de programmation avec un type de données pour les couples de nombres entiers, vous permettant les manipulations suivantes :

- construire un couple à partir de deux nombres ;
- obtenir la valeur à gauche et la valeur à droite d'un couple ;
- prendre des couples en argument dans des fonctions ;
- écrire des fonctions qui renvoient des couples.

On se servira des couples d'entiers pour représenter des positions.

## Partie I : Jouons !

**Question 1** (2 points)

> (a) Préciser comment représenter un coup dans votre langage de programmation.
>
> (b) Écrire une fonction `executer_coup` dont :
> - Les arguments sont :
>  - le coup que l'utilisateur a sélectionné, et
>  - la position actuelle,
> - La valeur de retour est la position sur laquelle on tombe après avoir effectué ce coup.

**Question 2** (5 points)

> Écrire une fonction `executer_coups` (remarquez le pluriel) qui prend une suite de coups (par exemple sous forme de tableau) et renvoie le résultat obtenu à partir de la position initiale $(1, 1)$.

1. Toute cette phrase s'applique aussi au Jeu ; d'ailleurs, vous venez de perdre.
2. Mais $\mathbb{N}$ existe-t-il ? « Dieu a fait les nombres entiers, tout le reste est l'œuvre de l'homme », y compris l'*ensemble* des entiers naturels !

## Partie II : Analyse rétrograde

**Question 3** (5 points)

> Expliquer pourquoi le dernier coup joué avant d'atteindre la position $(n, m)$ est forcément :
> – gauche, si $n > m$
> – droite, si $n < m$

On admet qu'au cours d'une partie, les deux cases ne peuvent contenir des nombres égaux que lors de la position initiale ($n = 1$ et $m = 1$) ; les deux cas ci-dessus recouvrent donc toutes les possibilités, puisqu'il n'y a pas de coup joué avant la position initiale.

**Question 4** (10 points)

> Justifier qu'on peut toujours déterminer la position précédente $(n', m')$ à partir de la position actuelle $(n, m)$, et implémenter une fonction `dernier_coup` qui le fait.

**Question 5** (20 points)

> Justifier qu'on peut reconstituer l'unique suite de coups possible d'une partie depuis la position initiale jusqu'à la fin [3] en ne connaissant que la dernière position, et écrire une fonction `historique` qui le fait.
> *Indication : un tableau non redimensionnable n'est pas la structure la plus pratique pour manipuler des suites de coups ici !*

Dans la suite, on appelera *historique* d'une position la suite de coups depuis $(1, 1)$ qu'il faut effectuer pour atteindre cette position. La fonction `historique` calcule l'*historique*. [4]

On va maintenant classer tous les historiques possibles dans l'ordre : d'abord par longueur, puis par ordre lexicographique [5]. À partir des historiques, on va aussi ordonner les positions : deux positions dont les historiques sont consécutifs seront consécutives. Les premiers historiques et positions dans l'ordre sont ainsi :

| **Historique** | ∅ | G | D | GG | GD | DG | DD | GGG | … |
|---|---|---|---|---|---|---|---|---|---|
| **Position** | $(1,1)$ | $(2,1)$ | $(1,2)$ | $(3,1)$ | $(2,3)$ | $(3,2)$ | $(1,3)$ | $(4,1)$ | … |

**Question 6** (25 points)

> (a) Écrire une fonction `successeur` pour calculer le successeur d'un historique, c'est-à-dire l'historique qui vient juste après dans l'ordre.
> (b) En déduire une façon naïve d'implémenter la fonction `suivante` qui calcule la position suivante à partir d'une *position*.

3. Rappelez-vous que le jeu n'est jamais terminé. La « fin » ne désigne ici que le moment où le joueur ou la joueuse a atteint un état d'épuisement trop grave pour continuer à appuyer frénétiquement sur son écran.

4. Ceci est une identité non-triviale, les deux « historique » étant dans des polices différentes.

5. C'est comme l'ordre alphabétique, mais sur un alphabet où les deux lettres s'appellent « appuyer à gauche » et « appuyer à droite ».

**Question 7** (25 points)

(a) Expliquer comment la fonction `suivante` permet d'afficher la liste de toutes les positions atteignables en au plus $n$ coups.

(b) Implémenter une fonction n'utilisant pas `suivante` pour calculer cette liste.
*Indication : il est possible de le faire en temps linéaire en la taille de la sortie.*

## Partie III : Triche assistée par ordinateur

Dans une nouvelle version de Coprime Crush, il devient possible de gagner ou perdre au jeu[6]. Comme il n'y a pas de sens à Coprime Crush, ni à la vie en général, la condition de victoire ou de défaite est inconnue des joueurs[7].

On sait seulement qu'en jouant assez longtemps, l'on finit par gagner ou perdre, et qu'*il n'est pas impossible de gagner*. Pour savoir si l'on gagne ou perd, on dispose d'une fonction `test` prenant en argument le couple $(n, m)$ et renvoyant « gagnant », « perdant » ou « inachevé ».

L'application offre donc au joueur trois actions : gauche, droite, tester. Dès que l'on atteint une position gagnante ou perdante, notre sort est déterminé, quels que soient les coups que l'on joue après ; on ne l'apprend que lorsque l'on teste. Autrement dit, à partir d'une position gagnante (resp. perdante), on ne peut atteindre que des positions gagnantes (resp. perdantes).

**Question 8** (25 points)

(a) Donner deux algorithmes qui, grâce à la fonction `test`, trouvent une position gagnante dont l'historique est le moins long possible :
- une solution faisant appel à la fonction `suivante` ;
- une autre solution ne procédant pas par calcul itéré de position suivante.

(b) Comparer les performances des deux solutions.

Les deux prochaines questions ont pour but d'optimiser votre programme de triche.

**Question 9** (40 points)

(a) Décrire un algorithme qui, à partir d'une position $p$ en argument, détermine la position `q` telle que :
- $p$ et la position qui la suit soient toutes deux atteignables à partir de $q$ ;
- l'historique de $q$ soit le plus long possible.

De plus, il est préférable que votre algorithme ne nécessite pas de calculer l'historique entier de $p$, de `suivante`$(p)$ ou de $q$.

(b) S'en inspirer pour écrire une implémentation moins naïve de `suivante`.

**Question 10** (50 points)

(a) Montrer que `suivante` peut en fait être implémentée en n'ayant recours qu'à ces opérations arithmétiques : $+$, $-$, $*$, quotient et reste de la division euclidienne. (Sont proscrits toute forme de branchement (conditionnelles, boucles, `goto`…) et les appels à d'autres fonctions que les opérations autorisées.)

6. …
7. Certains auraient essayé de déchiffrer le code assembleur du jeu ; on ne sait pas ce qu'ils sont devenus.

(b) Comment cette implémentation de la mort qui tue influe-t-elle sur la performance des fonctions des questions 7 et 8 qui appellent `suivante` ?

## Partie IV : La stratégie de la victoire

En réaction à la prolifération des programmes de triche, l'éditeur de Coprime Crush sort encore une nouvelle version où on ne peut même plus tester si on a gagné ou perdu : tout ce qu'on peut faire, c'est jouer et attendre que l'application nous signale, une fois qu'elle le juge bon, qu'on a gagné ou qu'on a perdu. (Mais il est toujours garanti que si on joue suffisamment longtemps [8], on finira par avoir comme réponse « gagné » ou « perdu ».)

Là, c'en est vraiment trop, vous décidez donc de laisser un ordinateur jouer à votre place. Pour cela, vous voulez coder des *stratégies*, c'est-à-dire des processus générant une suite de coups. Ce que vous attendez d'une stratégie, c'est de pouvoir lui demander le prochain coup d'une partie ; ça devrait idéalement pouvoir se brancher sur un robot qui se charge d'appuyer au bon endroit. Quelques exemples :

- une stratégie répétant des coups selon un motif périodique ;
- une stratégie qui demande chaque prochain coup sur l'entrée standard, ce qui vous permet de jouer vous-même quand l'addiction vous reprend.

**Question 11** (30 points)

(a) Décrire une représentation des stratégies dans votre langage de programmation préféré, de sorte à ce que l'on puisse donner une stratégie en argument à une fonction, et écrire des fonctions qui renvoient des stratégies.

(b) Expliquer comment représenter les exemples ci-dessus.

*Indication : on pourra se servir de la programmation orientée objet, de fonctions de première classe, ou de structures paresseuses.*

Grâce à un effort colossal de rétro-ingénierie, des pirates ont réussi à détourner le code de l'appli pour fournir une fonction `strategie_gagnante` prenant une stratégie en entrée et renvoyant un booléen disant si cette stratégie est vouée à la réussite ou à l'échec [9]. L'avantage par rapport à l'utilisation de votre robot, c'est que vous pouvez ainsi tester une stratégie qui va appeler `strategie_gagnante` sur une autre stratégie, qui à son tour pourra appeler `strategie_gagnante`… tout ça sans avoir à dupliquer le complexe robot-smartphone !

**Question 12** (100 points)

À partir de la fonction `strategie_gagnante`, écrire une expression définissant une stratégie `s` telle que `strategie_gagnante(s)` renvoie `true`. Si aucune telle stratégie n'existe [10] , l'évaluation de votre expression devra signaler une erreur (par une valeur de retour spéciale, une exception…) et ne pas boucler infiniment.
*(Cette question est difficile. Vous pouvez passer aux questions bonus sans avoir traité celle-ci. La preuve est encore plus difficile et n'est pas exigée ; les courageux voulant gagner plus de la moitié des points pourront tenter d'appliquer le lemme de König.)*

Si vous avez réussi la question précédente, félicitations, vous avez gagné définitivement à Coprime Crush ! Mais le sujet n'est pas fini pour autant…

8. À la fois en temps et en nombre de coups.
9. Connaissant ces développeurs vicieux, la plupart du temps, ce sera l'échec.
10. cf. la note de pied de page précédente.

**Question bonus 13** (0 point)

> Vaut-il mieux passer l'éternité à rentrer une suite de coups infinie dans Coprime Crush, ou à pousser un roc au sommet d'une colline ?

## Partie bonus : Investigations arithmétiques

Où l'on démasque le réel voilé derrière les symboles… euh, plutôt, où l'on explique le titre.

**Question bonus 14** (5 points)

> Montrer qu'un couple $(n, m)$ est une position atteignable si et seulement si $n$ et $m$ sont premiers entre eux, et démontrer l'affirmation admise à la question 3. Reconnaissez-vous un algorithme de test de coprimalité dans le sujet ?

**Question bonus 15** (10 points)

> Écrire un programme énumérant les rationnels strictement positifs, de telle sorte que chacun d'entre eux apparaisse exactement une fois dans la suite (démonstration exigée). Une réutilisation des questions précédentes est évidemment attendue.

**Question bonus 16** (15 points)

> Définir la notion idoine d'historique d'un rationnel ($h : \mathbb{Q}_+^* \to \{\text{gauche}, \text{droite}\}^*$) [11], et prouver que
>
> $$\forall x, y \in \mathbb{Q}_+^*,\ x \leq y \Leftrightarrow h(x) \preceq h(y)$$
>
> où on note $\preceq$ non pas l'ordre employé plus haut, mais l'unique relation d'ordre partielle (qui se trouve en fait être totale, pourquoi ?) telle que
>
> $$\forall v, w \in \{\text{gauche}, \text{droite}\}^*,\ v \cdot \text{gauche} \cdot w \prec w \prec v \cdot \text{droite} \cdot w$$

**Question bonus 17** (12 points)

> Regardez attentivement l'image du début. [12] Pourquoi 337 ? POURQUOI ???

## Et pour finir…

Il est recommandé d'avoir traité toutes les questions précédentes en $(-1)$ heures avant de s'attaquer à cette ultime question.

**Dissertation** ($\emptyset$)

> Quelle est la place de Coprime Crush dans la Vie, l'Univers et le Reste ?

$$\mathcal{FIN}$$

Le sujet comporte 8 pages (incluant la page de garde) et 17 questions, parmi lesquelles 5 questions bonus [13]. Les questions normales sont notées sur 337 points, et les questions bonus rapportent au total 42 points, plus 21 points de présentation, ce qui fait au total 400 points.

11. Attention à ne pas confondre étoile privatrice de zéro et étoile de Kleene !
12. Attention à ne pas vous faire mal aux yeux.
13. Oh là là, c'est long ! Contrairement à la correction du QCM 2015, ce n'est pas un record de longueur, « Prologin Plays Pokémon » fait 9 pages et 14+3 questions.