SAE Comparaison d’approches Algorithmiques

Implémentation des algorithmes de recherche

Sommaire

Contexte

Le but de cette SAE est de charger une liste d'Étudiants à partir d'un fichier pour ensuite implémenter plusieurs algorithmes de recherche :

- La recherche sans rupture
- La recherche avec rupture
- La recherche dichotomique

Explication des structures de données / classes

Nous avons donc utilisé plusieurs classes

- *Etudiant*
- *StudentManager*
- *ListeEtudiants*
- LibTri2
- TNP
- Search
- Result

Etudiant

Cette classe nous permet de représenter un étudiant par son prénom, son nom et par son groupe. Cela nous permet d'utiliser ces informations dans d'autres classes.
Le constructeur ne requiert que trois éléments : le prénom de l'étudiant (un String), son nom de famille (un String), et son groupe TP (un String).
Nous avons décidé de ne pas stocker le groupe TD comme une variable séparée puisque vous pouvez facilement le déterminer en utilisant le groupe TP.
Le constructeur requiert donc une variable appelée groupe, qui est une chaîne de caractères (pour plus de commodité).
Lorsque le constructeur est appelé, le groupe est converti et puis stocké sous forme de tableau de deux caractères. Nous avons pris la décision de stocker le groupe sous cette forme là puisque nous savons que tous les groupes sont forcément composés de seulement deux caractères et cela nous permet de prendre seulement la place nécessaire en mémoire.

StudentManager

Le constructeur de la classe requiert un entier appelé size.
Lorsque le constructeur est appelé, un TNP est formé avec la variable size.
La classe peut ensuite être utilisée pour gérer nos étudiants en utilisant les méthodes suivantes:
createStudent(Etudiant student) ; créer un étudiant et l'ajoute au TNP
sort() ; trie notre tableau d'étudiants
getTPGroup(Etudiant student) ; retourne le groupe TP d'un étudiant donné
getTDGroup(Etudiant student) ; renvoie le groupe TD d'un étudiant donné
showStudents() ; affiche tous les étudiants dans la console
getStudents() ; retourne le TNP des étudiants
Note: le contenu de cette classe aurait pu être mis directement dans la classe TNP, mais nous avons souhaité préserver le fonctionnement de la classe TNP que nous avions eu en TP (Une classe simple avec uniquement un tableau et un nombre d'éléments)

ListeEtudiants

Cette classe nous permet de charger une liste d'étudiants à partir d'un CSV donné.
La classe a été légèrement modifiée pour permettre de stocker le groupe TP.
Le programme principal de cette classe charge la liste des étudiants sans accents et exécute une série de tests.

Lib2Tri

Cette classe est déjà fournie mais plusieurs modifications ont été effectuées.
Le code du tri simple a été adapté pour trier par groupe, puis par nom de famille, et enfin par prénom à l'aide de la fonction *isBigger().*

Search

Cette classe est utilisée pour implémenter les algorithmes de recherche, elle contient également une série de tests qui permettent de déterminer la rapidité des programmes.

Pour chacunes de ces fonctions on retourne un type 'Result' afin de pouvoir retourner à la fois une valeur de type boolean (si l'étudiant a été trouvé) ainsi qu'une valeur de type int (nombre de comparaisons effectués)

Explication des structures de contrôles :

testSansRupture(TNP tab, Etudiant student);
On crée une variable "found" qu'on initialise à false.
Pour chaque élément du tableau, si l'élément est identique on définit found comme étant à "true"
On retourne ensuite un type 'Result 'qui contient la variable found ainsi que le nombre de comparaisons.

testSansRupture(TNP tab, Etudiant student);
On crée une variable "found" qu'on initialise à false.
Pour chaque élément du tableau, si l'élément est identique on retourne directement vrai dans le type Result (ainsi que le nombre de comparaisons)
Si on arrive à la fin de la boucle, on retourne faux dans notre type Result.

dichotomie(TNP tab, Etudiant student);
On crée une variable "start" ,qu'on initialise à 0,qui représente de le début du tableau.
On crée une variable "end", qu'on initialise à la taille max -1, qui représente la fin du tableau.
On démarre la boucle "while" en initialisant une variable "center" qui se place au milieu du tableau.
Tant que l'indice de start est inférieur à celui de end, on continue la boucle.
On incrémente les valeurs des compteurs par rapport au nombre de boucles faites.
On récupère l'étudiant se situant au milieu de ce tableau (à l'indice "center") et on le compare avec l'étudiant choisi.

Si les 2 étudiants sont identiques alors on sort de la boucle en retournant "true" et le nombre d'itération de "compteur". (ici il vaut 2: une fois pour la boucle while et une fois pour la boucle si).
Si l'étudiant choisit est plus grand alors on incrémente de 1 le compteur et on passe la variable "start" à center+1. (on ne garde que la moitié supérieure).
Sinon, si l'étudiant choisit est plus petit, on incrémente de 1 le compteur et on passe la variable "start" à center-1. (on ne garde que la moitié inférieure)

Glossaire

English word	Definition	Mot français	Définition
Search algorithm	In the field of computer science, a search algorithm is a process that uses a set of mathematical instructions or rules to find information within a set of data	Algorithme de recherche	Dans le domaine de l'informatique, Un algorithme de recherche est un processus qui utilise un ensemble d'instructions ou de règles mathématiques pour trouver des informations dans un ensemble de données.
Algorithmic complexity	The complexity of an algorithm is used to determine the scalability of an	Complexité algorithmique	La complexité d'un algorithme est utilisée pour déterminer

	algorithm. It is measured by how long an algorithm would take to complete a certain task based on an input They are noted with the big o notation for instance O(1) means that no matter what the input, the execution time is the same.		l'évolutivité d'un algorithme. Elle est mesurée par le temps que prendrait un algorithme pour accomplir une certaine tâche à partir d'une entrée. Elles sont notées avec la notation du grand o par exemple O(1) signifie que, quelle que soit l'entrée, le temps d'exécution est le même.
Elementary operation	An elementary operation is an operation located among the 4 elementary operations which are addition, subtraction, multiplication, division.	Opération élémentaire	Une opération élémentaire est une opération se situant parmi les 4 opérations élémentaires qui sont l'addition, la soustraction, la multiplication, la division.
Sorting algorithm	A sorting algorithm is, in computer science or mathematics, an algorithm that organizes a collection of objects according	Algorithme de tri	Un algorithme de tri est, en informatique ou en mathématiques, un algorithme qui permet d'organiser

	to a determined ordering relationship.		une collection d'objets selon une relation d'ordre déterminée.
Data structure	In computer programming, a structure or struct is a method of complex data modeling and representation. it allows multiple variables to be grouped together under one name. Example : Etudiant is a data type that contains three variables (firstName, lastName, group)	Structure de données	En programmation informatique, une structure est une méthode de modélisation et de représentation de données complexes. Elle permet de regrouper plusieurs variables sous un même nom. Exemple : Etudiant est un type de données qui contient trois variables (firstName, lastName, group)
Binary Search	In computer science, binary search is a fast search algorithm that can be used to find the position of a target value within an array as long as it is sorted . The search consists of halving the search interval by checking the middle of the array.	Recherche dichotomique	En informatique, la recherche dichotomique est un algorithme de recherche rapide qui peut être utilisé pour trouver la position d'une valeur cible dans un tableau, à condition qu'il soit trié . La recherche consiste à diviser par deux l'intervalle de recherche en vérifiant le milieu du tableau.

Unit Tests	A unit test is a type of software testing where individual components of a software are tested. The objective is to make sure that the program behaves as intended.		Un test unitaire est un type de test de logiciel où les composants individuels d'un logiciel sont testés. L'objectif est de s'assurer que le programme se comporte comme prévu.

Jeux d'essais

Pour comparer les performances des méthodes de recherches nous allons utiliser deux fichiers avec à chaque fois le jeu d'essai suivant :

1) On cherche un étudiant qui n'existe pas pour s'assurer que les programmes fonctionnent correctement

2) On recherche un étudiant en se mettant le mauvais groupe pour s'assurer qu'il ne le trouve pas.

3) On recherche un étudiant en se mettant le mauvais nom de famille pour s'assurer qu'il ne le trouve pas.

4) On recherche un étudiant en se mettant le mauvais prénom pour s'assurer qu'il ne le trouve pas.

5) On recherche ensuite deux cas où l'étudiant existe

6) On recherche ensuite un cas où l'étudiant est au centre de la liste pour afficher le meilleur cas pour la recherche dichotomique.

7) On recherche le cas où l'étudiant est au début de la liste. Il s'agit du meilleur cas pour la recherche avec rupture.

8) On recherche ensuite un cas où l'étudiant est dernier de la liste pour afficher le pire cas pour le tri avec rupture.

On commence par faire un test simple avec seulement 10 étudiants :

Liste des étudiants :

Liste des étudiants (non trié)

Epahinteresse	Gael	1	1A
Afritt	Barack	1	1A
Onette	Camille	1	1A
Ade	Jeremy	1	1B
Bien	Samira	2	2A
Za	Pit	5	5B
Verse	Alain	1	1B
Agere	Tex	1	1B
Trahuile	Phil	5	5B
Naline	Andre	1	1A

Jeu d'essais

1) *runTest(new Etudiant("Troisieme","Sortie","3B"),studentManager);*

	Trouvé	Nb Comparaison
Test sans rupture	non	20
Test avec rupture	non	20
Test dicho	non	9

2) *runTest(new Etudiant("Tex","Agere","3B"),studentManager);*

	Trouvé	Nb Comparaison
Test sans rupture	non	20
Test avec rupture	non	20
Test dicho	non	9

3) *runTest(new Etudiant("Tex","Verse","1B"),studentManager);*

	Trouvé	Nb Comparaison
Test sans rupture	non	20
Test avec rupture	non	20
Test dicho	non	12

4) *runTest(new Etudiant("Phil","Agere","1B"),studentManager);*

	Trouvé	Nb Comparaison
Test sans rupture	non	20
Test avec rupture	non	20
Test dicho	non	9

5) *runTest(new Etudiant("Samira","Bien","2A"),studentManager);*

	Trouvé	Nb Comparaison
Test sans rupture	oui	20
Test avec rupture	oui	16
Test dicho	oui	5

5) *runTest(new Etudiant("Andre","Naline","1A"),studentManager);*

	Trouvé	Nb Comparaison
Test sans rupture	oui	20
Test avec rupture	oui	6
Test dicho	oui	8

6) *runTest(new Etudiant("Jeremy","Ade","1B"),studentManager);*

	Trouvé	Nb Comparaison
Test sans rupture	oui	20
Test avec rupture	oui	10
Test dicho	oui	2

7) *runTest(new Etudiant ("Barack", "Afritt","1A"), studentManager);*

	Trouvé	Nb Comparaison
Test sans rupture	oui	20
Test avec rupture	oui	2
Test dicho	oui	8

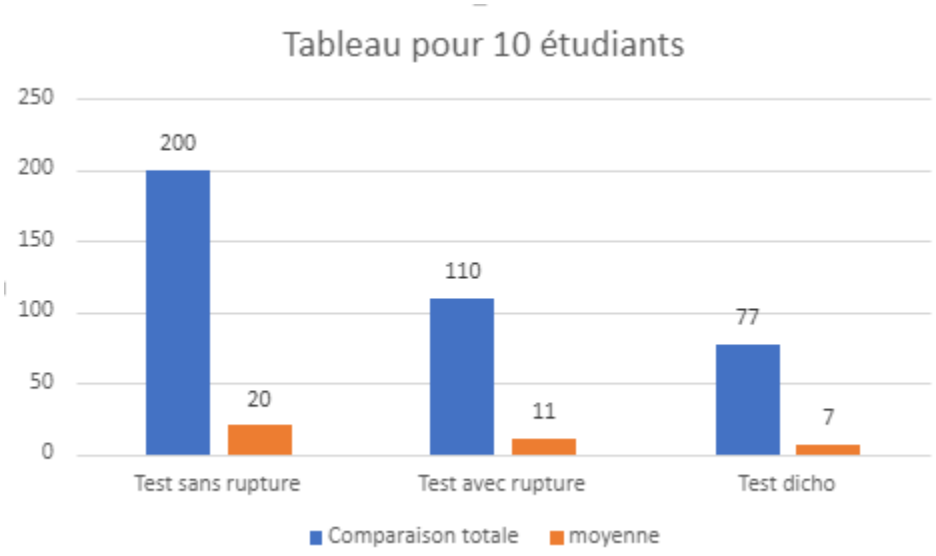
8) *runTest(new Etudiant ("Pit", "Za","5B"),studentManager);*

	Trouvé	Nb Comparaison

Test sans rupture	oui	20
Test avec rupture	oui	20
Test dicho	oui	11

Tableau global pour 10 étudiants:

Test pour 10 étudiants	Comparaison totale	moyenne
Test sans rupture	200	20
Test avec rupture	110	11
Test dicho	77	7



On fait ensuite un test avec 199 étudiants :

Jeu d'essais

1) runTest(new Etudiant("Troisieme","Sortie","3B"),studentManager2);

	Trouvé	Nb Comparaison
Test sans rupture	non	398
Test avec rupture	non	398
Test dicho	non	24

2) runTest(new Etudiant("Garcin","Lazare","3B"),studentManager2);

	Trouvé	Nb Comparaison
Test sans rupture	non	398
Test avec rupture	non	398
Test dicho	non	24

3) runTest(new Etudiant("Garcin","Hannibal","1A"),studentManager2);

	Trouvé	Nb Comparaison
Test sans rupture	non	398
Test avec rupture	non	398
Test dicho	non	24

4) runTest(new Etudiant("Farid","Lazare","1A"),studentManager2);

	Trouvé	Nb Comparaison
Test sans rupture	non	398
Test avec rupture	non	398

Test dicho	non	24

5) runTest(new Etudiant("Nordine","Ateur","1B"),studentManager2);

	Trouvé	Nb Comparaison
Test sans rupture	oui	398
Test avec rupture	oui	62
Test dicho	oui	14

5) runTest(new Etudiant("Paul","Chtron","2A"),studentManager2);

	Trouvé	Nb Comparaison
Test sans rupture	oui	398
Test avec rupture	oui	102
Test dicho	oui	20

6) runTest(new Etudiant("Sam","Gratte","3B"),studentManager2);

	Trouvé	Nb Comparaison
Test sans rupture	oui	398
Test avec rupture	oui	200
Test dicho	oui	20

7) runTest(new Etudiant ("Barack", "Afritt","1A"), studentManager2);

	Trouvé	Nb Comparaison
Test sans rupture	oui	398
Test avec rupture	oui	2

Test dicho	oui	20

8) runTest(new Etudiant ("Pit", "Za","5B"),studentManager2);

	Trouvé	Nb Comparaison
Test sans rupture	oui	398
Test avec rupture	oui	398
Test dicho	oui	23

Tableau global pour 199 étudiants:

Test pour 199 étudiants	Comparaisons totale	Moyenne
Test sans rupture	79202	398
Test avec rupture	39800	200
Test dicho	3836	19

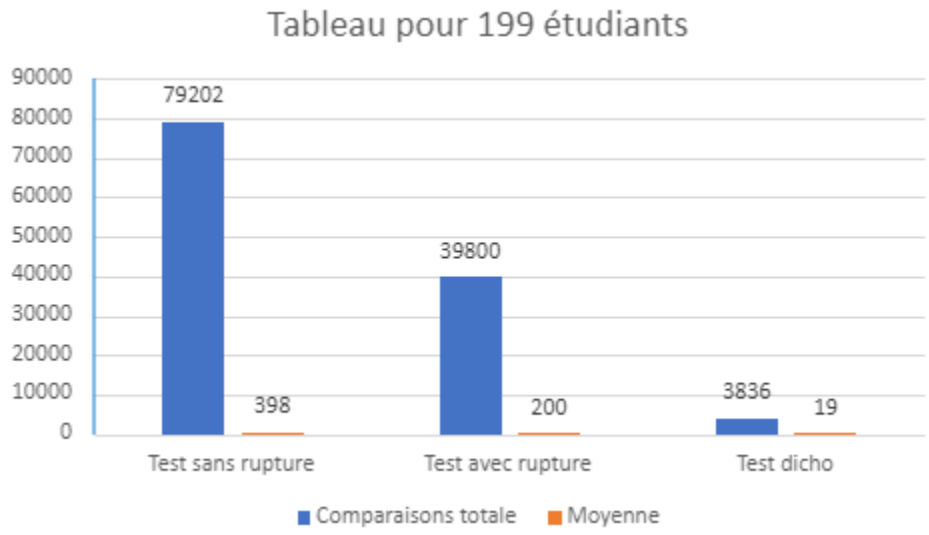


Tableau pour 199 étudiants
90000
80000
70000
60000
50000
40000
30000
20000
10000
0
79202
398
39800
200
3836
19
Test sans rupture
Test avec rupture
Test dicho
Comparaisons totale
Moyenne

Conclusion

De ces résultats, nous pouvons conclure ce qui suit :

- La recherche sans rupture est toujours plus lente ce qui peut s'expliquer par le fait qu'elle vérifie tous les éléments même si elle trouve l'élément qu'elle cherche.

- La recherche avec rupture est, dans presque tous les cas, plus rapide que la "recherche sans rupture" puisqu'elle arrête de vérifier dès qu'elle trouve sa cible.

- La recherche dichotomique est, quant à elle, presque toujours plus rapide puisqu'elle coupe de moitié son intervalle de recherche de façon répétée. Par ailleurs, on remarque grâce aux deux exemples que l’on a testé que plus il y a d’éléments dans la liste, plus la recherche dichotomique est rapide par rapport aux autres.

La recherche dichotomique doit donc être privilégiée dans les situations où le nombre de valeurs à stocker est élevé.