Algorithmique

TD2 : Listes, tableaux, tuples

1 Tableaux

1. Écrire un algorithme `croissant` qui prend en argument un tableau `tab`, et qui détermine si le tableau est trié dans l'ordre croissant.

2. Écrire un algorithme `minmax` qui prend en argument un tableau `tab`, et renvoie un couple `(x,y)` tel que x est l'élément maximal de tab, et y l'élement minimal.

3. Écrire un algorithme `occurrences` qui prend en argument un tableau `tab` et une valeur `x`, et qui renvoie la liste de toutes les positions `i` telles que `tab[i]=x`.

4. Écrire un algorithme `absents` qui prend en argument un tableau `tab` et un entier `n`, et qui renvoie la liste de tous les entiers inférieurs à n qui n'apparaissent pas dans `tab`.

   On pourra dans un premier temps écrire un algorithme quadratique faisant n appels à `occurrences`. Dans un second temps, on cherchera un algorithme à complexité linéaire.

2 Carrés magiques

Un carré magique $n \times n$ est une matrice de n^2 nombres telle que la somme des éléments d'une même ligne, d'une même colonne, ou d'une même diagonale donne toujours la même valeur. Un carré magique $n \times n$ est naturel si en plus il contient les nombres $1, \ldots, n^2$.

1. Écrire un algorithme `Magic` qui prend en paramètre un tableau bidimensionnel, et qui renvoie `Vrai` si ce tableau représente un carré magique, et `Faux` sinon.

   On pourra commencer par écrire des fonctions `SommeLigne`, `SommeColonne`, `SommeDiagonale`.

2. Écrire un algorithme `Natural` qui prend en paramètre un tableau bidimensionnel, et qui renvoie `Vrai` si ce tableau représente un carré magique naturel, et `Faux` sinon.

3 Crible d'Eratosthène

Voici la description Wikipédia du crible d'Eratosthène.

> L'algorithme procède par élimination : il s'agit de supprimer d'une table des entiers de 2 à N tous les multiples d'un entier (autres que lui-même).
>
> En supprimant tous ces multiples, à la fin il ne restera que les entiers qui ne sont multiples d'aucun entier à part 1 et eux-mêmes, et qui sont donc les nombres premiers.
>
> On commence par rayer les multiples de 2, puis les multiples de 3 restants, puis les multiples de 5 restants, et ainsi de suite en rayant à chaque fois tous les multiples du plus petit entier restant.
>
> On peut s'arrêter lorsque le carré du plus petit entier restant est supérieur au plus grand entier restant, car dans ce cas, tous les non-premiers ont déjà été rayés précédemment.
>
> À la fin du processus, tous les entiers qui n'ont pas été rayés sont les nombres premiers inférieurs à N.

1. En utilisant le crible d'Eratosthène, écrire un algorithme prenant en paramètre un entier n, et qui renvoie la liste de tous les entiers premiers inférieurs à n.

2. Exprimer, à l'aide d'une somme faisant intervenir l'ensemble $\mathbb{P}_n$ des nombres premiers inférieurs à n, le nombre de fois que l'algorithme effectue l'opération *rayer un nombre*. On verra en cours que cette somme vaut approximativement $n \ln \ln n$.

3. Comment pourrait-on faire pour éviter à l'algorithme de rayer plusieurs fois le même nombre ? On verra en cours que cela nécessite d'utiliser des structures de données bien particulières, comme les *listes doublement chaînées*.

4 Bon parenthésage

1. Écrire un algorithme `BienParenthese` qui vérifie si une expression donnée en paramètre sous la forme d'une chaîne de caractères `s` est bien parenthésée.

```
s0 = "(x+y)(z*(x/(z+y)+t)*(z+a))"
s1 = "(x+y*(z-t)"
s2 = "(x+y)+z)*(a*(b+c)
BienParenthese(s0)
>> Vrai
BienParenthese(s1)
>> Faux
BienParenthese(s2)
>> Faux
```

   Cela signifie que dans tout préfixe de l'expression, on a au moins autant de parenthèses ouvrantes "(" que de parenthèses fermantes ")", et un même nombre dans l'expression totale.

2. Même question, mais cette fois-ci différent types de parenthèses peuvent apparaître dans l'expression : `( )` `[ ]` `{ }`.

   Cette fois, il faut également vérifier qu'une parenthèse ouvrante n'est pas suivie d'une parenthèse fermante d'un autre type.