Algorithmique

Vous avez dit trier ?

Support de cours du professeur

DIU, Bloc 2 : Projet

Activité 3, question 2

Groupe :
Delrot Sylvain
Hounkpatin Bonaventure
Jaubert Philippe

1 Préambule

Comment ranger des données afin de faciliter leur accès ?
C'est par exemple l'ordre alphabétique du dictionnaire, où les mots sont rangés dans un ordre logique qui permet de ne pas devoir parcourir tout l'ouvrage pour retrouver une définition.

Ce peut être aussi l'ordre intuitif dans lequel un joueur de cartes va ranger son jeu afin de limiter le temps de recherche pendant le déroulement de la partie.

Le tri peut avoir plusieurs sens : séparer, ordonner, choisir

Le tri permet essentiellement d'accélérer les recherches, grâce à l'algorithme de recherche dichotomique.

2 Objectif de la séquence

Faire découvrir aux élèves 2 types de tris, par insertion et par sélection.

Capacités attendues : Écrire un algorithme de tri. Décrire un invariant de boucle qui prouve la correction des tris par insertion, par sélection.

La terminaison de ces algorithmes est à justifier. On montre que leur coût est quadratique dans le pire cas.

La séquence se décompose en 4 étapes :

1. La découverte des algorithmes de tri, en mode débranché.
2. Écriture du pseudo code, codage et le test de ces algorithmes en python.
3. L'étude théorique des invariants de boucle, des terminaisons et des complexités.
4. Un mini-projet mettant en œuvre ces algorithmes, et permettant d'évaluer leurs propriétés.

3 Comment trier ? tris par insertion et sélection

Ici trois scénarios dont 2 en mode débranché sont proposés, à choisir par le professeur.

1 Scénario 1 : trier des cartes

Procédure :
Répartissez-vous par groupes de 3 au maximum. Vous avez pour chaque équipe un tas de 13 cartes de la même couleur.

1) On dévoile les cartes une par une et vous allez les trier au fur et à mesure.
Décrivez, par écrit, très précisément, pas à pas, comment se passe le processus de tri. Décomposez bien les étapes du raisonnement

2) Vous avez toutes les cartes en main d'un seul coup, comment les triez vous ? Comme dans l'étape précédente vous détaillez bien les différentes étapes.

Mise en commun, puis synthèse.

La première méthode s'apparente au tri par insertion
La seconde au tri par sélection.

Explication collective et démonstration de ces 2 tris à l'aide de gif animés (voir scénario 3).

2 Scénario 2 : trier des élèves

Inspiré des vidéos vues sur le web, on munit plusieurs élèves d'une feuille avec un nombre de 1 à 10 inscrit dessus.

Un élève ou le professeur fait se déplacer les élèves en fonction du mode de tri choisi.

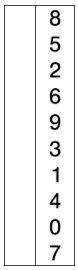
Renforcement possible de l'explication de ces 2 tris à l'aide de gif animés (voir scénario 3).

3 Scénario 3 : des animations sur écran

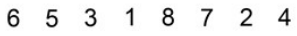
On montre des animations sur les ordinateurs et on demander à chaque groupe de 3 élèves la description de ce qu'ils viennent de visionner.
Le professeur peut apporter des précisions par des schémas au tableau.

Gif animés :

selection-sort.gif



insertion-sort.gif



4 Écriture et test des algorithmes

1 Écriture de l'algorithme en pseudo code

Activité : Ecrire individuellement un algorithme de tri par insertion et un algorithme de tri par sélection pour une liste T de n entiers naturels.
Relever.

Donner un corrigé type.

Tri par insertion :

Principe : Les i premiers éléments sont triés en ordre croissant, l'élément i+1 est inséré dans l'ordre parmi les i premiers.

A tout moment, le tableau à trier sera en 2 parties :

- Une partie triée [1 .. TailleCourante]
- Une partie non triée [TailleCourante+1 .. TailleMax]

TailleCourante en abrégé : TC dans le pseudo-code

Tri par insertion : Pseudo-code	**Commentaires**
TriInsertion(T : tableau d'entiers, TailleMax : entier) ✔ Variables Locales TC , i, p, temp : entiers **Début** **pour** TC de 1 à TailleMax - 1 faire temp ← T [TC + 1] p ← 1	
tant que T [p] < temp faire p ← p+1 **fin tant que**	Chercher la position p
pour i de TC en décroissant à p faire T[i+1] ← T[i]	Décaler les éléments

fin pour	
T[p] ← temp **fin pour** **Fin**	

Tri par sélection :

Principe : les i premiers éléments sont définitivement triés en ordre croissant et on cherche parmi les n-i restants le plus petit à mettre à l'indice i

A tout moment, le tableau à trier sera en 2 parties :

- Une partie triée [1 .. TailleCourante]
- Une partie non triée [TailleCourante+1 .. TailleMax]

TailleCourante en abrégé : TC dans le pseudo-code

- A chaque étape,
  - Choisir le plus petit élément de la partie non triée
  - Mettre cet élément à la fin de la partie triée

Tri par sélection : Pseudo-code	Commentaires
TriParSelection(T : Tableau d'entiers, TailleMax : entier) **Entrées :** T(tableau d'entiers), TailleMax (taille du tableau) **Sortie :** Tableau trié T **Début** TC, p, min, temp : entier **Pour** TC de 1 jusqu'à TailleMax -1 faire min ← T[TC] p ← TC **Pour** i de TC+1 jusqu'à TailleMax faire **Si** T[i] < min alors min ← T[i] p ← i **Fin Si** **Fin Pour** temp ← T[p] T[p] ← T[TC] T[TC] ← temp **Fin Pour** **Fin**	

2 Codage et tests

Programmer en python en solo ou en binôme les deux tris.
Les tester sur des courtes listes calibrées à la main :

- liste vide
- liste avec mini à droite
- liste avec maxi à gauche
- liste ordonnée
- liste avec des doublons.

Programme python : Tri par insertion

```
def tri_insertion(T):
    for TC in range(1,len(T)): # balayage à partir du 2ème élément
        temp = T[TC] # temp représente la valeur à insérer
        p = TC # p représente l'indice où insérer temp
        while p >= 1 and temp < T[p-1] : #insertion dans partie triée
            T[p] = T[p-1] # décale valeurs d'un rang vers la droite
            p -= 1
            T[p] = temp #Place trouvée : on insère temp à l'indice p
    return T
```

Remarque *Comme ce tri se fait en place, le return T de la fin est « facultatif » mais commode. Il permet par exemple de tester directement notre fonction dans la console.*

Programme python : Tri par sélection

```
"""
Remarque : Nous avons d'abord besoin d'une fonction qui calcule iMin l'indice
du minimum de la partie du tableau comprise entre les indices j et n-1.
Pour la lisibilité, rédigeons la séparément
"""
def indice_min(T,j):
    """retourne l'indice du minimum du sous tableau T[j:]"""
    mini = j
    for i in range(j+1,len(T)):
        if T[i] < T[mini]:
            mini = i
    return mini

# Nous pouvons maintenant implémenter facilement le tri par sélection
def tri_selection(T):
    for k in range(len(T)-1):  # le dernier élément sera forcement trié
        iMin = indice_min(T,k)
        if iMin != k:  #A quoi ça sert ?
            T[iMin],T[k] = T[k],T[iMin]
    return T   #Par commodité (car T modifié)
```

5 Correction, terminaison et coût des algorithmes.

Idéalement, lors de la conception d'un algorithme et d'un programme on veut montrer que *dans les situations connues* pour lesquelles le programme est conçu celui-ci se terminera avec succès :

- le code s'achève t-il bien (pas de programme qui « mouline », qui « plante » comme on le rencontre parfois) ?
- le résultat produit est-il fiable et conforme à ce qu'on attend ?

Le raisonnement adopté ressemble un peu aux démonstrations par récurrence vues en cours de mathématique de première S : si on peut démontrer qu'une propriété est vraie pour un terme d'une suite implique qu'elle est *aussi vraie* pour le terme suivant, alors trouver un seul terme pour lequel la propriété est vraie permet de le prouver pour tous les suivants.

Le but de cette partie est d'appliquer ces notions sur les 2 types de tri.

1 Invariant de boucle

Un invariant est une propriété qui est vraie **avant et après** chaque itération d'une boucle.

Nous devons montrer trois choses, concernant un invariant de boucle (Rappel de cours aux élèves):
Initialisation :
Il est vrai avant la première itération de la boucle.
Conservation :
S'il est vrai avant une itération de la boucle, il le reste avant l'itération suivante.
Terminaison :
Une fois terminée la boucle, l'invariant fournit une propriété utile qui aide à montrer la validité de l'algorithme

- Si les deux premières propriétés sont vérifiées, alors l'invariant est vrai avant chaque itération de la boucle.
- **La troisième propriété est peut-être la plus importante : elle est utilisée pour prouver la validité de l'algorithme**

Invariant de boucle du tri par insertion

Au début de chaque itération de la boucle **pour,**
le sous-tableau [1 .. TailleCourante] se compose des éléments qui occupaient initialement les positions T[1 . . TC] mais qui sont maintenant triés.

Commençons par montrer que:

l**'invariant est vérifié avant la première itération de la boucle**, quand TC = 1.
Le sous-tableau T[1 .. TC] se compose donc
uniquement de l'élément T[1]
Autrement dit l'élément originel de T[1].
En outre, ce sous-tableau est trié (c'est une trivialité), ce qui montre bien que l'invariant est vérifié

avant la première itération de la boucle.

Passons ensuite à la **deuxième propriété**
Il faut montrer que chaque itération conserve l'invariant
De manière informelle:
le corps de la **boucle Pour** extérieure fonctionne:

- en déplaçant T[TC], T[TC − 1], T[TC − 2], etc. d'une position vers la droite
- jusqu'à ce qu'on trouve la bonne position pour T[TC], auquel cas on insère la valeur de T[TC].

Examinons se qui se passe à la **terminaison de la boucle** :
la **boucle Pour** extérieure prend fin quand TC dépasse n-1, c'est-à-dire quand TC = n.
En substituant n à TC dans la formulation de l'invariant de boucle,
l'on a que le sous-tableau T[1..n] se compose des éléments qui appartenaient originellement à T[1..n] mais qui ont été triés depuis lors.
Or, le sous-tableau T[1 . . n] n'est autre que le tableau complet !
Par conséquent, le tableau tout entier est trié. **Donc l'algorithme est correct.**

Invariant de boucle du tri par sélection

Le tableau est une permutation du tableau initial et le sous tableau [1..taillecourante] est trié et tous ses éléments sont inférieurs ou égaux aux éléments de l'autre partie du tableau.

Autre manière de l'énoncer *: le tableau est une permutation du tableau initial et les i premiers éléments du tableau coïncident avec les i premiers éléments du tableau trié (wikipedia).*

Avant la boucle :
Cette partie est vide. Donc la propriété est trivialement vérifiée.

Dans la boucle :
on suppose que la propriété est vraie à l'indice k.

À k+1 on cherche le minimum de la partie non triée du tableau.
On l'échange avec celui placé à l'indice k+1, c'est à dire à la suite du tableau trié.
Cet élément est plus grand que ceux du tableau [0...k] donc le tableau [0...k+1] demeure trié, et tous ses éléments sont inférieurs ou égaux au reste du tableau.

A la terminaison de la boucle :
Il reste un élément supérieur ou égal à tous les éléments de la partie triée. En le plaçant à la suite, on conserve l'ordre. **Le tableau est trié.**

2 Terminaison

Un problème courant en informatique est que nous disposons des programmes qui fonctionnent mal car ils tournent « infiniment ». Ces codes comportent des erreurs en général dans les boucles (mauvais contrôle ou gestion des itérations).
Pour prouver qu'un algorithme *prend fin*, on parle de terminaison. On utilise pour cela une notion semblable à un *compte à rebours* :
Lors de l'étude d'une boucle dans le programme :

On identifie une grandeur particulière que l'on nomme ***variant***

Le variant d'une boucle doit :

a) prendre uniquement des valeurs entières ;
b) avoir une valeur positive avant la boucle ;
c) décroître strictement à chaque nouvelle itération.

Si ces prérequis sont vérifiés, comme dans un décompte, le variant finira par atteindre une valeur nulle ou négative et l'algorithme s'achèvera. Cela permet de montrer de *manière théorique* qu'un code se termine.

Preuve de terminaison de l'algorithme de tri par insertion

- La boucle ***Tant que*** dans l'algorithme s'achève toujours (*elle contient au plus TailleMax itérations)*.
- De même, la boucle ***Pour TC allant de 1 à TailleMax - 1*** produit *TailleMax-1* itérations. Donc, l'algorithme se termine après un nombre fini d'étapes.

Preuve de terminaison de l'algorithme de tri par sélection

Nous disposons de **deux boucles for imbriquées** qui forcément, terminent :

- une boucle for pour tri_selection
- une boucle for pour indice_min

3 Complexités

Complexité tri par insertion

Complexité pour n éléments:

- le corps de la boucle est exécuté n − 1 fois
- Une itération :
  - Recherche de la position : p
  - Décalage des éléments : TC − p
  - Total : TC

Au total : $\sum_{i=1}^{i=n-1} i = \frac{n(n-1)}{2}$. La complexité du tri par insertion est en $O(n^2)$

Complexité tri par sélection

L'opération fondamentale est la comparaison de 2 éléments.
Si la longueur du tableau est n alors on fait n-1 comparaison puis n-2, jusqu'à 1 comparaison.
Donc :

$$\sum_{0}^{n-2}(n-i-1)=\sum_{1}^{n-1} i=n*(n-1)/2$$

La complexité est donc pour le tri par sélection : $O(n^2)$

6 Autres type de tris

Évoquer d'autres types de tris comme le tri par fusion ou tri rapide pour notamment expliquer qu'il existe des tris plus efficaces (qui pourront être vus en terminale).
La présentation peut-être faite à l'aide de ce tableau.

Algorithme	Au pire comparaisons	En moyenne comparaisons	Au pire déplacements	En moyenne déplacements
Tri sélection	O(n^2)	O(n^2)	O(n)	O(n)
Tri insertion (séquentielle)	O(n^2)	O(n^2)	O(n^2)	O(n^2)
Tri insertion dichotomique	O(n log n)	O(n log n)	O(n^2)	O(n^2)
Tri bulles	O(n^2)	O(n^2)	O(n^2)	O(n^2)
Tir rapide	O(n^2)	O(n log n)	O(n^2)	O(n log n)
Tri fusion	O(n log n)	O(n log n)	O(n log n)	O(n log n)

Ch. Froidevaux - DIU EIL - Algorithmique 2

7 Mini projet

Voir fiche élève séparée.

Sujet du projet: Tri des communes de France en fonction de la population.

Pour cela, adapter le programme de tri déjà écrit, soit par sélection soit par insertion.

On part d'un fichier contenant une liste de Tuples à 3 éléments dans l'ordre suivant : (N° de département, commune, population)

Exemple : (01, Ambérieu-en-Bugey, 14518)

Concepts techniques mobilisés :
Algorithmes de tris sélection et insertion.
Tuples
Lecture et écriture de fichier CSV
Utilisation de compteurs de temps.

Nombre d'élèves par équipe : 2
Durée : 4h
Travail à rendre : fichier commenté.

Déroulé séance 1 (2 heures) :
présentation du projet.
Programmation lecture et écriture du ficher test (un département)
Bilan d'étape

Déroulé séance 2 (2 heures) :
codage du tri
essai sur fichier test (un département)
Codage de la mesure du temps de traitement
Essai sur les 4 fichiers type (1 département, 5 départements, 10 départements, tous les départements)
Mesure du temps de traitement
Faire le lien avec la notion de coût abordé auparavant.
Livraison du code .

Modalités d'évaluation :
Évaluation du travail rendu selon les critères suivants :

- Travail réussi
- Clarté du code et des commentaires
- participation au travail