Correction du DS 1

Informatique pour tous, deuxième année

Julien Reichert

Exercice 1

Voir le cours et le TP 1, les deux implémentations, naturelle et à l'aide de tableaux, étaient acceptées. Toute autre éventuelle implémentation cohérente pouvait également être acceptée.

Exercice 2

Les chiffres dans l'écriture en base 3 s'obtiennent en prenant le reste dans la division euclidienne par 3 et en répétant en faisant à chaque fois le quotient par 3 (rappel du chapitre 1 de première année).

On écrit donc simplement :

```
def pas_de_deux(n):
    if n < 0:
        return pas_de_deux(-n)
    if n < 3:
        return n != 2
    return n % 3 != 2 and pas_de_deux(n // 3)
```

Exercice 3

On utilise uniquement des opérations élémentaires (piles illimitées a priori au vu de la fonction de création).

```
def i_eme(p,i):
    """Renvoie le i-ième élément en partant du fond (pour lequel i vaut 1)."""
    p2 = creer_pile()
    while not(est_vide(p)):
        empiler(p2,depiler(p))
    k = 0
    while k < i and not(est_vide(p2)):
        empiler(p,depiler(p2))
        k += 1
    if k < i:
        raise ValueError("Pile trop courte.")
    reponse = sommet(p)
    while not(est_vide(p2)):
        empiler(p,depiler(p2))
    return reponse
```

Exercice 4

Il s'agit d'une extension d'un exercice du TP 3. On a ici besoin de connaître le contenu des piles. Une solution intuitive revient à mettre en argument les piles. À ce stade, il est plus simple de considérer celles-ci comme des

variables globales (favoriser le langage par rapport au paradigme de programmation), et donc la fonction ne sera pas elle-même récursive mais appellera une sous-fonction récursive après une préparation des variables.

```
def hanoi_explicite(n):
    piles = [list(range(n,0,-1)), [], []]
    print ("Départ :",piles[0],piles[1],piles[2])
    def hanoi_aux(i,origine,arrivee):
        if i > 0:
         transfert = 3-origine-arrivee
         hanoi_aux(i-1,origine,transfert)
         print(origine+1,"->",arrivee+1)
         piles[arrivee].append(piles[origine].pop())
         print ("Piles :",piles[0],piles[1],piles[2])
         hanoi_aux(i-1,transfert,arrivee)
    hanoi_aux(n,0,2)
```

Exercice 5

D'après la formule du cours, on a $c_n = ac_{\frac{n}{b}} + \mathcal{O}(n^\alpha)$ avec $\log_b(a) = \alpha = 0$. Ainsi, $c_n = \mathcal{O}(n^0 \log n)$ soit une complexité logarithmique. C'est le cas d'une dichotomie simple.

Exercice 6

C'est la combinaison de deux exercices du TP 2 :

```
def multiparenthesage(L):
    parentheses = []
    couples = []
    for i in range(1,len(L)+1):
        if L[-i] < 0:
            parentheses.append((L[-i],len(L)-i))
        elif L[-i] > 0:
            try:
                (type,position) = parentheses.pop()
                assert(type == -L[-i])
                couples.append((len(L)-i,position))
            except:
                raise ValueError("Mauvais parenthesage")
    if parentheses == []:
        return list(reversed(couples))
    else:
        raise ValueError("Une parenthese fermante au moins est de trop")
```

On notera que la lecture de la liste se fait à l'envers. La raison est qu'alors les couples seront ordonnés par position croissante des parenthèses ouvrantes.