e3a

Épre

Durée 3 h

015

CONCOURS ARTS E

n candic

f de sall

les raison

lat repère ce qui lui semble être une erreur d'énoncé, d'une

, d'autre part il le signale sur sa copie et poursuit sa

s des initiatives qu'il est amené à prendre.

---

le calculatrices est interdit.

usage c

Si, au cours de l'épreuve, u

part il le signale au che

composition en indiquant

AVERTISSEMENT

---

orthographe, la qualité de la **rédaction, la clarté et la**

entreront pour une **part importante** dans

L'bilité, l'carticulier, les résultats non justifiés ne seront pas pris

nements vités à encadrer les résultats de leurs calculs.

es. En p

s sont ir

ints.

La **présentation**, la lisil

précision des raison

indépendа

l'appréciation des copi

plexité, on attend un ordre de grandeur, par exemple $O(n)$.

en compte. Les candidat

r une com

partie entière.

Les différents exercices sont

ote $[x]$ sa

Lorsqu'on demande d'évalue

Tournez la page S.V.P

Pour un nombre réel x, on n

osition ou d'y mettre un signe quelconque pouvant indiquer sa provenance.

leur comp

Tournez la page S.V.P.

IERS ParisTech - ESTP - POLYTECH

Il est interdit aux candidats de signer

Exercice 1

On admettra que la multiplication de deux entiers naturels se fait en temps constant.

1. On s'intéresse dans cette partie au calcul de la puissance k-ième d'un entier naturel n, pour un entier naturel $k \geq 1$.

   (a) Dans un premier temps, on utilise un algorithme naïf :

   ```
   let rec puissance n k =
       if k = 1 then n
       else n * (puissance n (k-1));;
   val puissance : int -> int -> int
   ```

   Quelle est la complexité de cet algorithme ? Le justifier précisément.

   (b) On utilise ensuite l'algorithme suivant :

   ```
   let rec puissance2 n k =
       if k > 1 then
           let x = puissance2 n (k/2) in
           if k mod 2 = 0 then
               x * x
           else
               x * x * n
       else n;;
   val puissance2 : int -> int -> int
   ```

   i. Décrire l'exécution du programme `puissance2` sur l'entrée $(2, 7)$.
   ii. Décrire l'exécution du programme `puissance2` sur l'entrée $(2, 8)$.
   iii. Démontrer que le nombre d'appels récursifs à l'intérieur du programme `puissance2` sur l'entrée (n, k) est au plus de $\log_2 k + 1$. En déduire que le programme termine.
   iv. Justifier que ce programme est correct.
   v. Evaluer la complexité de ce programme.

2. On s'intéresse ici au problème de déterminer si un entier naturel est une puissance non triviale d'un nombre entier ou non : on dit qu'un nombre entier naturel n est une *puissance entière* s'il existe deux entiers naturels k et m tous deux > 1 tels que $n = m^k$.

   (a) Ecrire la fonction :

   `test_puissance : int -> int -> bool`

   qui prend en entrée les entiers naturels $n > 1$ et $k > 1$ et renvoie le booléen `true` s'il existe un entier naturel m tel que $N = m^k$ et `false` sinon.

   (b) i. Soit n un entier naturel. On suppose que n est une puissance entière : Soient k et m deux entiers naturels > 1 tels que $n = m^k$. Justifier que $k \leqslant \log_2(n)$.

ii. Ecrire la fonction :

`test_puissance_entiere : int ->` [illegible]

qui prend en entrée l'entier naturel [illegible] entiers naturels k et m tous deux $>$ [illegible] tels que $n = m^k$ et false sinon.

iii. Démontrer que la complexité de ce programme est au plus $O(n \log_2(n) \log_2(k))$.

iv. En déduire la fonction :

`liste1_puissances_entieres : int -> int list`

qui prend en entrée l'entier naturel $n > 1$ et renvoie la liste des entiers naturels compris entre 2 et n qui sont des puissances entières.

(c) En vous inspirant du crible d'Erathosthène, écrire la fonction :

`liste2_puissances_entieres : int -> int list`

qui prend en entrée l'entier naturel $n >$ [illegible] 1 et renvoie la liste des entiers naturels compris entre 2 et n qui sont des puissances entières.

Exercice 2

Soit Σ l'alphabet des chiffres : $\Sigma = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$.
Un nombre entier naturel non nul n est considéré à l'aide de son écriture en base 10 comme un mot sur l'alphabet Σ :

$$n \text{ s'écrit } a_l a_{l-1} \cdots a_0 \Leftrightarrow n = a_l 10^l + a_{l-1} 10^{l-1} + \cdots + a_0, \text{ avec } a_l \neq 0.$$

Un mot sur Σ sera considéré comme un tableau de type `int vect` dont les éléments sont compris entre 0 et 9. Par exemple le nombre 2015 sera représenté par `[|2;0;1;5|]`.

On dit qu'un entier naturel non nul a la propriété P_{2015} si son écriture en base 10 comprend le motif 2015. On cherche à reconnaître les entiers naturels non nuls ayant cette propriété.

1. Dessiner un automate déterministe complet qui reconnaît exactement les entiers naturels non nuls ayant la propriété P_{2015}.

2. On considère la fonction G qui envoie un mot $abcd$ dans Σ^4 sur l'entier naturel $a * 1000 + b * 100 + c * 10 + d$.

   (a) Combien de valeurs distinctes la fonction G peut-elle prendre ?

   (b) Etant donné un mot $abcde$ dans Σ^5, expliciter comment calculer $G(bcde)$ en fonction de $G(abcd)$ et e.

   (c) Ecrire la fonction :

   `decalG : int -> int -> int`

   qui prend en entrée l'entier naturel $G(abcd)$ et le chiffre e dans Σ et renvoie la valeur $G(bcde)$, pour $abcde$ un mot dans Σ^5.

(d) On utilise la fonction G pour résoudre le problème. Etant donné un entier naturel non nul n s'écrivant $a_l a_{l-1} \cdots a_0$ en base 10, on calcule la valeur prise par G sur chaque facteur $a_i a_{i-1} a_{i-2} a_{i-3}$. Ecrire la fonction :

```
testG_motif : int vect -> bool
```

qui prend en entrée le tableau défini par l'écriture en base 10 d'un entier naturel non nul n et renvoie le booléen `true` si l'entier naturel n a la propriété P_{2015} et `false` sinon. Cette fonction ne lira qu'une fois au plus chacun des caractères de l'écriture en base 10 de n.

(e) Evaluer la complexité de votre algorithme.

(f) Ecrire la fonction :

```
nbG_motif : int vect -> int
```

qui prend en entrée le tableau défini par l'écriture en base 10 d'un entier naturel non nul n et renvoie le nombre d'occurences du motif 2015 dans l'écriture en base 10 de n.

3. On considère la fonction H qui envoie un mot *abcd* dans Σ^4 sur l'entier naturel compris entre 0 et 10 égal à $a * 1000 + b * 100 + c * 10 + d$ modulo 11.

   (a) Combien de valeurs distinctes la fonction H peut-elle prendre ?

   (b) Ecrire la fonction :

   ```
   calculH : int vect  -> int
   ```

   qui prend en entrée un mot *abcd* dans Σ^4 et renvoie la valeur de H prise sur ce mot.

   (c) Etant donné un mot *abcde* dans Σ^5, expliciter comment calculer $H(bcde)$ en fonction de $H(abcd)$, a et e.

   (d) En utilisant la fonction H, écrire la fonction :

   ```
   testH_motif : int vect -> int
   ```

   qui prend en entrée le tableau défini par l'écriture en base 10 d'un entier naturel non nul n et renvoie renvoie le booléen `true` si l'entier naturel n a la propriété P_{2015} et 0 (ou `false`) sinon. Cette fonction ne lira que trois fois au plus chacun des caractères de l'écriture en base 10 de n.

   (e) Evaluer la complexité de votre algorithme.

Exercice 3

On souhaite analyser les résultats de sondages concernant une élection. Les candidats à l'élection sont numérotés de 0 à $k-1$. Les résultats de chaque sondage sont stockés dans un tableau : si le sondage a recueilli N réponses, le tableau comporte N cases, une pour chaque réponse : la i-ème case du tableau contient le numéro du candidat proposé par la i-ème personne sondée. On a ainsi un tableau T de longueur N qui contient des entiers naturels entre 0 et $k-1$.

Le sondage donne le candidat numéroté i élu si le nombre i est dans strictement plus de $N/2$ cases de T. Par exemple, un sondage correspondant au tableau $[2,4,5,0,4,4,4]$ donne le candidat 4 élu. Mais un tableau ne donne pas toujours un élu, par exemple le tableau $[1,2,3,4,6,2,3,3]$.

On suppose qu'un entier k est défini.

1. (a) Écrire une fonction `nb` de type `int vect -> int -> int` telle que si `a` est un entier naturel et `tab` un tableau de taille N issu d'un tel sondage, `nb tab a` est le nombre de cases du tableau `tab` qui contiennent `a`.
   (b) Evaluer la complexité de l'appel de `nb` en fonction de N et de k.
   (c) En déduire une fonction `elu1` de type `int vect -> int` telle que `elu1 tab` est l'entier donné élu par le tableau `tab` si celui-ci existe et `-1` sinon.
   (d) Evaluer la complexité de votre algorithme en fonction de N et de k.

2. On se propose d'utiliser la stratégie diviser pour régner pour déterminer l'éventuel élu donné par un tableau. On dispose de deux fonctions, qu'on ne cherchera pas à décrire :

   - `miGauche : int vect -> int vect` qui prend en argument un tableau `tab` de longueur $N \geqslant 2$ et retourne le tableau de longueur $\lfloor N/2 \rfloor$ formé par les $\lfloor N/2 \rfloor$ premières cases de `tab`,
   - `miDroite : int vect -> int vect` qui prend en argument un tableau `tab` de longueur $N \geqslant 2$ et retourne le tableau de longueur $N - \lfloor N/2 \rfloor$ formé par les $N - \lfloor N/2 \rfloor$ dernières cases de `tab`.

   (a) Soit `tab` un tableau de longueur $N \geqslant 2$. Démontrer que si `tab` donne a comme élu alors celui-ci est aussi donné élu par le tableau `miGauche tab` ou par le tableau `miDroite tab`.
   (b) Proposer une fonction `elu2`, utilisant la stratégie diviser pour régner, de type `int vect -> int * int` ; si `tab` est un tableau, `elu2 tab` est le couple `(a,n)` si l'entier `a` est donné élu par `tab` et apparait dans `n` cases exactement de `tab`, et `(-1,0)` si `tab` ne donne pas d'élu. On pourra utiliser la fonction `nb` définie en 1(a).
   (c) Evaluer la complexité de cette fonction en fonction de N.

3. Soit T un tableau de longueur N. On dit que le nombre entier a est un *postulant* pour la valeur n du tableau T si : n est un entier strictement supérieur à $N/2$ tel que a apparait au plus (au sens large) n fois dans T et tout entier b distinct de a, apparait au plus (au sens large) $N - n$ fois dans T. Par exemple, 3 est un postulant pour $n = 5$ du tableau $[1, 2, 3, 4, 3, 2, 3, 3]$.

   On dit que le nombre entier a est un postulant du tableau T s'il existe un nombre entier $n > N/2$ tel que a est un postulant pour la valeur n du tableau T

   (a) Démontrer que si le tableau T donne a élu alors a est un postulant de T.
   (b) Démontrer que si a est un postulant de T alors aucun autre élément de T ne pourrait être donné comme élu.
   (c) Donner un exemple de tableau qui contient un postulant mais ne donne aucun élu et un exemple de tableau n'ayant aucun postulant.
   (d) Soit T un tableau de longueur un entier pair N. On note TG le tableau de longueur $N/2$ formé par les $N/2$ premières cases de T et TD le tableau de longueur $N/2$ formé par les $N/2$ dernières cases de T.
      i. On suppose que le tableau TD ne donne pas d'élu. Soit a un postulant pour la valeur l du tableau TG. Démontrer que a est un postulant de T. On exprimera la valeur d'un entier n tel que $n > N/2$ et a est un postulant pour la valeur n du tableau T en fonction de l et N.

ii. Soient a un postulant pour la valeur l du tableau TG et b un postulant pour la valeur m du tableau TD.

A. On suppose que $a = b$. Démontrer que a est un postulant de T. On exprimera la valeur d'un entier n tel que $n > N/2$ et a est un postulant pour la valeur n du tableau T en fonction de l et m.

B. On suppose que $a \neq b$ et $m > l$. Démontrer que b est un postulant pour la valeur $N/2 + m - l$ de T.

C. On suppose que $a \neq b$ et $m = l$. Démontrer que T ne donne pas d'élu.

(e) Ecrire une fonction `postulant : int vect -> int * int` telle que, si `tab` est un tableau d'entiers naturels de longueur N, `postulant tab` est un couple `(a,n)` tel que

- lorsque `postulant tab` renvoie le couple `(-1,0)`, le tableau `tab` n'a pas d'élu,
- lorsque `postulant tab` renvoie le couple `(a,n)` avec `n` $> N/2$, `a` est un postulant pour la valeur `n` du tableau `tab`. On supposera pour simplifier que la taille du tableau est une puissance de 2. La procédure utilisera la stratégie diviser pour régner et aura une complexité linéaire, ce qu'on justifiera précisément.

(f) En déduire une fonction `elu3`, de type `int vect -> int` qui prend en argument un tableau `tab` et retourne l'entier élu de `tab` si celui-ci existe et `-1` sinon, de complexité linéaire.