corrigé

Algorithmique

Recherche séquentielle dans un tableau uni-dimensionnel

```
def maximum(t):
    m = t[0]
    for x in t:
        if x > m:
            m = x
    return m
```

```
def secondMax(t):
    m1, m2 = max(t[0], t[1]), min(t[0], t[1])
    for x in t:
        if x > m1:
            m1, m2 = x, m1
        elif x > m2:
            m2 = x
    return m2
```

```
def distanceMin(t):
    m = abs(t[1] - t[0])
    for i in range(len(t)):
        for j in range(i + 1, len(t)):
            if abs(t[i] - t[j]) < m:
                m = abs(t[i] - t[j])
    return m
```

```
def comptage(txt):
    d = {}
    for c in txt:
        if c not in d:
            d[c] = 1
        else:
            d[c] += 1
    return d
```

```
def chercheMot(txt, mot):
    compte = 0
    for s in range(len(txt) - len(mot)):
        b = True
        for i in range(len(mot)):
            if txt[s + i] != mot[i]:
                b = False
        if b:
            compte += 1
    return compte
```

Algorithmes de tri

```
def fusion(t1, t2):
    t = []
    i, j = 0, 0
    while i + j < len(t1) + len(t2):
        if j == len(t2) or i < len(t1) and t1[i] < t2[j]:
            t.append(t1[i])
            i += 1
        else:
            t.append(t2[j])
            j += 1
    return t
```

```
def triFusion(t):
    n = len(t)
    if n < 2:
        return t[:]
    t1 = triFusion(t[:n // 2])
    t2 = triFusion(t[n//2:])
    return fusion(t1, t2)
```

```
def dichotomie(x, t):
    i, j = 0, len(t)
    while i + 1 < j:
        k = (i + j) // 2
        if x < t[k]:
            j = k
        else:
            i = k
    return t[i], t[j]
```

Théorie des graphes

```
def degreEntrant(g, i):
    d = 0
    for j in range(len(g)):
        for k in g[j]:
            if k == i:
                d += 1
    return d
```

```
def transpose(g):
    gt = [[] for _ in range(len(g))]
    for i in range(len(g)):
        for j in g[i]:
            gt[j].append(i)
    return gt
```

```python
def distance(g, i, j):
    n = len(g)
    dejaVu = [False for _ in range(n)]
    dist = [None for _ in range(n)]
    dejaVu[i] = True
    dist[i] = 0
    aTraiter = [i]
    while len(aTraiter) > 0:
        s = aTraiter.pop(0)  # parcours en largeur
        for k in g[s]:
            if k == j:
                return dist[s] + 1
            elif not dejaVu[k]:
                dejaVu[k] = True
                dist[k] = dist[s] + 1
                aTraiter.append(k)
```

```python
def composantes(g):
    n = len(g)
    dejaVu = [False for _ in range(n)]
    nb_comp = 0
    for i in range(n):
        if not dejaVu[i]:
            nb_comp += 1
            dejaVu[i] = True
            aTraiter = [i]
            while len(aTraiter) > 0:
                s = aTraiter.pop()  # parcours en profondeur
                for k in g[s]:
                    if not dejaVu[k]:
                        dejaVu[k] = True
                        aTraiter.append(k)
    return nb_comp
```

```python
def chercheMin(dist, dejaVu):
    dmin = float('inf')
    for i in range(len(dist)):
        if not dejaVu[i] and dist[i] <= dmin:
            s, dmin = i, dist[i]
    return s
```

```python
def distance(g, i, j):
    n = len(g)
    dist = [float('inf') for _ in range(n)]
    dejaVu = [False for _ in range(n)]
    dist[i] = 0
    while True:
        print(dist)
        s = chercheMin(dist, dejaVu)
        if s == j:
            return dist[j]
        dejaVu[s] = True
        for k in range(n):
            dist[k] = min(dist[k], dist[s] + g[s][k])
```

Programmation dynamique

On a $p(i,j) = \begin{cases} t(0,0) & \text{si } (i,j) = (0,0) \\ t(0,j) + p(0,j-1) & \text{si } i = 0 \\ t(i,0) + p(i-1,0) & \text{si } j = 0 \\ t(i,j) + \min\big(p(i-1,j), p(i,j-1)\big) & \text{sinon} \end{cases}$

```
def poids(t):
    n = len(t)
    p = [[0 for _ in range(n)] for _ in range(n)]
    for i in range(n):
        for j in range(n):
            if i == 0 and j == 0:
                p[0][0] = t[0][0]
            elif i == 0:
                p[0][j] = t[0][j] + p[0][j - 1]
            elif j == 0:
                p[i][0] = t[i][0] + p[i - 1][0]
            else:
                p[i][j] = t[i][j] + min(p[i - 1][j], p[i][j - 1])
    return p
```

```
def poidsMinimal(t):
    n = len(t)
    p = poids(t)
    return p[n - 1][n - 1]
```

```
def cheminDePoidsMinimal(t):
    n = len(t)
    p = poids(t)
    i, j = n - 1, n - 1
    chemin = [(i, j)]
    while (i, j) != (0, 0):
        if j == 0:
            i -= 1
        elif i == 0:
            j -= 1
        elif p[i - 1][j] < p[i][j - 1]:
            i -= 1
        else:
            j -= 1
        chemin.append((i, j))
    return chemin[::-1]
```