Programmation Fonctionnelle

Séance 3 : Algorithmes de recherche et tri

Alexandros Singh

Université Paris 8

15 octobre 2023

Problème : Tri d'une liste de nombres

Etant donné une liste d'entiers naturels, créez une nouvelle liste avec les mêmes éléments mais ordonnés selon un ordre donné. éléments ordonnés selon un ordre donné.

Il existe de nombreux algorithmes qui permettent de résoudre ce problème. Aujourd'hui, nous allons en étudier deux : le tri par sélection et le tri à bulles.

Tri par sélection

Le tri par sélection prend en entrée une liste `l` et la manipule pour créer une liste triée comme suit :

- Si la liste est vide, le résultat est la liste triée (initialisé à une liste vide) que nous avons créée jusqu'à présent.
- Sinon, nous recherchons le plus petit (ou le plus grand, selon l'ordre final souhaité) élément de `l` et nous le plaçons dans la liste triée.
  Nous reprenons ensuite la même procédure avec la liste `l` moins l'élément `e`.

Le tri par sélection prend en entrée une liste `l` et la manipule pour créer une liste triée comme suit :

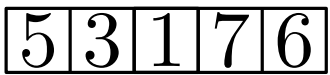
- Si la liste est vide, le résultat est la liste triée (initialisé à une liste vide) que nous avons créée jusqu'à présent.
- Sinon, nous recherchons le plus petit (ou le plus grand, selon l'ordre final souhaité) élément de `l` et nous le plaçons dans la liste triée.
  Nous reprenons ensuite la même procédure avec la liste `l` moins l'élément `e`.

Le tri par sélection prend en entrée une liste `l` et la manipule pour créer une liste triée comme suit :

- Cas de base : Si la liste est vide, le résultat est la liste triée (initialisé à une liste vide) que nous avons créée jusqu'à présent.
- Appel récursif : Sinon, nous recherchons le plus petit (ou le plus grand, selon l'ordre final souhaité) élément de `l` et nous le plaçons dans la liste triée.
  Nous reprenons ensuite la même procédure avec la liste `l` moins l'élément `e`.

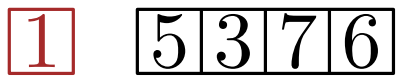
Cet algorithme peut être implémenté de façon récursive !

Tri d'une liste par ordre croissant à l'aide du tri par sélection :



Tri d'une liste par ordre croissant à l'aide du tri par sélection :

Tri d'une liste par ordre croissant à l'aide du tri par sélection :



Tri d'une liste par ordre croissant à l'aide du tri par sélection :

Tri d'une liste par ordre croissant à l'aide du tri par sélection :

Tri par sélection - exemple

Tri d'une liste par ordre croissant à l'aide du tri par sélection :

Tri par sélection - exemple

Tri d'une liste par ordre croissant à l'aide du tri par sélection :

Tri d'une liste par ordre croissant à l'aide du tri par sélection :

Tri par sélection - exemple

Tri d'une liste par ordre croissant à l'aide du tri par sélection :

Tri d'une liste par ordre croissant à l'aide du tri par sélection :

Tri d'une liste par ordre croissant à l'aide du tri par sélection :

Pour implémenter le tri par sélection, nous avons besoin d'une fonction qui trouve l'élément minimal (ou maximal) d'une liste donnée. Réutilisons le morceau de code suivant que nous avons présenté lors de la séance #1 :

```
(define find-min (lambda (l)
 (if (null? (cdr l))
   (car l)
   (min (car l) (find-min (cdr l))))))
```

Combien de comparaisons (en utilisant `min`) cette fonction fait-elle en fonction de la taille n de `l` ?

Pour implémenter le tri par sélection, nous avons besoin d'une fonction qui trouve l'élément minimal (ou maximal) d'une liste donnée. Réutilisons le morceau de code suivant que nous avons présenté lors de la séance #1 :

```
(define find-min (lambda (l)
 (if (null? (cdr l))
   (car l)
   (min (car l) (find-min (cdr l))))))
```

Combien de comparaisons (en utilisant `min`) cette fonction fait-elle en fonction de la taille n de `l` ? Réponse : n.

Et le tri par sélection ? Combien de comparaisons entre des entiers naturels effectue-t-il ?

Et le tri par sélection ? Combien de comparaisons entre des entiers naturels effectue-t-il ?
Pour une liste `l` de longueur n, il faudra appliquer `find-min` n fois.

Et le tri par sélection ? Combien de comparaisons entre des entiers naturels effectue-t-il ?
Pour une liste `l` de longueur n, il faudra appliquer `find-min` n fois.

$$\sum_{1}^{n} \text{ nombre de comparaisons effectuées par } \texttt{find-min}.$$

Et le tri par sélection ? Combien de comparaisons entre des entiers naturels effectue-t-il ?
Pour une liste `l` de longueur n, il faudra appliquer `find-min` n fois.

$$\sum_{1}^{n} \text{ nombre de comparaisons effectuées par } \texttt{find-min}.$$

$$= \sum_{1}^{n} n = n^2.$$

Et le tri par sélection ? Combien de comparaisons entre des entiers naturels effectue-t-il ? Pour une liste `l` de longueur n, il faudra appliquer `find-min` n fois.

$$\sum_{1}^{n} \text{ nombre de comparaisons effectuées par } \texttt{find-min}.$$

$$= \sum_{1}^{n} n = n^2.$$

Enfin, il y a un coût associé à la construction de la liste triée, qui dépend de l'implémentation spécifique que nous utilisons. En gros, nous aurons besoin de n opérations (comme `cons`) pour cela. Mais,

$$\lim_{n \to \infty} \frac{n^2 + n}{n^2} = 1$$

Donc, pour des grandes listes, la majorite du temps est consacrée à des comparaisons de nombres naturels.

Le tri à bulles fonctionne en appliquant l'itération suivante (en supposant que la liste d'entrée n'est pas vide) :

- Pour chaque paire x, y d'éléments contigus dans la liste, vérifier si x est plus petit (ou plus grand, selon l'ordre final souhaité) que y : si c'est le cas, gardez-les tels quels, sinon échangez-les.

Si nous continuons à appliquer cette itération, nous finirons par obtenir une liste triée.

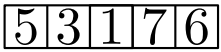
Le tri à bulles fonctionne en appliquant l'itération suivante (en supposant que la liste d'entrée n'est pas vide) :

- Pour chaque paire x, y d'éléments contigus dans la liste, vérifier si x est plus petit (ou plus grand, selon l'ordre final souhaité) que y : si c'est le cas, gardez-les tels quels, sinon échangez-les.

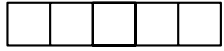
Si nous continuons à appliquer cette itération, nous finirons par obtenir une liste triée. Cet algorithme peut être implémenté de manière récursive !

Tri à bulles - exemple

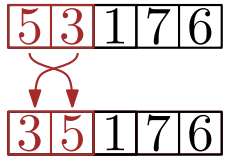
Tri d'une liste par ordre croissant à l'aide du tri à bulles :



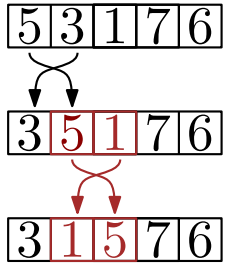
Tri d'une liste par ordre croissant à l'aide du tri à bulles :



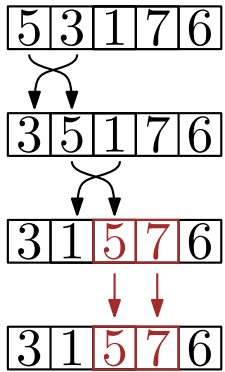
Tri d'une liste par ordre croissant à l'aide du tri à bulles :



Tri d'une liste par ordre croissant à l'aide du tri à bulles :

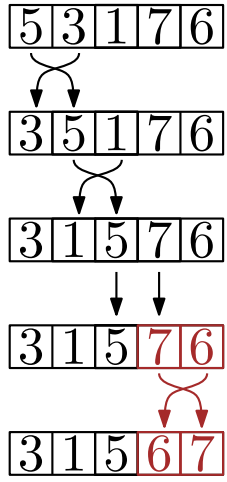


Tri d'une liste par ordre croissant à l'aide du tri à bulles :

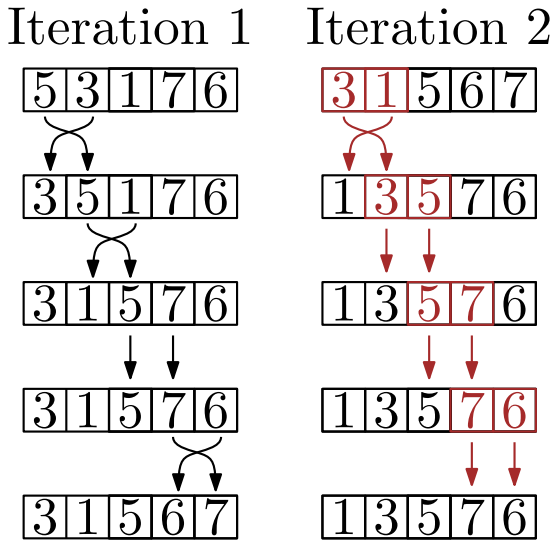


Tri à bulles - exemple

Tri d'une liste par ordre croissant à l'aide du tri à bulles :



Tri d'une liste par ordre croissant à l'aide du tri à bulles :



Combien d'itérations faut-il pour trier une liste `l` de n éléments ?

Combien d'itérations faut-il pour trier une liste `l` de n éléments ?
La réponse dépend de la liste `l` :

- Pour une liste déjà ordonnée, nous n'aurions besoin d'aucune itération.

Combien d'itérations faut-il pour trier une liste `l` de n éléments ?
La réponse dépend de la liste `l` :

- Pour une liste déjà ordonnée, nous n'aurions besoin d'aucune itération.
- Pour une liste dont seuls les deux premiers éléments sont en désordre, une seule itération suffit.

Combien d'itérations faut-il pour trier une liste `l` de n éléments ?
La réponse dépend de la liste `l` :

- Pour une liste déjà ordonnée, nous n'aurions besoin d'aucune itération.
- Pour une liste dont seuls les deux premiers éléments sont en désordre, une seule itération suffit.
- Pour une liste ordonnée dans l'ordre inverse de ce que nous voulons, nous aurions besoin de $\approx n^2$ iterations, c'est le pire des cas !

Problème : recherche (sous forme de problème de décision)

Etant donné une liste `l` et une valeur `e`, décider si `e` apparaît dans `l`.

Rappelez-vous le devoir de la semaine dernière : nous avons implémenté un algorithme qui résout exactement ce problème !

```
(define linear-search
  (lambda (l e)
    (foldr (lambda (x y) (or (eq? x e) y)) #f l)))
```

Combien de comparaisons (via eq?) l'algorithme aura-t-il besoin en fonction de la taille n de la liste `l` ?

Problème : recherche (sous forme de problème de décision)

Etant donné une liste `l` et une valeur `e`, décider si `e` apparaît dans `l`.

Rappelez-vous le devoir de la semaine dernière : nous avons implémenté un algorithme qui résout exactement ce problème !

```
(define linear-search
  (lambda (l e)
    (foldr (lambda (x y) (or (eq? x e) y)) #f l)))
```

Combien de comparaisons (via eq?) l'algorithme aura-t-il besoin en fonction de la taille n de la liste `l` ? Réponse : n.

Pouvez-vous imaginer un meilleur algorithme ?

Problème : Devinez le nombre

Je pense à un nombre entre 0 et 100. Vous pouvez me poser des questions du type "est-ce x ?" et je réponds par l'une des trois réponses suivantes et je réponds par l'une ou l'autre des trois réponses appropriées :

- Non, il est plus petit que x.
- Non, il est plus grand que x.
- Oui, c'est x !

Problème : Devinez le nombre

Je pense à un nombre entre 0 et 100. Vous pouvez me poser des questions du type "est-ce x ?" et je réponds par l'une des trois réponses suivantes et je réponds par l'une ou l'autre des trois réponses appropriées :

- Non, il est plus petit que x.
- Non, il est plus grand que x.
- Oui, c'est x !

Il s'agit d'un cas particulier du problème de la recherche.

- L'algorithme de recherche linéaire correspond à la demande successivement "est-ce 0 ?", "est-ce 1 ?", "est-ce 2 ?"
- Pouvez-vous imaginer un meilleur algorithme ? Utilisez le fait que la liste `l` est composée de tous les nombres de 0 à 100 et est donc ordonnée.

Etant donné une liste ordonnée `l` et une valeur `e`, faites ce qui suit :

- Comparez la valeur `e` à l'élément qui se trouve au milieu de la liste. [1]
- Si elle est égale, nous décidons que la valeur apparaît dans la liste.
  Si elle est plus petite, nous divisons la liste en deux et continuons à chercher de la même manière dans la première moitié de la liste.
  Si elle est plus grande, continuons à chercher de la même manière dans la seconde moitié.

1. Ceci n'est bien défini que si la liste est de taille impaire. Si sa taille est paire, on peut, par exemple, vérifier la valeur par rapport aux deux éléments qui se trouvent à gauche et à droite du "milieu" de la liste.

Etant donné une liste ordonnée `l` et une valeur `e`, faites ce qui suit :

- Comparez la valeur `e` à l'élément qui se trouve au milieu de la liste. [1]
- Cas de base : Si elle est égale, nous décidons que la valeur apparaît dans la liste.
  Appel récursif :
  Si elle est plus petite, nous divisons la liste en deux et continuons à chercher de la même manière dans la première moitié de la liste.
  Si elle est plus grande, continuons à chercher de la même manière dans la seconde moitié.

1. Ceci n'est bien défini que si la liste est de taille impaire. Si sa taille est paire, on peut, par exemple, vérifier la valeur par rapport aux deux éléments qui se trouvent à gauche et à droite du "milieu" de la liste.

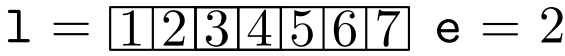
Etant donné une liste ordonnée `l` et une valeur `e`, faites ce qui suit :

- Comparez la valeur `e` à l'élément qui se trouve au milieu de la liste. [1]
- Cas de base : Si elle est égale, nous décidons que la valeur apparaît dans la liste.
  Appel récursif :
  Si elle est plus petite, nous divisons la liste en deux et continuons à chercher de la même manière dans la première moitié de la liste.
  Si elle est plus grande, continuons à chercher de la même manière dans la seconde moitié.

Le nombre de comparaisons est, dans le pire des cas, $\approx log(n)$, où n est la taille de la liste `l`.

1. Ceci n'est bien défini que si la liste est de taille impaire. Si sa taille est paire, on peut, par exemple, vérifier la valeur par rapport aux deux éléments qui se trouvent à gauche et à droite du "milieu" de la liste.

Recherche dichotomique - exemple



$$\texttt{l} = \boxed{1}\boxed{2}\boxed{3}\boxed{4}\boxed{5}\boxed{6}\boxed{7}\ \texttt{e} = 2$$

$$3 \geq 4?$$

$\texttt{l} = \boxed{1}\boxed{2}\boxed{3}\boxed{4}\boxed{5}\boxed{6}\boxed{7}$ $\texttt{e} = 3$

$\boxed{1}\boxed{2}\boxed{3}$ $\boxed{5}\boxed{6}\boxed{7}$

$3 \geq 2?$

$$\texttt{l} = \boxed{1}\boxed{2}\boxed{3}\boxed{4}\boxed{5}\boxed{6}\boxed{7} \quad \texttt{e} = 3$$

$$\boxed{1}\boxed{2}\boxed{3} \qquad \boxed{5}\boxed{6}\boxed{7}$$

$$\boxed{1} \qquad \boxed{3}$$

$$3 \geq 3?$$