

Stage d'informatique MP2I

Je m'appelle Quentin Fortier :

- ENS Lyon en informatique (2009-2013)
- Thèse en théorie des graphes (2013-2016)
- Professeur d'informatique en CPGE (2016-2020)
- Ingénieur en optimisation et data science (2020-2021)
- Professeur d'informatique en CPGE (2021-...)

Organisation du cours

Pour le cours (après le stage) nous allons utiliser Jupyter.

Organisation du cours

Pour le cours (après le stage) nous allons utiliser Jupyter.

- Site du cours : <https://github.com/mp2i-fsm/mp2i-2021>
- Amenez si possible vos PC portables en cours/TP/TD d'informatique pour tester les exemples/exercices
- Sinon, vous le ferez sur papier

Objectif du stage

Objectif : voir l'algorithmique de 1ère NSI (en Python).

- Recherche linéaire dans une liste

Objectif du stage

Objectif : voir l'algorithmique de 1ère NSI (en Python).

- Recherche linéaire dans une liste
- Recherche par dichotomie

Objectif du stage

Objectif : voir l'algorithmique de 1ère NSI (en Python).

- Recherche linéaire dans une liste
- Recherche par dichotomie
- k plus proches voisins

Objectif du stage

Objectif : voir l'algorithmique de 1ère NSI (en Python).

- Recherche linéaire dans une liste
- Recherche par dichotomie
- k plus proches voisins
- Algorithmes gloutons

Objectif du stage

Objectif : voir l'algorithmique de 1ère NSI (en Python).

- Recherche linéaire dans une liste
- Recherche par dichotomie
- k plus proches voisins
- Algorithmes gloutons
- Algorithmes de tri

Parcours séquentiel d'un tableau

Exercice

Écrire une fonction telle que :

- **Entrée** : une liste L et un élément e
- **Sortie** : True si e appartient à L, False sinon

Parcours séquentiel d'un tableau

Exercice

Écrire une fonction telle que :

- **Entrée** : une liste L et un élément e
- **Sortie** : True si e appartient à L, False sinon

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

Recherche du maximum

Exercice

Écrire une fonction permettant d'obtenir l'indice du maximum d'une liste.

Recherche du maximum

Exercice

Écrire une fonction permettant d'obtenir l'indice du maximum d'une liste.

```
def maximum(L):  
    i_max = 0  
    for i in range(len(L)):  
        if L[i] > L[i_max]:  
            i_max = i  
    return i_max
```

Calcul de la somme

Exercice

Écrire une fonction permettant d'obtenir la somme des éléments d'une liste.

Exercice

Écrire une fonction permettant d'obtenir la somme des éléments d'une liste.

```
def somme(L):  
    s = 0  
    for i in range(len(L)):  
        s += L[i]  
    return s
```

Soit L une liste **triée**.

Pour savoir si L contient un élément e , on a vu qu'on peut parcourir tous les éléments un par un.

Question

Pouvez-vous trouver une méthode plus efficace ?

Question

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

Question

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

On peut comparer e avec le **milieu** $L[m]$ de L :

- Si $e == L[m]$,

Question

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

On peut comparer e avec le **milieu** $L[m]$ de L :

- Si $e == L[m]$, on a trouvé notre élément.
- Si $e > L[m]$,

Question

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

On peut comparer e avec le **milieu** $L[m]$ de L :

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

Question

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

On peut comparer e avec le **milieu** $L[m]$ de L :

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

Recherche dichotomique

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

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

Nombre d'itérations en regardant les éléments un par un :

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

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

Nombre d'itérations en regardant les éléments un par un : 11.

Recherche dichotomique

Avec la recherche dichotomique :

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

$$9 < 14$$

Recherche dichotomique

Avec la recherche dichotomique :

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

$9 < 14$

Recherche dichotomique

Avec la recherche dichotomique :

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

14 < 15

Recherche dichotomique

Avec la recherche dichotomique :

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

$12 < 14$

Recherche dichotomique

Avec la recherche dichotomique :

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

14 trouvé!

Recherche dichotomique

Avec la recherche dichotomique :

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

14 trouvé!

On a fait seulement 4 itérations.

Recherche par dichotomie

```
def appartient_dichotomie(e, L):  
    i = 0  
    j = len(L) - 1  
    while i <= j:  
        m = (i + j) // 2  # milieu  
        if L[m] == e:  
            return True  # on a trouvé e  
        elif L[m] < e:  
            i = m + 1  # chercher à droite de m  
        else :  
            j = m - 1  # chercher à gauche de m  
    return False # aucun élément ne peut être égal à e
```

k plus proches voisins

L'algorithme KNN (k plus proches voisins) est un exemple d'algorithme de classification supervisée en apprentissage automatique (machine learning).

k plus proches voisins

L'algorithme KNN (k plus proches voisins) est un exemple d'algorithme de classification supervisée en apprentissage automatique (machine learning).

Il sert à séparer des données (qui peuvent être des points dans l'espace par exemple) en k groupes, chaque groupe étant similaire.

k plus proches voisins

L'algorithme KNN (k plus proches voisins) est un exemple d'algorithme de classification supervisée en apprentissage automatique (machine learning).

Il sert à séparer des données (qui peuvent être des points dans l'espace par exemple) en k groupes, chaque groupe étant similaire.

Pour utiliser KNN on a besoin :

- de données initiales pour lesquelles on connaît les groupes
- d'une notion de distance entre deux données

k plus proches voisins

Il fonctionne en deux temps :

- 1 **Entraînement** : on utilise des données dont on connaît déjà la classe.

k plus proches voisins

Il fonctionne en deux temps :

- ❶ **Entraînement** : on utilise des données dont on connaît déjà la classe.
- ❷ **Prédiction** : étant donnée une nouvelle donnée, on lui associe la classe majoritaire parmi ses k plus proches voisins.

Démo avec p5.js

Démo avec Jupyter

Algorithmes gloutons

La stratégie d'un **algorithme glouton** est d'effectuer à chaque étape l'action qui semble la plus intéressante localement (sans regarder ce qui va se passer plus tard).

Algorithmes gloutons

La stratégie d'un **algorithme glouton** est d'effectuer à chaque étape l'action qui semble la plus intéressante localement (sans regarder ce qui va se passer plus tard).

Problème du sac à dos

On considère un sac à dos de capacité 10kg et les objets suivants :

poids (kg)	2	2	2	3	5	5	8
valeur (€)	1	1	1	7	10	10	13

Quelle est la valeur maximum que l'on peut mettre dans le sac ?

Problème du sac à dos

On considère un sac à dos de capacité 10kg et les objets suivants :

poids (kg)	2	2	2	3	5	5	8
valeur (€)	1	1	1	7	10	10	13

Quelle est la valeur maximum que l'on peut mettre dans le sac ?

Algorithme glouton n°1 : ajouter les éléments dans l'ordre croissant de poids, tant que c'est possible

Problème du sac à dos

On considère un sac à dos de capacité 10kg et les objets suivants :

poids (kg)	2	2	2	3	5	5	8
valeur (€)	1	1	1	7	10	10	13

Quelle est la valeur maximum que l'on peut mettre dans le sac ?

Algorithme glouton n°2 : ajouter les éléments dans l'ordre décroissant de valeur, tant que c'est possible

Problème du sac à dos

On considère un sac à dos de capacité 10kg et les objets suivants :

poids (kg)	2	2	2	3	5	5	8
valeur (€)	1	1	1	7	10	10	13
valeur/poids	0.5	0.5	0.5	2.3	2	2	1.6

Quelle est la valeur maximum que l'on peut mettre dans le sac ?

Algorithme glouton n°3 : ajouter les éléments dans l'ordre décroissant de valeur/poids, tant que c'est possible

- ① Un algorithme glouton peut donner la solution optimale : arbre couvrant de poids minimum, problème fractionnaire du sac à dos...

Algorithmes gloutons

- ① Un algorithme glouton peut donner la solution optimale : arbre couvrant de poids minimum, problème fractionnaire du sac à dos...
- ② Même si l'algorithme glouton n'est pas optimal, il peut être intéressant pour donner une approximation de l'optimum.

Algorithmes de tris

Deux tris en 1ère NSI :

- Tri par insertion

Deux tris en 1ère NSI :

- Tri par insertion
- Tri par sélection

Tri par sélection

- 1 Chercher le minimum

Tri par sélection

- 1 Chercher le minimum
- 2 Le mettre à l'indice 0

Tri par sélection

- 1 Chercher le minimum
- 2 Le mettre à l'indice 0
- 3 Chercher le 2ème élément le plus petit

Tri par sélection

- 1 Chercher le minimum
- 2 Le mettre à l'indice 0
- 3 Chercher le 2ème élément le plus petit
- 4 Le mettre à l'indice 1
- 5 ...

Tri par sélection

```
def tri_selection(L):  
    for i in range(len(L)): # cherche le ième minimum  
        i_mini = i  
        for k in range(i, len(L)):  
            if L[k] < L[i_mini]:  
                i_mini = k  
        L[i], L[i_mini] = L[i_mini], L[i] # échange
```