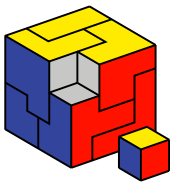
Prologin 2019



# Concours national d'informatique

## Épreuve écrite d'algorithmique
## Toulouse & Strasbourg

23 février 2019

# Des vacances bien méritées

## 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, ou préparez votre présentation pour l'entretien.
— 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, sinon ça va barder.
— 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 Une envie de partir loin, très loin

Cette année, Joseph Marchand doit corriger [1] toutes les copies de toutes les épreuves régionales du concours Prologin. Le seul bémol est qu'il aimerait beaucoup partir en vacances dès que les épreuves régionales seront finies. Mais il sait que cette tâche lui prendra beaucoup de temps... Après de longues heures d'intense réflexion, une brillante idée lui traversa l'esprit : **faire un programme qui corrige les copies à sa place**.

Avec l'aide d'un ami, il arrive facilement à scanner toutes les copies et récupérer le contenu des copies dans un format textuel. Cependant, il souhaiterait aussi pouvoir détecter la triche, pour cela il a besoin de pouvoir chercher un mot dans une chaîne de caractères. Après de très longues soirées de recherche infructueuses, il n'arrive pas à trouver une manière efficace pour chercher un mot dans une chaîne de caractère.

Votre mission, si toutefois vous l'acceptez [2] est de trouver une méthode révolutionnaire [3] de chercher un mot dans une chaîne de caractères.

## 3 Les vacances sont encore loin...

Dans la suite du sujet, un mot sera une sous chaîne d'une chaîne de caractères. Par exemple, `"Hello, Wo"` est un mot de la chaîne de caractères de `"Hello, World!"`. De plus, si vous ne savez manipuler des chaînes de caractères, pour pouvez :

- Accéder à au `i`ème d'une chaîne de caractères avec `chaine[i]`
- Concatener plusieurs chaînes de caractères ensemble avec `a + b`, par exemple `"Hello, " + "World!"` donne `"Hello, World!"`
- Comparer deux caractères avec les opérateurs : >, < et ==

### Question 1 (1 point)

> Écrire une fonction `chaine_equal(s1, s2)` qui renvoie `True` si les deux chaînes de caractères sont identiques.

### Question 2 (2 points)

> Écrivez une fonction `liste_mot(s)` qui renvoie la liste de tous les mots possible du texte `s`.

### Question 3 (2 points)

> Écrivez une fonction `chercher(foin, aiguille)` qui retourne l'index (la position) du premier caractère de la première occurrence (apparition) du mot `aiguille` dans le texte `foin` ou `-1` si vous ne trouvez pas l'`aiguille`.

### Question 4 (1 point)

> Estimez le nombre d'instructions exécutées dans le pire des cas par l'algorithme écrit pour la question précédente en fonction de la taille du texte et de la taille du mot.

1. Il a été déclaré volontaire
2. Bien heureusement, cette copie ne s'auto-détruira pas dans 42 secondes !
3. mais surtout efficace

# 4 Joseph dépasse les bords

Joseph découvre que la solution étudiée dans la section précédente n'est pas du tout assez efficace pour la quantité de texte qu'il a [4]. Mais en se baladant sur internet, il découvre que quand on travaille sur les chaînes de caractères, il existe un outil super efficace : **les bords**. Seulement il ne comprends pas tout et vous demande [6] de l'aide, ou plutôt vous transmet les définitions qu'il a trouvé.

*Les bords sont des sous-chaînes de caractères qui sont à la fois un préfixe et un suffixe, c'est à dire que pour une chaîne $S$ on retrouve le bord $B$ au début et à la fin de $S$. Un préfixe d'un mot est un sous mot qui se trouve au début de ce mot : `"tata"` a comme préfixe `"t"` `"ta"` `"tat"` et `"tata"`. Un suffixe d'un mot est un sous mot qui se trouve à la fin de ce mot : `"tata"` a comme suffixe `"a"` `"ta"` `"ata"` et `"tata"`. Les bords de `"tata"` sont donc `"ta"` et `"tata"`.*

## Question 5 (1 point)

> Selon cette définition, énumérez tous les bords de la chaîne de caractères `"ababab"`.

## Question 6 (2 points)

> Écrivez une fonction `est_bord(mot, texte)` qui retourne `True` si un mot `mot` est un bord d'un texte `texte`.

## Question 7 (1 point)

> Estimez le nombre d'instructions exécutées dans le pire des cas par l'algorithme écrit pour la question précédente en fonction de la taille du texte et de la taille du mot.

## Question 8 (2 points)

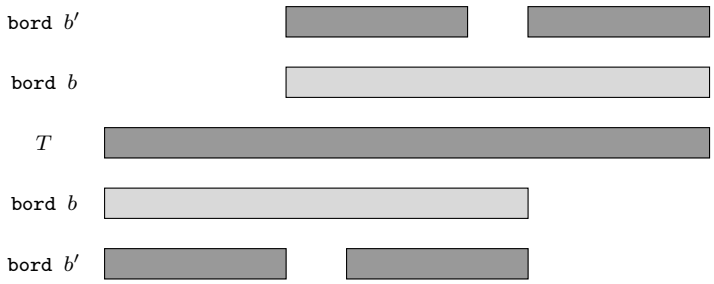
> Écrivez une fonction `bords_naif(texte)` qui calcule tous les bords du texte `texte` naïvement.

## Question 9 (1 point)

> Estimez le nombre d'instructions exécutées dans le pire des cas par l'algorithme écrit pour la question précédente en fonction de la taille du texte et de la taille du mot.

Seulement dans cette implémentation naïve on répète beaucoup de comparaisons, on peut notamment remarquer que si on sait que `"a"` n'est pas un bord de `"tata"` `"ata"` ne le sera pas non plus. Par contre si on sait que `"tata"` est un bord de notre texte $T$ alors `"ta"` est aussi un bord de $T$, en fait on peut remarquer que si un mot $b'$ est un bord de $b$ et que $b$ est le bord d'un texte $T$, alors $b$ est aussi un bord de $T$.

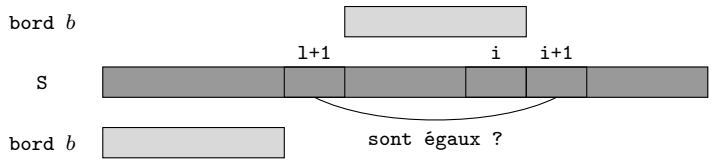
FIGURE 1 – Un bord d'un bord est un bord du texte total.




4. C'est que ça écrit un candidat Prologin... [5]
5. ... Surtout quand il vous êtes 420 !
6. encore

Cette remarque nous permet de voir que le plus long bord d'un texte $T$ qui n'est pas tout le texte $T$ est très intéressant. On les appelle les bords `"stricts"` et quand on parlera de plus long bord dans la suite du sujet on parlera toujours du plus long bord `"strict"`, (le texte tout entier est un bord de lui-même, mais ce n'est pas passionnant...). En fait, les ressources que vous a données Joseph vous disent que ce qui est crucial pour calculer efficacement sur des textes ce sont les bords des préfixes du texte. C'est dû à une propriété cruciale : si on a $b$ le plus long bord de $T[1:i]$ (préfixe de $T$ jusqu'au caractère $i$) pour trouver facilement le plus long bord de $T[1:i+1]$, on peut utiliser l'astuce suivante : si le bord $b$ est de taille $l$, il suffit de comparer $T[i+1]$ et $T[l+1]$.

FIGURE 2 – Extension d'un bord de $T[1:i]$.




### Question 10 (1 point)

> Donnez le plus long bord de `"ababab"`.

### Question 11 (5 points)

> Écrivez une fonction `taille_bords(s)` qui retourne un tableau dont la `i`ème case contient la taille du plus long bord des `i`ème premiers caractères de la chaîne de caractères `s`.

### Question 12 (2 points)

> Donnez la complexité de l'algorithme de la question précédente, c'est-à-dire le nombre d'opérations effectuées par l'algorithme dans le pire des cas en de la taille de l'entrée.

## 5 Par dessus bord [7]

Vous pouvez désormais supposer que vous disposez d'une fonction `calculer_bord` qui vous donne le tableau `bord` des plus longs bords de tous les préfixes, peu importe que vous ayez réussi à répondre à la question `calcul_bord` ou non. On passe maintenant à l'utilisation de ce tableau pour résoudre tout un tas de problèmes [8]. Dans cette partie on va aborder des propriétés classiques que l'on peut exploiter.

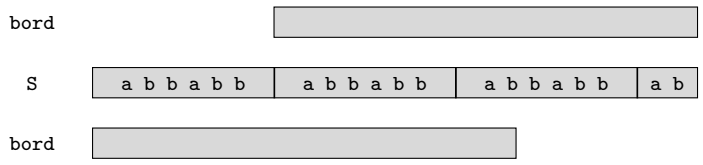
### Question 13 (2 points)

> Donnez un algorithme qui, à partir de la fonction `calculer_bord`, vous permet de trouver toutes les apparitions d'un mot `P` dans un texte `T` en faisant un nombre d'opération de l'ordre de la taille du texte plus la taille du pattern.

Un des grands avantages des bords c'est qu'ils sont liés à la répétitivité du texte, plus un texte a un bord long plus il est répétitif. Par exemple par `"abcabcabcabcabc"` à pour plus long bord `"abcabcabcabc"` et peut en fait être décris comme un mot de période `"abc"` répété 5 fois, alors que `"prologin"` n'a pas de bord strict parce qu'il n'y a pas de répétition. Ce n'est pas une coïncidence on pourra en fait se convaincre qu'un bord nous donne toujours une période. Plus généralement, on dit qu'un mot $W$ est une période d'un mot $M$ si $M$ est écrit par un certain nombre de répétitions de $W$ : $M = W\ W\ W\ ...\ W$.

7. kimicol sort de ce corps
8. et enfin cette de recherche de mot, c'est pas trop tôt !

FIGURE 3 – Un bord nous donne une période.




### Question 14 (3 points)

> À partir de ce constat et du calcul des bords écrivez une fonction qui énumère toutes les périodes possible d'un mot.

## 6 Les vacances sont en vues !

Les problèmes de cette section sont plus difficiles, n'hésitez pas à donner une esquisse d'idée et à détailler les problèmes dont vous n'arriveriez pas à trouver une solution.

### Question 15 (4 points)

> On vous donne un mot $w$ de taille $n$ et $m$ entiers $k_1, ..., k_m$, écrivez une fonction qui donne pour chaque $k_i$ la longueur du plus petit mot $p$ tel que $w$ est un sous mot de $p^{k_i}$.

Exemple : `"abcabcabca"` pour $k = 2$ $p = abcabc$ donc la réponse est 6 mais pour $k = 4$, il suffit d'avoir $p = abc$ donc la réponse est 3.

### Question 16 (2 points)

> Donnez la complexité de l'algorithme de la question précédente.

### Question 17 (4 points)

> On a un texte qu'on cherche à taguer sur un mur mais on veut le faire avec le plus petit pochoir, seulement quand on tague un pochoir on est obligé de peindre tout le pochoir on ne peux pas juste en utiliser une partie. Étant donné un texte donner la plus petite taille du pochoir possible.

FIGURE 4 – Exemple pour le texte `"ababbababbababbababababbababbaba"`.


### Question 18 (2 points)

> Donnez la complexité de l'algorithme de la question précédente.

# 7 Bonus

### Question bonus 19 (2 points)

> Réécrivez la fonction de la question 13 avec la pire complexité possible, cette fonction doit converger vers la solution et s'arrêter.

### Question bonus 20 (1 point)

> Donnez le mot avec le plus long bord qui vous connaissez.

### Question bonus 21 (0 point)

> Dessiner le lieu de vacances idéal pour Joseph.

### Question bonus 22 (42 points)

> Aidez Joseph à partir plus vite en vacance en corrigeant votre copie. Écrivez le nombre de points que vous pensez avoir [9].

Le sujet comporte 6 pages (sans compter la page de garde), 22 questions, et 45 questions bonus. Les questions normales sont notées sur 38 points, plus les questions bonus et 1 point de présentation.

9. Bien évidemment, la **totalité** de la partie bonus compte.