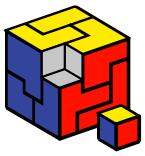
Prologin 2014

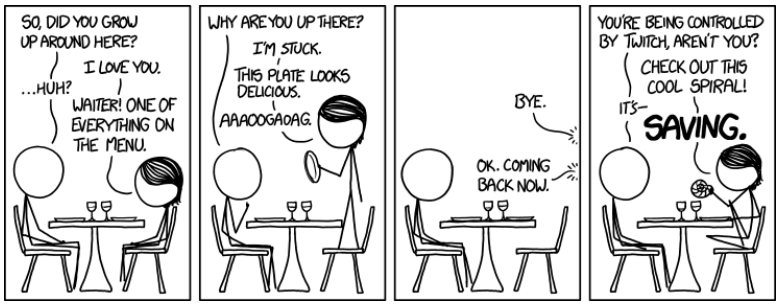


Concours national d'informatique

Épreuve écrite d'algorithmique
Louvain-la-Neuve, Lyon & Nantes

Samedi 1er mars 2014

Prologin Plays Pokémon



XKCD – « First Date », par Randall Munroe

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 Sujet

Introduction

Ce samedi 1er mars 2014, l'épreuve régionale Prologin ne comportait pas d'épreuve écrite. C'est pourquoi vous [1] avez décidé de lancer une partie de Pokémon [2]. Manque de chance, votre idée séduit les 58 autres candidats, mais vous n'avez qu'une seule cartouche.

Généreux comme vous êtes, vous testez une expérience sociale inédite [5]. Vous allez jouer tous ensemble de manière coopérative. Ainsi vous prêtez l'oreille à vos camarades et dès que l'un d'entre eux vous conseille une touche, vous appuyez dessus.

Après cette proposition, la pression vous envahit, vous ne pouvez plus reculer [6] même si ça tourne au vinaigre.

Et le vinaigre ne se fait pas attendre. Des organisateurs mignons militants motivés par milliers [7] vous rejoignent et vous n'arrivez plus à suivre.

Vous avez des dizaines de secondes de retard sur les ordres, ce qui engendre de véritables catastrophes. Comme l'informatique ce n'est pas votre fort, vous désignez un scribe, qui note les ordres avant que vous puissiez les exécuter [8]. Il fait notamment le tri entre les commandes et les commentaires (comme « Le Fossile Nautile a payé pour nos péchés ! »).

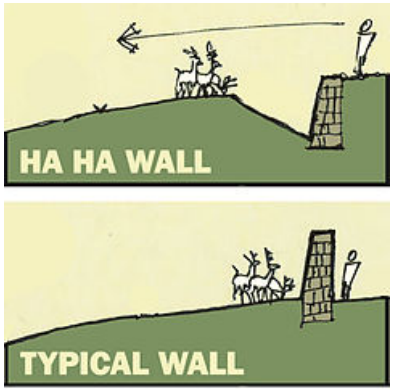
Les commandes sont :

— « a », représenté par le code 0 ;
— « b », représenté par le code 1 ;
— « start », représenté par le code 2 ;
— « select », représenté par le code 3 ;
— « haut », représenté par le code 4 ;
— « bas », représenté par le code 5 ;
— « gauche », représenté par le code 6 ;
— « droite », représenté par le code 7.

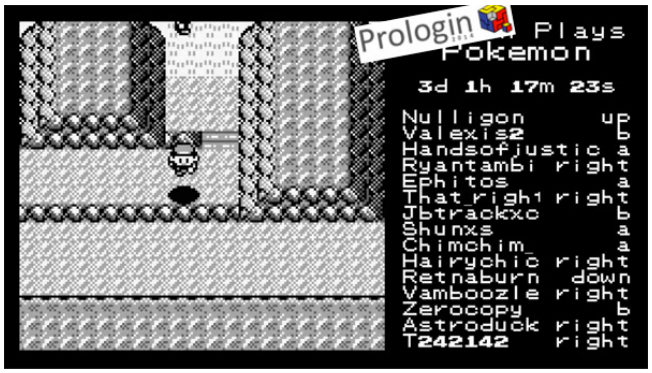
Comme vous le savez, dans Pokémon, il existe des bordures [9] que l'on ne peut franchir uniquement dans un sens (comme illustré par la figure 1).

En suivant la bêtise collective, vous craignez fort de franchir cette bordure [9] qui vous forcerait à reprendre votre trajet depuis le départ.

1. Comprendre : vous seul.
2. Puisque jeudi dernier on fêtait les 18 ans des premières versions de Pokémon [3]
3. Rouge et vert, pour les incultes [4].
4. Ou ceux d'entre vous qui ne sont pas encore majeurs.
5. Du jamais vu depuis *Twitch Plays Pokémon*.
6. La fuite est impossible, héhé [9].
7. Ouais, j'écris ce que je veux.
8. Ce qui revient exactement au même.
9. Ou ha-ha [10].
10. Ou hâ-hâ ou saut de loup [11].
11. Ou ax-ax, en russe.



(a) Ha-ha [10]



(b) Bordure [9]

FIGURE 1 – Exemples de bordures [9]. Si vous arrivez par le haut, vous franchissez la bordure [9], mais si vous arrivez par le bas elle vous stoppe.

Question 1 (2 points)

> Écrire une fonction qui prend en entrée la liste des commandes (une suite de nombres représentant les commandes « haut », « bas », « gauche » ou « droite ») de vos camarades et renvoie vrai si vous franchissez la bordure [9] (qui est actuellement juste en dessous de vous) et faux sinon.

Il semble que certains individus ne soient pas très coopératifs. Alors que vous souhaitez avancer tranquillement vers la prochaine ville, ils appuient sans cesse sur des boutons dont l'utilité est douteuse, comme « start » ou « select ». Vous essayez donc de repérer ces phases inutiles.

Question 2 (2 points)

> Écrire une fonction qui prend en entrée la liste des commandes et renvoie le nombre maximum de « start » proposés à la suite (sans interruption).

La plèbe gronde. Vous l'avez écoutée sans réfléchir et vous avez mis tout Kanto en flammes. Il faut réagir. En bon meneur, vous savez que pour contenter un peuple en colère il faut lui donner l'illusion du pouvoir. Ainsi, vous proposez le système [12] suivant. Vous n'allez pas tout de suite appuyer sur les touches. Vous allez les laisser parler pendant quelques instants puis vous choisirez la proposition majoritaire (un élément est dit majoritaire s'il recueille plus de la moitié des voix) parmi celles notées par le scribe.

Pour déterminer l'élément majoritaire s'il existe, on propose l'algorithme suivant.

Question 3 (1 point)

> Voici un exemple de liste de commandes : « droite », « haut », « haut », « droite », « droite », « haut », « droite ».
>
> Appliquer l'algorithme *Majoritaire* sur cet exemple.

12. Soi-disant démocratique.

Algorithme retournant la commande majoritaire.

```
procédure MAJORITAIRE(flux)
    compteur ← 1
    majoritaire ← lire_caractere(flux)
    tant que x ← lire_caractere(flux) faire        ▷ Tant qu'il y a une commande à lire.
        si x = majoritaire alors
            compteur ← compteur + 1
        sinon
            compteur ← compteur − 1
            si compteur = 0 alors
                compteur ← 1
                majoritaire ← x
    renvoyer majoritaire
```

Question 4 (3 points)

> Justifier que s'il existe une commande majoritaire, alors cet algorithme renverra effectivement cette commande.

Question 5 (3 points)

> Combien de comparaisons effectue cet algorithme ? Quelle quantité de mémoire utilise-t-il ?

Après plusieurs heures, vous avez relâché la majorité des Pokémon de votre équipe. Vous avez réussi à isoler la suite de coups qui provoque ces malheurs et vous voulez désormais l'éviter à tout prix. Vous décidez alors que ces commandes ne doivent pas apparaître dans cet ordre, même avec d'autres commandes intercalées, dans un souci de sûreté (par exemple, si 1, 2, 3, 4 est dangereux, vous n'accepterez pas 1, 6, 5, 2, 2, 4, 3, 1, 4).

Question 6 (2 points)

> Écrire une fonction qui prend en entrée la liste des commandes et la suite de commandes dangereuses, et renvoie vrai si la suite dangereuse est présente (même avec interruptions) et faux sinon.

Ouf, vous allez vous en sortir. Mais pour repartir du bon pied, il vous faut remplir à nouveau votre équipe de Pokémon. Vous connaissez l'enchaînement de commandes à effectuer mais aucune autre commande ne doit s'intercaler, ou l'opération sera un échec (par exemple, si vous devez réaliser 1, 2, 3, 4, vous accepterez 1, 1, 6, 1, 2, 3, 4, 7, 5 mais pas 1, 2, 5, 5, 3, 4).

Question 7 (4 points)

> Écrire une fonction qui prend en entrée la liste des commandes et l'enchaînement de commandes nécessaires, et renvoie vrai si l'enchaînement est présent (sans interruption) et faux sinon [13].

Votre système commence à bien fonctionner mais vous perdez drastiquement en audience. En effet, avec ces votes la vitesse de jeu a fortement ralenti et certains trouvent le système injuste. Afin de pallier ces problèmes, vous faites une nouvelle proposition : les 3 commandes les plus plébiscitées seront exécutées. On propose l'algorithme suivant.

Algorithme retournant les k commandes majoritaires.

```
procédure MAJORITAIRES(flux, k)
    majoritaires ← Tableau associatif vide
                                  ▷ Par exemple dictionnaire en Python ou map en C++.
    tant que x ← lire_caractere(flux) faire        ▷ Tant qu'il y a une commande à lire.
        si x est dans majoritaires alors
            majoritaires[x] ← majoritaires[x] + 1
        sinon si nombre d'éléments dans  majoritaires < k-1 alors
            majoritaires[x] ← 1
        sinon
            pour y ∈ majoritaires faire
                majoritaires[y] ← majoritaires[y] − 1
                si majoritaires[y] = 0 alors
                    retirer y de majoritaires
    renvoyer majoritaires
```

Question 8 (1 point)

> Voici un exemple de liste de commandes : « haut », « droite », « a », « haut », « gauche », « bas », « a », « bas », « haut », « b », « b », « start », « bas », « select », « bas », « start », « haut ».
>
> Appliquer l'algorithme *Majoritaires* sur cet exemple.

Question 9 (3 points)

> Justifier que si une commande est présente plus de $\frac{n}{k}$ fois alors la commande est présente parmi les commandes majoritaires renvoyées par cet algorithme.

Question 10 (2 points)

> Combien de comparaisons effectue cet algorithme ? Quelle quantité de mémoire utilise-t-il ?

Votre soif de pouvoir est insatiable. Malgré ce système de votes, vous voulez manipuler le scrutin pour arriver à votre fin. Vous choisissez de relancer un vote toutes les 100 commandes et allez déterminer à quel moment il fallait commencer le premier scrutin. Ainsi, il vous faut calculer la commande majoritaire sur chaque intervalle de 100 commandes consécutives.

13. Pour obtenir les 4 points à cette question, on s'attend à un algorithme faisant un nombre linéaire de comparaisons.

Question 11 (4 points)

> Écrire une fonction qui prend en entrée la liste de n commandes et renvoie la commande majoritaire pour chacun des $n - 99$ intervalles de 100 commandes consécutives.

Votre dernière manipulation n'est pas passée inaperçue. L'audience vous suspecte de ne pas tenir compte de certaines commandes. Afin de ne pas risquer une pluie de tomates, vous n'exécuterez plus la commande la moins votée. On propose l'algorithme suivant pour obtenir rapidement et sans utiliser trop de mémoire une estimation du nombre de votes pour chaque commande.

Algorithme retournant le nombre de votes minimum, d'abord la première procédure est appelée sur le flux de commandes, puis on utilise la seconde procédure pour obtenir une estimation du nombre de votes pour chaque coup.

---

procédure PRECALCUL_VOTE($flux, d, w, p$)
  $D \leftarrow d$ tableaux de taille w initialisés à 0
  a et b deux tableaux de nombres pris aléatoirement entre 1 et p.
  tant que $x \leftarrow lire\_caractere(flux)$ **faire** ▷ Tant qu'il y a une commande à lire.
    pour j allant de 0 à $d - 1$ **faire**
      $cible \leftarrow (a_j * x + b_j \mod p) \mod w$
      $C[j][cible] \leftarrow C[j][cible] + 1$
  renvoyer C

procédure ESTIMATION_VOTE(C, x)
  $min \leftarrow +\infty$
  pour j allant de 0 à $d - 1$ **faire**
    $cible \leftarrow (a_j \times x + b_j \mod p) \mod w$
    $min \leftarrow$ minimum entre $C[j][cible]$ et min
  renvoyer min

Question 12 (4 points)

> Justifier que cet algorithme renvoie une valeur supérieure ou égale aux nombre de votes pour n'importe quelle commande.

Vous touchez au but. Il vous faut simplement faire un dernier pas vers le haut mais vous craignez qu'une fois de plus votre tentative échoue. Vous devez alors trouver une longue période pendant laquelle « haut » est majoritaire.

Question 13 (4 points)

> Écrire une fonction qui prend en entrée la liste de commandes et renvoie le plus grand intervalle pour lequel « haut » est majoritaire.

Vous pouvez attaquer les questions suivantes si et seulement si vous avez réussi à collecter toutes les baies.

Question bonus 14 (1 point)

Écrire une fonction qui renvoie le plus petit intervalle de commandes sur lequel « haut » est majoritaire.

Question bonus 15 (5 points)

Quelle est la probabilité que l'algorithme présenté précédemment ne renvoie pas le nombre de votes pour n'importe quelle commande ?

Algorithme tirant aléatoirement k éléments parmi n.

```
procédure RESERVOIR_SAMPLING(S, n, k)
    R ← tableau de taille k
    pour i allant de 0 à k − 1 faire
        R[i] ← S[i]
    pour i allant de k à (taille de S) − 1 faire
        Tirer j aléatoirement entre 0 et i − 1
        si j < k alors
            R[j] ← S[i]
    renvoyer R
```

Question bonus 16 (4 points)

Justifier que cet algorithme (qui prend en entrée une liste S de n commandes) tire k commandes uniformément au hasard.

Question bonus 17 (1 point)

Dans quel objet placez vous votre foi ? Justifiez votre réponse.

1. Fossile nautile
2. Fossile dôme
3. Passe bateau
4. Pierre lune
5. Autre objet
6. Pas de foi particulière

Le sujet est sur 35 points, et les questions bonus rapportent au total 11 points, plus 1 point de présentation.