Test algorithmique n°1 : structures de contrôle, tableaux

Dans tout ce devoir, les algorithmes seront écrits en français, ou éventuellement en "pseudo-Pascal". Une syntaxe lisible est demandée, en particulier les structures devront être correctement indentées. mais l'accent sera surtout mis lors de l'évaluation sur le bon fonctionnement des algorithmes demandés.
Quelques questions demandent de compter le nombre d'opérations effectuées par ces algorithmes. La réponse devra être donnée comme une fonction de la taille de l'entrée, et bien entendu justifiée.

Exercice 1 : Deux calculs simples

1) (a) Écrire un algorithme qui, un entier `n` et un entier `k` étant donnés, calcule $\texttt{n}^{\texttt{k}}$. On ne s'occupera pas des éventuels débordements. On rappelle que par convention, $n^0 = 1$, pour tout entier n.

   (b) Combien (en fonction de `k`) de multiplications sont nécessaires pour faire ce calcul ?

2) On rappelle que si p est un entier naturel non nul, $p! = 1 \times 2 \times 3 \times \cdots \times p$. Ainsi, par exemple, $5! = 1 \times 2 \times 3 \times 4 \times 5 = 120$. Par convention, $0! = 1$.

   (a) Écrire un algorithme qui, un entier `n` étant donné, calcule `n`$!$. Encore une fois, on ne s'occupera pas des éventuels débordements.

   (b) Combien de multiplications sont nécessaires pour faire ce calcul ?

Exercice 2 : Plus haute marche

Un tableau `m`, de type `array[1..N] of integer`, contient des entiers positifs, rangés en ordre croissant, représentant les altitudes atteintes par un alpiniste à différents instants d'une escalade. On cherche à savoir à quel moment son ascension a été la plus rapide. Pour cela, on cherche la plus grande différence entre deux cases consécutives du tableau. Par exemple, pour le tableau

```
m = [ 2; 5; 7; 8; 8; 11; 13; 17; 18; 18; 19; 21]
```

l'intervalle au cours duquel l'alpiniste a le plus grimpé est l'intervalle entre les indices 7 et 8, au cours duquel il est passé d'une altitude de 13 m à une altitude de 17 m et a donc grimpé de 4 m. On considère que l'altitude initiale est de 0 m (donc dans notre exemple, l'alpiniste grimpe initialement de 2 m).

1) Écrire un algorithme qui prend en argument le tableau `m` et qui renvoie la *hauteur* maximale d'ascension (la saisie du tableau n'est pas demandée).

2) Combien de comparaisons votre algorithme effectue-t-il pour obtenir la réponse cherchée ? Combien de soustractions ?

3) Modifier l'algorithme précédent de manière à ce qu'il renvoie non pas la hauteur maximale, mais l'*indice* de départ de cette plus grande marche.

EXERCICE 3 : RECHERCHE DANS UN TABLEAU, DANS UNE MATRICE

Dans tout cet exercice, `t` est un tableau de type `array[1..N] of integer`, et `m` une matrice de type `array[1..N,1..M] of integer`.

1) **Exploration itérative du tableau et de la matrice**

   (a) On souhaite compter le nombre de cases du tableau `t` égales à 0. Par exemple, si `t` est le tableau

   `[1; 0; 0; 3; 8; 3; -2; 0; 9; 0; 0; 5]`

   alors la fonction doit renvoyer 5. Écrire un algorithme réalisant ce calcul.

   (b) Combien de comparaisons votre algorithme effectue-t-il ?

   (c) Écrire un algorithme comptant le nombre de cases de la matrice `m` égales à 0. Combien de comparaisons sont-elles effectuées ?

2) **Exploration conditionnelle du tableau et de la matrice**

   (a) On souhaite cette fois-ci vérifier si le nombre de cases du tableau `t` contenant un 0 est supérieur à un entier k fourni par l'utilisateur. Par exemple, avec le tableau ci-dessus, la fonction doit renvoyer `vrai` si `k=3`, mais `faux` si `k=8`.

   Écrire un algorithme réalisant ce calcul. Le résultat attendu est un booléen. On souhaite que l'algorithme arrête ses calculs dès que la réponse est connue. Ainsi, dans l'exemple précédent, si `k=3`, il n'est pas nécessaire d'aller explorer la partie du tableau à partir de `9`.

   (b) Peut-on savoir exactement combien de comparaisons sont effectuées par cet algorithme ? Donner une estimation dans le pire cas de ce nombre.

   (c) Écrire un algorithme similaire renvoyant `vrai` ou `faux` selon que la matrice `m` contient plus ou moins de `k` cases égales à 0.

   Expliquer précisément comment votre boucle conditionnelle est construite pour explorer la matrice.