

Exercice 1 (4 points)

Cet exercice traite de programmation, d'algorithmique et de complexité.

On souhaite rechercher dans un tableau les k plus proches voisins d'un objet donné.

On dispose pour cela d'un tableau t non vide contenant des objets d'un même type et d'une fonction `distance` qui renvoie la distance entre deux objets quelconques de ce type.

Étant donné un objet `cible` du même type que ceux du tableau t , on cherche à déterminer les indices des k éléments du tableau t qui sont les plus proches de cet objet (c'est-à-dire ceux dont la distance à l'objet `cible` est la plus petite).

1. On suppose dans cette question que $k = 1$.

La fonction `plus_proche_voisin(t, cible)` ci-dessous prend en argument le tableau t et l'objet `cible`. Écrire sur votre copie uniquement le bloc d'instructions manquant pour que la fonction renvoie l'indice d'un plus proche voisin de `cible`.

```
def plus_proche_voisin(t, cible) :
    dmin = distance(t[0], cible)
    idx_ppv = 0
    n = len(t)
    for idx in range(1, n) :
        } bloc à écrire

    return idx_ppv
```

2. On considère que le coût en temps du bloc manquant est constant. Quelle est la complexité de la fonction `plus_proche_voisin` quand $k = 1$?

Dans la suite, on suppose que $k \geq 1$.

3. Une approche naïve consiste à parcourir le tableau t pour trouver l'indice de l'élément le plus proche de `cible`, puis à recommencer pour trouver l'indice du deuxième élément le plus proche de `cible`, et ainsi de suite. Cela implique de parcourir k fois tout le tableau.
Afin de réduire le nombre d'appels à la fonction `distance`, la stratégie suivante permet de ne parcourir le tableau t qu'une seule fois. Lors de ce parcours, on stocke dans une liste `kppv`, initialement vide, les tuples (idx, d) où idx est l'indice d'un k plus proche voisin de `cible` déjà rencontré et d la distance correspondante, triés dans l'ordre décroissant de leur distance à `cible`.
La fonction `recherche_kppv(t, k, cible)` ci-après renvoie ainsi la liste des tuples (idx, d) où idx est l'indice d'un k plus proche voisin de `cible` dans le tableau t et d la distance correspondante.

On admet que la fonction `insertion(kppv, idx, d)` insère le tuple `(idx, d)` dans la liste `kppv` de sorte que celle-ci demeure triée dans l'ordre décroissant des distances.

```
def recherche_kppv(t, k, cible):
    kppv = []
    n = len(t)
    for idx in range(n):
        obj = t[idx]
        if len(kppv) < k:
            insertion(kppv, idx, distance(obj, cible))
        else:
            i0, d0 = kppv[0]
            if distance(obj, cible) < d0:
                kppv.pop(0) # supprime le 1er élément de kppv
                insertion(kppv, idx, distance(obj, cible))
    return kppv
```

- a. On remarque qu'il y a plusieurs appels identiques à la fonction `distance(obj, cible)`. Comment ne faire qu'un seul appel de cette fonction ?
- b. Expliquer l'intérêt de maintenir la liste `kppv` triée.
- c. Écrire une fonction `insertion(kppv, idx, d)` qui insère le tuple `(idx, d)` dans la liste `kppv` préalablement triée en préservant l'ordre décroissant selon l'élément `d`.

On pourra éventuellement utiliser la méthode `insert` dont la documentation, fournie par la commande `help(list.insert)`, est la suivante :

```
insert(self, index, object, /)
    Insère l'objet object avant la position index dans l'objet
    appelant référencé par self.
```

Exemple d'utilisation de la méthode `insert` :

```
>>> liste = [4, 2, 8, 9]
>>> liste.insert(1, 3)
>>> liste
[4, 3, 2, 8, 9]
```