DS 2

Informatique pour tous, première année

Julien Reichert

On signale pour l'exercice 1 que les nombres écrits en virgule flottante sur 32 bits ont huit bits d'exposant.

Pour les exercices de programmation, quand l'énoncé précise ce que la fonction à écrire doit prendre en entrée, il faut tout d'abord respecter le nombre et le type des arguments, mais aussi ne pas s'encombrer de vérifications dans le code de la fonction, on peut admettre qu'il n'y aura pas d'instances dégénérées.

Exercice 1 : Écrire le nombre $\frac{100}{17}$ en virgule flottante sur 32 bits.

Exercice 2 : Écrire en Python une fonction `century` qui prend en entrée un entier compris entre `1` et `9999`, représentant une année, et qui retourne une chaîne de caractères indiquant le numéro en anglais du siècle de l'année en question.

Par exemple, l'appel `century(2021)` renverra la chaîne `"21st"` car 2021 est au vingt-et-unième siècle, et l'appel `century(1800)` renverra `"18th"` car 1800 était au dix-huitième siècle. La connaissance des abréviations des ordinaux en anglais fait **aussi** partie de l'évaluation.

Exercice 3 : Écrire en Python une fonction `filtre` qui prend en entrée une liste de nombres et retourne la liste contenant les mêmes éléments, mais réordonnées de sorte que les nombres strictement négatifs apparaissent en premier, et dans l'ordre d'apparition dans la liste, puis les valeurs nulles, puis les nombres positifs également dans l'ordre d'apparition dans la liste.

Par exemple, l'appel `filtre([5, -3, 0, 2, -4, 0, 3, -2])` renverra la liste `[-3, -4, -2, 0, 0, 5, 2, 3]`.

Exercice 4 : On considère une liste de `n` couleurs, toutes égales à rouge, vert ou bleu, et représentées par les chaînes de caractères `"R"`, `"V"` et `"B"`. Cette liste subit la transformation suivante : pour les `n-1` couples de couleurs voisines, chacune sauf les extrémités apparaissant donc dans deux couples, on construit la liste des couleurs obtenues suivant la règle que deux couleurs identiques se transforment en cette couleur et deux couleurs différentes se transforment en la couleur absente. Faire cette transformation `n-1` fois donne une liste finale de taille `1`, dont on relève la couleur. Écrire une fonction `fusion(l)` prenant en argument une telle liste et renvoyant la couleur finale.

Par exemple, l'appel `fusion(["R", "V", "B"])` donne la chaîne de caractères `"V"` car la première transformation change le couple `("R", "V")` en `"B"` et le couple `("V", "B")` en `"R"`, d'où la liste intermédiaire `["B", "R"]`, dont la transformation donne la liste `["V"]`.

Exercice 4 bis : Déterminer et prouver la complexité asymptotique de la fonction `fusion` en nombre d'opérations élémentaires, sachant qu'on compte pour une opération élémentaire l'accès à un élément d'une liste, l'ajout d'un élément à une liste et la modification d'un élément d'une liste.

Attention aux copies de listes qui ont un coût linéaire, de même que les tranches qui ont un coût linéaire en la taille de la tranche obtenue.

L'exercice 5 a pour thème le paradoxe de Von Mises, aussi appelé « paradoxe des anniversaires ». La partie mathématique reposant sur une partie de cours n'ayant pas encore été traitée, toutes les formules seront données.

Selon ce paradoxe, en admettant que le jour de l'année où une personne est née peut être n'importe quel jour de l'année avec la même probabilité (on exclut pour faciliter le 29 février, cela rend l'hypothèse plus acceptable), il suffit de considérer vingt-trois personnes prises au hasard pour qu'il y ait plus d'une chance sur deux qu'au moins deux personnes aient leur anniversaire en commun.

La formule donnant la probabilité que $k \geq 1$ personnes soient nées un jour différent de l'année est précisément :

$$\bar{p}(k) = \frac{\prod_{i=0}^{k-1}(365 - i)}{365^k}$$

et on détermine la plus petite valeur de k telle que $\bar{p}(k) \leq \frac{1}{2}$, cette valeur étant effectivement 23 [1].

On considère pour les trois premières questions que la liste des $p(k)$, où on définit $p(k) = 1 - \bar{p}(k)$ pour k de 1 à 366 est fournie, de sorte que son coût de construction ne soit pas pris en compte.

Question 5.1 : Quelle propriété, condition pour qu'une recherche dichotomique soit pertinente, cette liste a-t-elle ?

Question 5.2 : Trois versions d'une recherche dichotomique sont proposées. L'une n'a pas de terminaison garantie, une autre donne parfois un résultat incorrect, une enfin est bonne. Déterminer laquelle est laquelle en prouvant la terminaison des deux qui terminent et la correction en cas de terminaison des deux qui sont correctes, ainsi qu'en exhibant un cas de non terminaison pour celle qui ne termine pas et un cas d'échec pour celle qui n'est pas correcte.

```
def dicho(l):
    deb, fin = 0, len(l)-1
    while deb < fin:
        mil = (deb + fin)//2
        if l[mil] < .5:
            deb = mil+1
        else:
            fin = mil-1
    return deb
```

```
def dicho2(l):
    deb, fin = 0, len(l)-1
    while deb < fin:
        mil = (deb + fin)//2
        if l[mil] == .5:
            return mil
        elif l[mil] < .5:
            deb = mil
        else:
            fin = mil
    return deb
```

```
def dicho3(l):
    deb, fin = 0, len(l)-1
    while deb < fin:
        mil = (deb + fin)//2
        if l[mil] < .5:
            deb = mil+1
        else:
            fin = mil
    return deb
```

Question 5.3 : Une quatrième version est proposée ci-après. Quel est son problème ?

```
# On suppose ici que l[i] est un couple (i, p(i)) pour tout indice i
def dicho(l): # Au début l[0] vaut 0 et l[-1] vaut 1
    while len(l) > 2: # Il restera les deux valeurs encadrant .5, en cas d'égalité ce sera la deuxième
        mil = len(l)//2 # milieu de la liste, au moins un élément à droite et au moins un à gauche
        if l[mil][1] < .5: # on compare p(i)
            l = l[mil:] # jusqu'à mil inclus c'est inférieur strict
        else:
            l = l[:mil+1] # ça devient supérieur en mil ou avant
    return l[-1][0] # la valeur originale de i
```

Question 5.4 : Écrire une fonction `seuil(n)` généralisant le cas du paradoxe des anniversaires à une propriété pouvant prendre `n` valeurs différentes en admettant une équirépartitions et déterminant le seuil au-delà duquel la probabilité d'avoir deux valeurs identiques dépasse $\frac{1}{2}$.

Il s'agit donc de calculer les valeurs de $p_n(k)$ ou de $\bar{p}_n(k)$ jusqu'à atteindre ou dépasser le seuil fixé. La complexité devra être linéaire (pas à prouver).

1. elle serait la même avec 366 jours