Correction du DS 1

Informatique pour tous, deuxième année

Julien Reichert

Exercice C-1

Cf. cours pour les deux implémentations acceptées, d'éventuelles autres implémentations peuvent être considérées.

Exercice C-2

D'après la formule du cours, $c(2n) = c(n) + n$, donc $c(2^k) = 2^{k-1} + 2^{k-2} + \cdots + 2 + 1 + 1 = 2^k$, d'où $c(n) = \mathcal{O}(n)$ pour un n quelconque.

```
def fonction_artificielle(n):
    def artif(n, cptr):
        assert n > 0, "n non strictement positif"
        if n == 1:
            return cptr ** 2
        else:
            for i in range(n//2, n):
                cptr += 42
            return artif(n//2, cptr)
    return artif(n, 0)
```

Vous l'aurez deviné, cette fonction renvoie $(42 * n)^2$.

Exercice A-1

On a besoin de deux piles supplémentaires, car avec une seule pile on ne peut que transférer des objets pour accéder à un élément quelconque sans pouvoir perturber l'ordre.

L'algorithme est simple : vider la pile dans une pile supplémentaire, puis vider cette dernière dans la dernière pile, puis vider celle-ci dans la pile de départ, elle sera alors « retournée ». Cela fait donc trois fois autant d'empilements et dépilements que d'éléments dans la pile au départ.

```
def retourne(p):
    p1 = creer_pile(len(p)-1)
    p2 = creer_pile(len(p)-1)
    while not (est_vide(p)):
        empiler(p1,depiler(p))
    while not (est_vide(p1)):
        empiler(p2,depiler(p1))
    while not (est_vide(p2)):
        empiler(p,depiler(p2))
    # return p # (accepté, mais ce n'est pas dans la spécification)
```

Exercice A-2

```
def euclide(p, q):
    p, q = abs(p), abs(q)
    if p == 0 and q == 0:
        raise ValueError("p et q sont tous les deux nuls")
    elif p == 0 or q == 0:
        return max(p, q)
    else:
        return euclide(q, p % q) # Si p < q, cela échange les deux

def bezout(p, q):
    if p < 0:
        (u, v) = bezout(-p, q)
        return (-u, v)
    if q < 0:
        (u, v) = bezout(p, -q)
        return (u, -v)
    if p == 0 and q == 0:
        raise ValueError("p et q sont tous les deux nuls")
    elif p < q: # ceci n'arrive qu'une fois au plus, tout au début
        u, v = bezout(q, p)
        return v, u
    elif q == 0:
        return (1, 0)
    else:
        (u, v) = bezout(q, p % q) # u q + v (p mod q) = pgcd(p,q)
        r = p // q # u q + v (p - rq) = pgcd(p,q)
        return (v, u-v*r) # v p + (u-vr) q = pgcd(p,q)
```

Exercice A-3

```
def gare_triage(jetons):
    npi, pile_ops = [], []
    for jeton in jetons:
        if type(jeton) == int:
            npi.append(jeton)
        elif jeton in ["+", "-"]:
            while pile_ops != [] and pile_ops[-1] != "(":
                npi.append(pile_ops.pop())
            pile_ops.append(jeton)
        elif jeton in ["*", "/"]:
            while pile_ops != [] and pile_ops[-1] in ["*", "/"]:
                npi.append(pile_ops.pop())
            pile_ops.append(jeton)
        elif jeton == ")":
            while pile_ops != [] and pile_ops[-1] != "(":
                npi.append(pile_ops.pop())
            pile_ops.pop()
        else: # jeton == "("
            pile_ops.append(jeton)
    while pile_ops != []:
        op = pile_ops.pop()
        npi.append(op)
    return npi
```

Exercice A-4

Premier puzzle (numéro 24)

F1 : ↑↷ F2

F2 : ↑↶ F1

Deuxième puzzle (numéro 39)

F1 : F3 F2 F3 F3 F2 F2 F3 F2 F2 F3

F2 : ↑↑↷

F3 : ↑↑↶

On peut finir F1 par F2, le tout est de ne pas oublier de lancer une fonction qui avance encore. L'avantage est que le nombre d'instructions allouées permet d'éviter cette omission.

Troisième puzzle (numéro 539)

F1 : ↑ ↶ F2 F1

F2 : ↶↶ F3

F3 : ↑ ↷ F3

D'après le site, il existe une solution avec sept instructions.

Quatrième puzzle (numéro 648)

Une fonction est inutile, et dans l'idée on se sert de trois fonctions pour simuler une fonction avec trois instructions plus un appel récursif à la fonction elle-même :

F1 : ↶ F2

F2 : ↑ F3

F3 : ↶ F4

F4 : ↶ F2

Cinquième puzzle (numéro 587)

Certaines instructions sont automatiques, ensuite il reste à broder :

F1 : ↶ ↑ F2 F1

F2 : ↑↷↷↑

Autre solution :

F1 : ↑ ↶ F2 F1

F2 : ↑ ↷↷ F2

D'après le site, il existe une solution avec sept instructions.

Sixième puzzle (numéro 656)

C'était le puzzle le plus compliqué. En pratique, les colonnes étaient de taille si chaotique qu'on devait se douter que ces tailles n'avaient pas d'importance (peut-être y a-t-il un message codé, ce serait amusant !). Ainsi, il fallait empiler des instructions d'avancement mises en attente par des appels récursifs prioritaires. À l'instar du premier TP de première année, la question qu'on peut se poser est « Comment passer de la configuration de départ à la même configuration mais un cran en avant et en ayant visité la colonne ? », et la réponse est par la fonction F1 dont la partie concernant la colonne se fait dans F2, au rétablissement de l'orientation près.

F1 : ↶ F2 ↶↑ F1

F2 : ↑ F2 ↶↶ ↑

L'ordre des trois instructions du milieu de F2 n'est pas important.

D'après le site, il existe une solution avec neuf instructions.