DS d'informatique n° 4
le Samedi 6 Avril 2024 - durée 2h.

Problème I : En informatique une image de taille $w \times h$ est stockée sous forme de matrice (ou liste à deux dimensions) `Img` de pixels (w pixels de large, h pixels de haut). On a donc une liste `Img` contenant h listes de longueur w.

Pour $0 \leq i < h$ et $0 \leq j < w$, chaque pixel `Img[i][j]` de cette matrice de la liste correspond à un élément de couleur homogène utilisé pour représenter une image sous forme numérique. La teinte d'un pixel en couleur est composée de trois composantes qui correspondent aux couleurs rouge, vert et bleu. Chaque pixel `Img[i][j]` correspond à un triplet d'entiers `[r,g,b]` tous entre 0 et 255. Chacune des trois composantes donne l'intensité de la couleur correspondante dans la teinte finale, il s'agit d'un entier codé sous 8 bits entre 0 et 255 (0 code le noir et 255 code le blanc). Ainsi, le triplet `[0,0,0]` désigne un pixel noir, le triplet `[255,0,0]` désigne un pixel tout rouge, etc.

1. Combien de couleurs différentes peut-on encoder sur un pixel ? Combien d'octets sont nécessaires pour encoder un pixel ?
2. Quelles sont les coordonnées du pixel en bas à gauche de l'image ?
3. Déterminer la taille en Mo (à 10^{-1}) d'une image de résolution 3840×2160 pixels.
4. Ecrire une fonction `dim(Img)` qui renvoie le tupple `(w,h)`.
5. Ecrire une fonction `nouvelle(w,h)` qui renvoie une nouvelle image blanche de taille $w \times h$.
6. Ecrire une fonction `NbreCouleurs(Img)` renvoie le nombre de couleurs différentes contenues dans d'une image.
7. Ecrire une fonction `Negatif(Img)` qui crée une nouvelle image de la même taille en inversant les couleurs. Par exemple, le pixel orange `[255,200,0]` devient bleu `[0,55,255]`.
8. Evaluer les compléxités respectives en temps et en espace de `NbreCouleurs` et `Negatif` en fonction de $N = wh$ le nombre de pixels de l'image.

Problème II : On recherche par plusieurs méthodes indépendantes à calculer les coefficients binomiaux $\binom{n}{k}$ avec $0 \leq k \leq n$.

Dans chacun des cas, on sera vigilant dans les calculs pour que le résultat final soit bien un entier (i.e. de type 'int').

1. Méthode 1 avec la formule $\binom{n}{k} = \frac{n!}{k!(n-k)!}$
   (a) Ecrire une fonction `facto(n)` qui calcul $n!$.
   (b) Ecrire une fonction `binom1(n,k)` qui calcul $\binom{n}{k}$.
   (c) Préciser la complexité temporelle de la fonction.
2. Méthode 2 avec la formule $k\binom{n}{k} = n\binom{n-1}{k-1}$
   (a) Ecrire une fonction `binom2(n,k)` qui calcul $\binom{n}{k}$.
   (b) Préciser la complexité temporelle de la fonction.
3. Méthode 3 avec la formule $\binom{n}{k} = \binom{n-1}{k} + \binom{n-1}{k-1}$
   (a) Ecrire une fonction `ligne(n)` qui calcul la n-ième ligne du triangle de pascal de manière récursive.

(b) En déduire une fonction `binom3(n,k)` et préciser sa complexité temporelle et spatiale.

4. Quelle méthode est la plus efficace ? (Argumenter !)

Problème III : Les livres dispose d'un numéro unique afin de les identifier ISBN (International Standard Book Number)

1. Avant 2007, la norme ISBN-10 était utilisée. Le nombre est composé de 10 chiffres (de 0 à 9). Les 9 premiers permettent d'identifier le livre et le dernier est un chiffre de contrôle. Il permet de repérer une éventuelle erreur de transmission. L'ISBN-10 est valide si la somme des chiffres pondérés par un coefficients décroissant (de 10 à 1) est divisible par 11.

   Par exemple : 030640615c avec c le chiffre de contrôle.
   Le calcul $0\times10+3\times9+0\times8+6\times7+4\times6+0\times5+6\times4+1\times3+5\times2+c\times1=130+c$.
   Donc $c=2$ afin que la somme soit $132=11\times12$ divisible par 11.

   (a) Ecrire une fonction `vers_liste(nb)` qui prend en entré un nombre (de type 'int') et renvoie la liste de ses chiffres.

   (b) Ecrire une fonction `valide10(isbn10)` qui renvoie `True` si l'isbn est valide et `False` sinon.

   (c) Ecrire une fonction `controle10(livre)` qui prend en entré les 9 premiers chiffres sont forme d'une liste `livre` et renvoie la valeur du chiffre de Contrôle.

2. La norme actuelle est ISBN-13. Elle repose sur le même principe avec 12 chiffres d'identification et le dernier de contrôle.
   On réalise désormais la somme alternée avec des coefficients égaux à 1 et 3. Puis la somme doit être divisible par 10.

   Par exemple : 978030640615$c$avec c le chiffre de contrôle.
   $9\times1+7\times3+8\times1+0\times3+3\times1+0\times3+6\times1+4\times3+0\times1+6\times3+1\times1+5\times3+c\times1=93+c$.
   Donc $c=7$ afin que la somme soit 100 divisible par 10.

   (a) Combien de livres pouvaient-on encoder avec l'ancienne norme ? et avec la nouvelle ?

   (b) Ecrire les fonctions `valide13(isbn13)` et `controle13(livre)` qui adaptent les fonctions précédentes à la nouvelle norme.

   (c) Les deux normes coexistent actuellement, écrire une fonction `valide(isbn)` qui prend en entré une ISBN et renvoie `True` si elle est valide et `False` sinon.

Problème IV : Insertion dans une file de priorité.

On suppose que la liste `l` est une liste d'entiers déjà triées et on écrit les fonctions suivantes :

```
def fct1(l:list, a:int):
        r = 0
        for x in l:
                if x<a:
                        r += 1
                else:
                        break
        return l[:r]+[a]+l[r:]
```

```
def fct2(l:list, a:int):
        i = len(l)-1
        l.append(l[i])
        while i>0 and a<l[i-1]:
            l[i] = l[i-1]
                i-=1
        l[i] = a
        return l
```

1. (a) Exécuter les fonctions pour `l=[1,5,7,8]` et `a=6`, on présentera sous forme d'un tableau l'évolution des variables au fil de la boucle.
   (b) Que renvoie les fonctions `fct1` et `fct2` ?
   (c) Expliquer la différence entre les instructions `l=l+[a]` et `l.append(a)` ?
   (d) Montrer la complexité temporelle et spatiale de `fct1` est linéaire en $N = len(l)$.
2. On étudie la boucle 'while' de `fct2`.
   (a) Montrer que le variant est `i`. En déduire la complexité de la fonction. Quel est le pire des cas et le meilleur des cas ?
   (b) Montrer que l'invariant est 'la liste est triée et contient tous les éléments de l'. En déduire la validité du programme.
3. Une file de priorité est une liste qui doit rester triée en permanence. On doit pouvoir extraite l'élément le plus grand et ajouter un élément quelconque rapidement.
   (a) Ecrire une fonction `tri(file:list)` qui part d'une tri une liste (désordonnée). Préciser le nom de l'algorithme et sa complexité.
   (b) Pourquoi le code suivant n'est pas efficace :

```
def insertion(file:list, a:int):
        file.append(a)
        tri(file)
```

   (c) Comment peut-on extraire et ajouter efficacement un élément de la file de priorité.
4. (Question subsidiaire) Des élèves de classe préparatoire recevoir des devoirs (tous les jours) mais dont l'échéance et la durée peut varier d'un professeur à l'autre. Comment pourrait-on utiliser les files de priorité pour aider ces élèves ?