Algorithmes sur les listes

Informatique pour tous

Concaténation de listes

⚠ Il n'est pas possible d'ajouter plusieurs éléments avec `append`.

Concaténation de listes

⚠ Il n'est pas possible d'ajouter plusieurs éléments avec `append`.

Le $+$ permet de concaténer (mettre à la suite) deux listes ou deux chaînes de caractères :

```
In [14]: [2, 3] + [5, 7, 11]
Out[14]: [2, 3, 5, 7, 11]

In [15]: "abc" + "def"
Out[15]: 'abcdef'
```

Concaténation de listes

⚠ Il n'est pas possible d'ajouter plusieurs éléments avec `append`.

Le $+$ permet de concaténer (mettre à la suite) deux listes ou deux chaînes de caractères :

```
In [14]: [2, 3] + [5, 7, 11]
Out[14]: [2, 3, 5, 7, 11]

In [15]: "abc" + "def"
Out[15]: 'abcdef'
```

`L1 + L2` est en complexité $O($`len(L2)`$)$.

Égalité de listes

Question

Écrire une fonction déterminant si deux listes sont égales.

Égalité de listes

Question

Écrire une fonction déterminant si deux listes sont égales.

```
def egal(L1, L2):
    if len(L1) != len(L2):
        return False
    for i in range(len(L1)):
        if L1[i] != L2[i]:
            return False
    return True
```

Complexité de `egal(L1, L2)` :

Égalité de listes

Question

Écrire une fonction déterminant si deux listes sont égales.

```
def egal(L1, L2):
    if len(L1) != len(L2):
        return False
    for i in range(len(L1)):
        if L1[i] != L2[i]:
            return False
    return True
```

Complexité de `egal(L1, L2)` : $O($`len(L1)`$)$.

Égalité de listes

Question

Écrire une fonction déterminant si deux listes sont égales.

```
def egal(L1, L2):
    if len(L1) != len(L2):
        return False
    for i in range(len(L1)):
        if L1[i] != L2[i]:
            return False
    return True
```

On peut aussi directement écrire `L1 == L2` pour savoir si deux listes sont égales.

Supprimer un élément d'une liste

Si `L` est une liste, `L.pop(i)` a pour effet de supprimer et renvoyer l'élément d'indice `i` de `L`.

```
In [16]: L = [2, 3, 5, 7, 11, 13]

In [17]: L.pop(2)
Out[17]: 5

In [18]: L
Out[18]: [2, 3, 7, 11, 13]
```

Supprimer un élément d'une liste

Si `L` est une liste, `L.pop(i)` a pour effet de supprimer et renvoyer l'élément d'indice `i` de `L`.

```
In [16]: L = [2, 3, 5, 7, 11, 13]

In [17]: L.pop(2)
Out[17]: 5

In [18]: L
Out[18]: [2, 3, 7, 11, 13]
```

Attention :
`L.pop(i)` décale tous les indices après `i`, avec complexité O(`len(L)`).
Par contre `L.pop()` supprime le dernier élément en O(1).

Récapitulatif

Opérations à connaître sur une liste `L` :

Opération	**Effet**	`str`
`len(L)`	Renvoie la taille de L	✓
`L[i]`	Renvoie l'élément d'indice i	✓
`L[i:j]`	Renvoie la sous-liste des éléments d'indices i à j-1	✓
`L[i] = ...`	Modifie l'élément d'indice i	×
`L.append(e)`	Ajoute l'élément e à la fin de L	×
`L.pop(i)`	Supprime et renvoie l'élément d'indice i	×
`L1 + L2`	Renvoie la concaténation de L1 et L2	✓
`L1 == L2`	Renvoie True ssi L1 et L2 sont égales	✓

Recherche d'un élément dans une liste

Question

Écrire une fonction `contient(L, e)` qui renvoie `True` si la liste `L` contient l'élément `e`, `False` sinon.

Recherche d'un élément dans une liste

Question

Écrire une fonction `contient(L, e)` qui renvoie `True` si la liste `L` contient l'élément `e`, `False` sinon.

```
def contient(L, e):
    for i in range(len(L)):
        if L[i] == e:
            return True
    return False
```

Recherche d'un élément dans une liste

```
def contient(L, e):
    for i in range(len(L)):
        if L[i] == e:
            return True
    return False
```

La complexité dans le pire cas de `contient(L, e)` est :

Recherche d'un élément dans une liste

```
def contient(L, e):
    for i in range(len(L)):
        if L[i] == e:
            return True
    return False
```

La complexité dans le pire cas de `contient(L, e)` est : O(`len(L)`).

Recherche d'un élément dans une liste

```
def contient(L, e):
    for i in range(len(L)):
        if L[i] == e:
            return True
    return False
```

La complexité dans le pire cas de `contient(L, e)` est : $O(\texttt{len(L)})$.

Nous allons voir un algorithme plus rapide quand la liste est **triée**.

Recherche dichotomique

Question

Comment trouver efficacement un élément `e` dans une liste **triée** `L` ?

Recherche dichotomique

Question

Comment trouver efficacement un élément `e` dans une liste **triée** `L` ?

On peut comparer `e` avec le **milieu** `m` de `L` :

- Si `e == m`, on a trouvé notre élément.
- Si `e > m`, il faut chercher `e` dans la partie droite de `L`
- Si `e < m`, il faut chercher `e` dans la partie gauche de `L`

Recherche dichotomique

Exemple : on veut savoir si 14 appartient à la liste :

$$L = [-2, 1, 2, 4, 6, 7, 8, 9, 11, 12, 14, 15, 18, 22, 54]$$

Le nombre d'itérations de la fonction `contient(L, 14)` est :

Recherche dichotomique

Exemple : on veut savoir si 14 appartient à la liste :

$$L = [-2, 1, 2, 4, 6, 7, 8, 9, 11, 12, 14, 15, 18, 22, 54]$$

Le nombre d'itérations de la fonction `contient(L, 14)` est : 11.

Recherche dichotomique

Avec la recherche dichotomique :

$$[-2, 1, 2, 4, 6, 7, 8, \underline{\mathbf{9}}, 11, 12, 14, 15, 18, 22, 54]$$

$$9 < 14$$

Recherche dichotomique

Avec la recherche dichotomique :

$$[\text{-2, 1, 2, 4, 6, 7, 8, 9, } \boxed{\text{11, 12, 14, 15, 18, 22, 54}}\ ]$$

$$9 < 14$$

Recherche dichotomique

Avec la recherche dichotomique :

$$[\text{-2, 1, 2, 4, 6, 7, 8, 9, } \boxed{\text{11, 12, 14, } \underline{\mathbf{15}}\text{, 18, 22, 54}}\,]$$

$$14 < 15$$

Recherche dichotomique

Avec la recherche dichotomique :

$$[-2, 1, 2, 4, 6, 7, 8, 9, \boxed{11, \underline{\mathbf{12}}, 14}, 15, 18, 22, 54]$$

$$12 < 14$$

Recherche dichotomique

Avec la recherche dichotomique :

[-2, 1, 2, 4, 6, 7, 8, 9, 11, 12, $\boxed{\mathbf{\underline{14}}}$, 15, 18, 22, 54]

14 trouvé !

Recherche dichotomique

Avec la recherche dichotomique :

[-2, 1, 2, 4, 6, 7, 8, 9, 11, 12, $\boxed{\underline{\mathbf{14}}}$, 15, 18, 22, 54]

14 trouvé !

On a fait seulement 4 itérations.

Recherche dichotomique

```python
def contient_dichotomie(L, e):
    debut = 0 # indice de début
    fin = len(L) # indice de fin exclu
    while debut < fin:
        milieu = (debut + fin) // 2
        if L[milieu] == e:
            return True
        elif L[milieu] < e: # il faut chercher à droite
            debut = milieu + 1
        else: # il faut chercher à gauche
            fin = milieu
    return False
```

Question

Quelle est la complexité de `contient_dichotomie(L, e)` si `L` est une liste de taille n ?

Recherche dichotomique

```
def contient_dichotomie(L, e):
    debut = 0 # indice de début
    fin = len(L) # indice de fin exclu
    while debut < fin:
        milieu = (debut + fin) // 2
        if L[milieu] == e:
            return True
        elif L[milieu] < e: # il faut chercher à droite
            debut = milieu + 1
        else: # il faut chercher à gauche
            fin = milieu
    return False
```

Complexité dans le meilleur des cas :

Recherche dichotomique

```
def contient_dichotomie(L, e):
    debut = 0 # indice de début
    fin = len(L) # indice de fin exclu
    while debut < fin:
        milieu = (debut + fin) // 2
        if L[milieu] == e:
            return True
        elif L[milieu] < e: # il faut chercher à droite
            debut = milieu + 1
        else: # il faut chercher à gauche
            fin = milieu
    return False
```

Complexité dans le meilleur des cas : $O(1)$.

Recherche dichotomique

```
def contient_dichotomie(L, e):
    debut = 0 # indice de début
    fin = len(L) # indice de fin exclu
    while debut < fin:
        milieu = (debut + fin) // 2
        if L[milieu] == e:
            return True
        elif L[milieu] < e: # il faut chercher à droite
            debut = milieu + 1
        else: # il faut chercher à gauche
            fin = milieu
    return False
```

Complexité dans le pire des cas :

Recherche dichotomique

```
def contient_dichotomie(L, e):
    debut = 0 # indice de début
    fin = len(L) # indice de fin exclu
    while debut < fin:
        milieu = (debut + fin) // 2
        if L[milieu] == e:
            return True
        elif L[milieu] < e: # il faut chercher à droite
            debut = milieu + 1
        else: # il faut chercher à gauche
            fin = milieu
    return False
```

Complexité dans le pire des cas : $O(\log_2(n))$.

Démonstration au tableau.

Recherche d'un mot dans une chaîne de caractère

En bio-informatique, on a besoin de savoir si une certaine séquence de bases azotées (A, C, G, T ou U) apparaît dans un ADN, pour savoir si une mutation est présente, par exemple.

Recherche d'un mot dans une chaîne de caractère

On modélise l'ADN par une chaîne de caractères. Par exemple, si l'ADN est la chaîne "ACTTUUUAACAGGT" on veut savoir si "UAAC" y appartient.

Question

Écrire une fonction `sous_mot(mot, chaine)` qui renvoie `True` ssi `mot` est une sous-chaîne contiguë de `chaine`.

Recherche d'un mot dans une chaîne de caractère

Idée : parcourir une à une les lettres de la chaîne et regarder si le mot commence par cette lettre.
Pour chercher "UAAC" dans "ACTTUUUAACAGGT" :

"ACTTUUUAACAGGT"

Recherche d'un mot dans une chaîne de caractère

Idée : parcourir une à une les lettres de la chaîne et regarder si le mot commence par cette lettre.
Pour chercher "UAAC" dans "ACTTUUUAACAGGT" :

"ACTTUUUAACAGGT"

Recherche d'un mot dans une chaîne de caractère

Idée : parcourir une à une les lettres de la chaîne et regarder si le mot commence par cette lettre.
Pour chercher "UAAC" dans "ACTTUUUAACAGGT" :

"ACTTUUUAACAGGT"

Recherche d'un mot dans une chaîne de caractère

Idée : parcourir une à une les lettres de la chaîne et regarder si le mot commence par cette lettre.
Pour chercher "UAAC" dans "ACTTUUUAACAGGT" :

"ACTTUUUAACAGGT"

Recherche d'un mot dans une chaîne de caractère

Idée : parcourir une à une les lettres de la chaîne et regarder si le mot commence par cette lettre.
Pour chercher "UAAC" dans "ACTTUUUAACAGGT" :

"ACTTUUUAACAGGT"

Recherche d'un mot dans une chaîne de caractère

Idée : parcourir une à une les lettres de la chaîne et regarder si le mot commence par cette lettre.
Pour chercher "UAAC" dans "ACTTUUUAACAGGT" :

"ACTTUUUAACAGGT"

Recherche d'un mot dans une chaîne de caractère

Idée : parcourir une à une les lettres de la chaîne et regarder si le mot commence par cette lettre.
Pour chercher "UAAC" dans "ACTTUUUAACAGGT" :

"ACTTUUUAACAGGT"

Recherche d'un mot dans une chaîne de caractère

Idée : parcourir une à une les lettres de la chaîne et regarder si le mot commence par cette lettre.
Pour chercher "UAAC" dans "ACTTUUUAACAGGT" :

"ACTTUU UAAC AGGT"

Recherche d'un mot dans une chaîne de caractère

Fonction à compléter :

```
def sous_mot(mot, chaine):
    for i in range(                          ):
        if chaine[                  ] == mot:
            return True
    return False
```

Recherche d'un mot dans une chaîne de caractère

```
def sous_mot(mot, chaine):
    for i in range(len(chaine) - len(mot) + 1):
        if chaine[i : i + len(mot)] == mot:
            return True
    return False
```

Recherche d'un mot dans une chaîne de caractère

Question

Modifier `sous_mot` de façon à ce qu'elle renvoie le nombre de fois que `mot` apparaît dans `chaine`.

```
def sous_mot(mot, chaine):
    for i in range(len(chaine) - len(mot) + 1):
        if chaine[i : i + len(mot)] == mot:
            return True
    return False
```

Recherche d'un mot dans une chaîne de caractère

```
def nb_sous_mot(mot, chaine):
    res = 0
    for i in range(len(chaine) - len(mot) + 1):
        if chaine[i : i + len(mot)] == mot:
            res = res + 1
    return res
```

Recherche d'un mot dans une chaîne de caractère

```
def nb_sous_mot(mot, chaine):
    res = 0
    for i in range(len(chaine) - len(mot) + 1):
        if chaine[i : i + len(mot)] == mot:
            res = res + 1
    return res
```

```
In [12]: nb_sous_mot("AG", "AAGTGAGCCGAAGT")
Out[12]: 3
```

Recherche d'un mot dans une chaîne de caractère

Remarque : les fonctions `sous_mot` et `nb_sous_mot` fonctionnent aussi avec des listes.

```
In [13]: nb_sous_mot([2, 5], [2, 3, 5, False, 2, 5, "ab"])
Out[13]: 1
```