Correction du DS 4

Informatique de tronc commun, première année

Julien Reichert

Exercice 1

```
def nombre_inversions(l):
    rep = 0
    for i in range(len(l)-1):
        for j in range(i+1, len(l)): # attention à ne pas démarrer à 0
            if l[i] > l[j]: # sinon il faut aussi tester i < j
                rep += 1
    return rep
```

Exercice 2

```
def decompose(perm):
    n = len(perm)
    places = [False] * n # Plus pratique qu'une liste où on teste l'appartenance
    i = 0
    rep = []
    while True:
        while i < n and places[i]:
            i += 1
        if i == n:
            return rep
        courant = [i]
        places[i] = True
        j = i
        while perm[j] != i:
            j = perm[j]
            courant.append(j)
            places[j] = True
        if len(courant) > 1: # Pas les cycles triviaux
            rep.append(courant)
        i += 1
    return rep
```

Exercice 3

```
def rectangles_milieu(fct, a, b, d):
    rep = 0
    point = a
    i = 1
    while point + d <= b:
        rep += d * fct(point + d/2)
        point = a + i * d # ne pas faire des additions qui accumulent les erreurs d'approximation
        i += 1
    rep += (b - point) * fct((point + b) / 2) # dernier rectangle, éventuellement vide
    return rep
```

Exercice 4

Question 4.1

Attention, pour la preuve de cet exercice comme pour la question 4.3 il faut penser au cas de la puissance nulle.

Initialisation : la puissance nulle de la matrice d'adjacence est l'identité, et les chemins de taille nulle reviennent à ne pas bouger, justifiant les zéros partout sauf la diagonale à un. C'est par ailleurs trivialement vrai pour la matrice elle-même car un chemin de taille un est réduit à un arc.

Hérédité : un chemin de taille $k+1$ est vu comme le prolongement d'un chemin de taille k en ajoutant un arc. Ainsi, tous les chemins de u à v de taille $k+1$ sont donc des chemins de u à un certain w de taille k prolongés par un arc de w à v, à supposer que cet arc existe. Leur nombre est donc la somme des nombres de chemins de taille k partant de u à tous les sommets pour lesquels l'arc existe, soit encore la somme des nombres de chemins de taille k partant de u à tous les sommets w, chacun multiplié par 1 si l'arc (w, v) existe et 0 sinon, nous ramenant à la formule qui donne le produit $M^k \times M$ évalué en (u, v) en utilisant l'hypothèse de récurrence.

Question 4.2

Attention lors de la création d'une matrice à ne pas créer des lignes toutes dépendantes l'une de l'autre !

```
def produit(M, N):
    n = len(M)
    rep = [[0 for j in range(n)] for i in range(n)]
    for i in range(n):
        for j in range(n):
            for k in range(n):
                rep[i][j] += M[i][k] * N[k][j]
    return rep
```

Question 4.3

Pas besoin de penser à l'exponentiation rapide vue au TP 5 ici.

```
def puissance(M, k):
    n = len(M)
    rep = [[int(i == j) for j in range(n)] for i in range(n)] # identité
    for i in range(k):
        rep = produit(rep, M)
    return rep
```

Question 4.4

Il s'agit d'additionner pour tout entier `k` entre `0` et une certaine valeur `seuil` tous les éléments diagonaux de la puissance `k`-ième de la matrice d'adjacence d'une matrice (vu que c'est le contexte). D'après ce qui précède, on déduit que la fonction calcule le nombre de circuits de taille au plus `seuil` dans le graphe représenté.

Question 4.5

Le produit matriciel naïf fait ici est cubique en la taille de la matrice. Comme il est répété `k` fois quand on calcule la puissance `k`-ième, on est en $\mathcal{O}(n^3 k)$ pour l'appel `puissance(M, k)` avec `M` de taille `n`. Or la fonction `puissance` est appelée à chaque tour de la boucle pour une valeur de `k` augmentant d'un étape par étape. La complexité finalement est en $\mathcal{O}(n^3 \times seuil^2)$, ce qui aurait dû être limité à $\mathcal{O}(n^3 \times seuil)$ en calculant la trace à la volée à chaque multiplication de la matrice.