Correction du DS 5

Informatique de tronc commun, première année

Julien Reichert

Exercice 1

```
def tous_indices(ll):
    return [(i, j) for i in range(len(ll)) for j in range(len(ll[i]))]
```

Exercice 2

```
def monotone(l):
    j = 1
    while j < len(l) and l[j] == l[j-1]:
        j += 1
    i = j
    while j < len(l) and (l[j] == l[j-1] or (l[j] > l[j-1]) == (l[i] > l[i-1])):
        j += 1
    return j == len(l)
```

Exercice 3

On retourne tous les éléments à égalité ici.

```
def carac_plus_frequent(s): # O(len(s)) avec mensonge
    dict = {}
    for car in s: # O(len(s)) avec mensonge
        if car not in dict: # O(1) avec mensonge
            dict[car] = 1 # O(1)
        else:
            dict[car] += 1 # O(1)
    les_max = []
    maxi = 0
    for car in dict: # O(k) où k est le nombre d'éléments différents de s, O(len(s)) en particulier
        if dict[car] > maxi: # O(1)
            maxi = dict[car] # O(1)
            les_max = [car] # O(1)
        elif dict[car] == maxi: # O(1)
            les_max.append(car) # O(1)
    return les_max # éventuellement les_max[0] si la taille est un, mais éviter deux types possibles
```

Exercice 4

```
def plus_petit_ecart(l):
    assert len(l) >= 2, "Liste trop courte"
    mini = abs(l[1] - l[0])
    for i in range(2, len(l)):
        if abs(l[i] - l[i-1]) < mini:
            mini = abs(l[i] - l[i-1])
    return mini
```

Exercice 5

Question 5.1

```
def pppd(n): # Plus petite puissance de dix
    rep = 1
    while rep <= n:
        rep *= 10
    return rep
```

Question 5.2

```
def encodage(g):
    g2 = []
    dpk = pppd(len(g)) # dix puissance k
    for liste in g:
        liste2 = []
        for (v, p) in liste:
            liste2.append((v, dpk * p + 1))
        g2.append(liste2)
    return g2
```

Question 5.3

```
def dijkstra(g, s): # Complexité O(n²) en notant n le nombre de sommets et m le nombre d'arcs
    distances = [None] * len(g) # O(n)
    distancesbis = {} # O(1) # version de travail
    distances[s] = 0 # O(1)
    distancesbis[s] = 0 # O(1)
    while distancesbis != {}: # O(n² + m) soit O(n²)
        mini, le_mini = None, None # O(1)
        for sommet in distancesbis: # O(n)
            if mini is None or distancesbis[sommet] < mini: # O(1)
                mini = distancesbis[sommet] # O(1)
                le_mini = sommet # O(1)
        distancesbis.pop(le_mini) # O(1) # Retirer la clé, retourne la valeur mais on l'a déjà
        for (t, p) in g[le_mini]: # O(m) mais en fait O(len(g[le_mini])) et la somme fait O(m)
            if distances[t] is None or distances[t] > mini + p: # O(1)
                distances[t] = mini + p # O(1)
                distancesbis[t] = mini + p # O(1)
    return distances # O(1)
```

Question 5.4

```
def optimal_et_court(g, s, t):
    g2 = encodage(g)
    dpk = pppd(len(g))
    distances_s = dijkstra(g2, s)
    d_s_t = distances_s[t]
    return d_s_t // dpk, d_s_t % dpk
```

Question 5.5

Sur le principe, tout fonctionne de la même façon, mais l'encodage des poids est changé : au lieu d'avoir $p \mapsto 10^k p+1$, on peut faire $p \mapsto 10^h + p$ avec 10^h la plus petite puissance de dix strictement supérieure à la somme de tous les poids des arcs, de sorte de ne pas avoir là non plus de problèmes de débordement. On peut évidemment raffiner la valeur de 10^h (par exemple la somme des poids maximaux par liste d'adjacence), mais l'intérêt est limité.