TP1 Révisions

1 Généralités

Exercice 1 On définit la liste `L=[10, 30, 42, 2, 17, 5, 30, -20]`.

1. Créer la liste `L1` constituée des termes d'indice pair de `L`.
2. Créer la liste `L2` constituée des termes pairs de `L`.

Exercice 2 Écrire une fonction `doublon` prenant en entrée une liste `L` non vide, et renvoyant `True` si la liste possède un élément répété au moins une fois, et `False` sinon.

```
def doublon(L):
....
>>> doublon( [10,2,5,2,42])
True
>>> doublon( [10,2,5,3,42])
False
```

Exercice 3

a) Justifier que $2^{2^{2^2}} = 65536$.

b) Écrire une fonction `puissance_iteree` (version récursive et version itérative) prenant en entrée deux entiers naturels k et n et retournant la $n^{ième}$ puissance itérée de k. Ainsi le nombre $k^{k^{\cdot^{\cdot^{\cdot^{k}}}}}$ formé de n exemplaires de k s'obtient par exécution de `puissance_iteree(k,n)`.

c) Vérifier vos résultats en exécutant `puissance_iteree(2,4)`.

Exercice 4 (facultatif pour ceux qui ont un peu d'avance)

1. Que fait le code *Python* suivant ? Quelle est l'une de ses particularités ?

```python
def ordonner(L) :
    if len(L) <= 1 :
        return L
    e0 = L[0]
    n = 1
    Linf, Lsup = [], []
    for ei in L[1:]:
        if ei == e0 :
            n += 1
        elif ei < e0 :
            Linf.append(ei)
        else :
            Lsup.append(ei)
    return ordonner(Linf)+n*[e0]+ordonner(Lsup)
```

2. Écrire une fonction `less` de deux nombres complexes `z1` et `z2` qui renvoie `True` si et seulement si $\mathcal{R}e(z_1) < \mathcal{R}e(z_2)$ ou "$\mathcal{R}e(z_1) = \mathcal{R}e(z_2)$ et $\mathcal{I}m(z_1) < \mathcal{I}m(z_2)$".

3. En s'inspirant du code définissant la fonction `ordonner`, écrire puis tester une fonction `ordonnerDansC` qui ordonne une liste de complexes selon la relation d'ordre définie précédemment.

4. En utilisant le module `cmath`, créer la liste des $(20 + \cos(10a))\exp(ia)$, pour a variant de $-\pi$ à π avec un pas de $\pi/100$. Ordonner ces points par `ordonnerDansC`. Faire tracer dans le plan complexe le nuage de points ainsi que la ligne les reliant. On utilisera `plot`, `axis` et `show` du module `matplotlib.pyplot` et le repère sera rendu orthonormé par `axis('equal')`.

2 Transmission de données

2.1 Bit de parité

Le bit de parité est une technique simple pour s'assurer qu'une donnée transmise sous la forme d'un mot binaire sera lue correctement par son récepteur. Celui-ci est défini par :

- 0 si la donnée contient un nombre pair de 1 (ie si ses bits sont de somme paire),
- 1 si la donnée contient un nombre impair de 1 (ie si ses bits sont de somme impaire).

Après réception de la donnée, le récepteur recalcule le bit de parité associé et le compare à celui que l'émetteur lui a adressé. Si les bits de parité diffèrent, la donnée a été altérée lors de la transmission.

Exercice 5

1. Donner les bits de parité associés aux représentations binaires de 5, 16 et 37.

2. Écrire une fonction `parite` prenant en argument une liste `bits` constituée de 0 et de 1 et retournant le bit de parité correspondant.

3. Après réception de la donnée, le récepteur recalcule le bit de parité associé et le compare à celui que l'émetteur lui a adressé : ceux-ci coïncident. Qu'en conclure ?

2.2 Code de Hamming

Le code de Hamming [7,4] consiste à joindre trois bits de parité à quatre bits de données. Si les quatre bits de données sont $[d_1, d_2, d_3, d_4]$, les trois bits de parité $[p_1, p_2, p_3]$ sont définis par :

- p_1 est le bit de parité du triplet (d_1, d_2, d_4),
- p_2 est le bit de parité du triplet (d_1, d_3, d_4),
- p_3 est le bit de parité du triplet (d_2, d_3, d_4).

Le message transmis sera $[p_1, p_2, d_1, p_3, d_2, d_3, d_4]$.

Exercice 6

1. Écrire une fonction `code_Hamming` prenant en argument une liste `d` de quatre bits et retournant la liste des sept bits obtenue après encodage du message.

2. Définir une fonction `altere` qui simule une altération lors de la transmission du message. Cette fonction prend en argument une liste `c` d'un nombre quelconque de bits, et renvoie la liste où l'un des bits, choisi de manière équiprobable, est modifié avec la probabilité 3/4.

Pour envoyer un mot `d`, on transmet `c=code_Hamming(d)` mais le correspondant reçoit `x=altere(c)`. À partir d'une liste de sept bits reçue $x = [x_1, x_2, ..., x_7]$, la technique proposée par Hamming pour contrôler la transmission est de calculer les trois bits de contrôle (q_1, q_2, q_3), appelés le syndrome, définis par :

- q_1 est le bit de parité de $[x_4, x_5, x_6, x_7]$,
- q_2 est le bit de parité de $[x_2, x_3, x_6, x_7]$,
- q_3 est le bit de parité de $[x_1, x_3, x_5, x_7]$.

On peut montrer que :

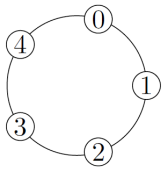
- Si le syndrome est nul, ie $(q_1, q_2, q_3) = (0, 0, 0)$, alors le message n'est pas altéré.
- S'il n'est pas nul, il indique la position (entre 1 et 7) du bit où l'erreur est apparue. Par exemple, si le syndrome est $(0, 1, 1)$, l'erreur porte sur le $\underline{011}_2 = 3$-ième bit du message.

Exercice 7

1. Écrire une fonction `decode` d'argument une liste `x` de sept bits, qui corrige le cas échéant l'erreur apparue sur un bit, et renvoie le mot `d` de quatre bits contenant la donnée décodée.
2. On suppose que `d=[1,0,1,1]` mais que, lors de la transmission, les deux premiers bits du message transmis `c` ont été modifiés. Quel est l'effet de la correction dans ce cas ?

3 Pour ceux qui ont tout fini

Exercice 8 n personnes numérotées de 0 à $(n-1)$ se mettent en cercle, comme le montre la figure suivante pour $n = 5$:



En commençant par la personne numéro 1 et en tournant dans le sens des numéros croissants (sens horaire sur la figure), on retire une personne sur deux, en ne prenant en compte que les personnes restant dans le cercle. Par exemple, pour $n = 5$, on retirera successivement les personnes 1, 3, 0 puis 4; il restera la personne 2.

Pour simuler cette procédure d'élimination progressive, on décrit le cercle par une liste de n booléens : la valeur numéro i est `True` si la personne i est éliminée, et `False` si elle est encore dans le cercle.

Au départ, la liste ne contient que des `False` puis ses valeurs passent progressivement à `True`, jusqu'à ce qu'il ne reste plus qu'un seul `False`.
Ainsi, pour $n = 5$, la liste passe par les états successifs :

Liste de booléens `E`	Rang `p` du dernier éliminé
`[ False , True , False , False , False ]`	1
`[ False , True , False , True , False ]`	3
`[ True , True , False , True , False ]`	0
`[ True , True , False , True , True ]`	4

Il reste 2

1. Écrire une fonction `suivant` de deux arguments, une liste `E` de n booléens et un entier `p` entre 0 et $(n-1)$ qui renvoie la position `q` du premier `False` rencontré en partant de la position juste après la position `p`, en parcourant la liste de façon circulaire.
   Par exemple :

   `suivant([True,True,False,True,False],0)` donne 2;
   `suivant([True,True,False,True,False],2)` donne 4;
   `suivant([True,True,False,True,False],4)` donne 2.

2. Se servir de la fonction `suivant` pour simuler le cas $n = 5$ décrit dans le tableau ci-dessus.

3. Écrire une fonction `reste` d'argument un entier naturel non nul n qui renvoie le numéro de la dernière personne restante parmi n personnes. Tester `reste` pour $n = 16$, $n = 31$, $n = 65$.

4. Pour $2 \leqslant n \leqslant 140$, afficher n suivi du numéro du dernier restant.
   En observant le résultat, que peut-on conjecturer sur le calcul du dernier restant ?