

Comparaison d'Algos

SAE 1.02



BOULOUIHA GNAOUI Yassir
CLAVEL Simon
Ceolin Thomas

Informatique 1^{ière} Année

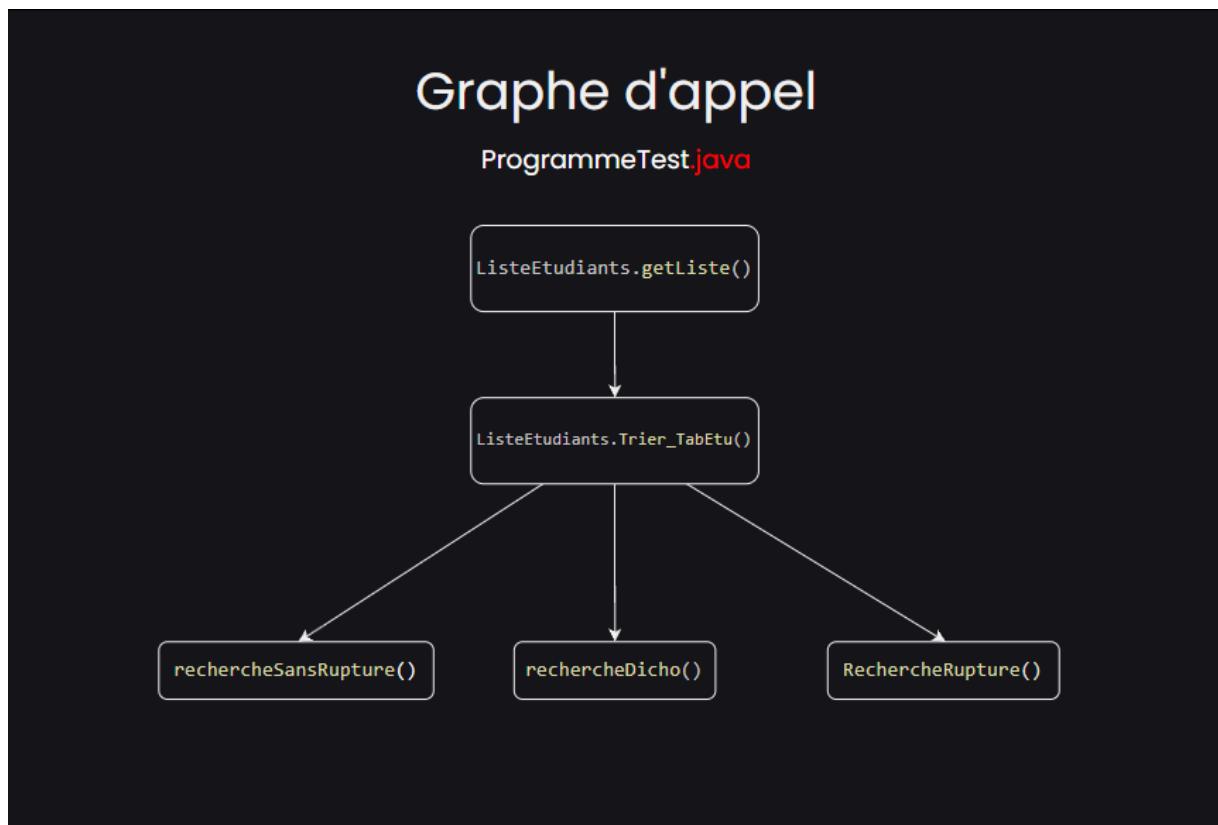
Table des matières :

Présentation de l'application.....	2
Structure de données utilisée.....	4
Découpage du Travail.....	5
Présentation argumentée des expérimentations réalisées :.....	6
Recherche avec-rupture.....	7
Recherche Sans Rupture.....	9
Recherche Dichotomique.....	12
Interprétation des résultats	12
Conclusion tirée	13
Tableau de synthèse des expérimentations.....	14
Conclusion.....	18
Annexes.....	19
Dichotomie.....	19
Tests pour 10000 étudiants :.....	19
Tests pour 1000 étudiants :.....	20
Tests pour 10 étudiants :.....	21

Présentation de l'application

Dans le cadre de la SAE 1.02 nous avons mis au point plusieurs programmes permettant de récupérer une liste de X étudiants d'un fichier déjà donné en format csv, de les placer dans une structure de données que nous avons imaginé nous permettant d'effectuer une recherche d'étudiant dans cette liste via des sous-programmes utilisant différents algorithmes de recherche tels que la dichotomie, la recherche simple et la recherche avec rupture afin d'étudier la complexité de ceux-ci.

Nous avons donc mis en place différents sous-programmes permettant cela, voici le graphe d'appel de notre application :

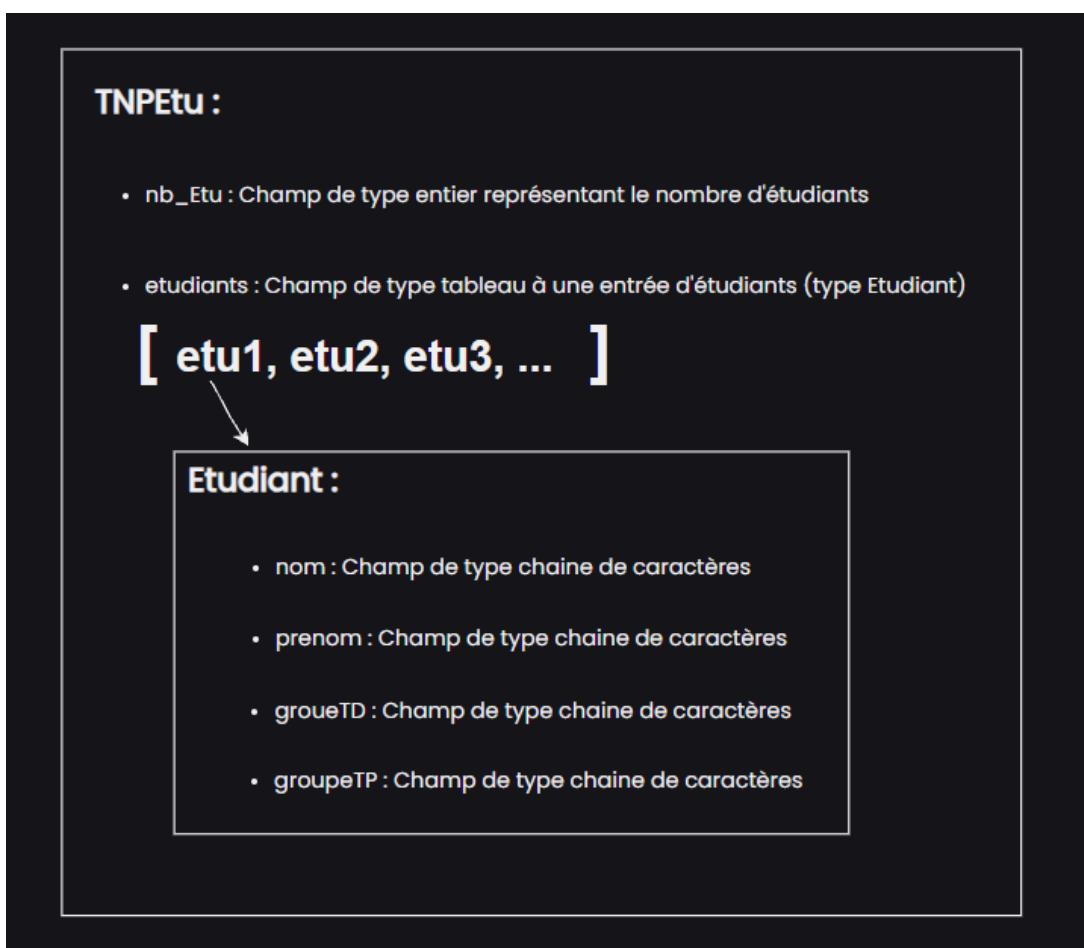


Tous nos tests sont faits dans le main de la classe ProgrammeTest.java, nous faisons appel à différents types d'objets et à des sous-programmes dans d'autres classes depuis ce programme afin d'effectuer nos tests de complexité algorithmique. Dans un premier temps nous appelons la fonction getListe() qui va permettre de charger la liste d'étudiants d'un fichier csv vers le tableau contenant des objets de type Etudiant constituant lui-même un champ de l'objet TNPEtu.

Avant d'appeler cette première fonction nous initialisons donc un objet de type TNPEtu afin de le remplir avec un tableau à une dimension d'étudiants et le nombre d'étudiants contenus dans cet objet. Suite à cela on appelle le sous-programme Trier_TabEtu() qui permet de trier le tableau d'étudiants par ordre alphabétique. Une fois les données chargées et triées nous n'avons plus qu'à faire nos tests avec les sous-programmes de recherche de notre choix que ce soit la dichotomie, recherche avec rupture, recherche simple ou les trois en même temps.

Structure de données utilisée

Nous vous avons parlé un peu plus haut de deux objets que l'on a manipulé pour charger la liste des étudiants et effectuer nos tests, nous allons rentrer un peu plus dans les détails en vous présentant plus en profondeur notre structure de données.



Notre structure des données repose sur deux objets. Un objet TNPEtu qui contient deux champs, le premier étant un entier représentant le nombre d'étudiants chargés à un instant t dans le tableau contenant des objets de type Etudiant qui constitue le deuxième champ de cet objet (TNPEtu). La classe étudiant représente les informations qui constituent un étudiant dans la promo donc : son nom, son prénom, son groupe de TD et de TP.

Au début on avait fait le choix de mettre un type int pour le groupe de TD et un char pour le groupe de TP, mais lorsqu'on a expérimenté la fonction getListe on avait des erreurs étant donné que les données lues étaient en format String (chaîne de caractères), on a fait le choix de tout basculer en String pour ne pas avoir ces problèmes de typage de nos champs lors du chargement des données.

Découpage du Travail

Yassir : Structure de données, sous-programme de tri Trier_TabEtu(), programme de recherche dichotomique, adaptation du programme de chargement des données getListe(), présentation de la structure de données, présentation de l'application, recherche dichotomique

Simon : Sous-programme de recherche avec rupture, Présentation argumentée des expérimentations réalisées, conclusion

Thomas : Sous programme de Recherche Sans rupture, Test algorithmique du sous programme + ajustement, Présentation du Tableau de synthèse d'expérimentation entre les 3 sous programmes

Présentation argumentée des expérimentations réalisées :

Ce jeu a été créé en comparant trois algorithmes de recherche différents avec différentes listes plus ou moins longues (de 10 éléments à 100 000 éléments). Nous avons comparé la recherche avec rupture et sans rupture ainsi que la recherche dichotomique. Pour pouvoir observer et comparer les différents résultats des différents algorithmes nous avons donc cherché à savoir le nombre de tours dans une boucle, ainsi que le nombre de comparaisons, aussi nous avons cherché à renvoyer l'occurrence de la recherche pour savoir si le résultat était bien juste et si l'élément que nous recherchons n'était pas là nous affectons donc l'indice à -1 pour dire qu'il n'a pas été trouvé.

Ce jeu est donc pertinent pour pouvoir comparer les différents algorithmes en comparants pour chacun leurs tours de boucle et leur nombre de comparaisons pour chaque cas : avec plus ou moins d'éléments, quand l'élément est au début ou à la fin et aussi quand l'élément n'est pas présent dans le tableau. Donc l'algorithme qui sera le plus optimal à la fin des comparaisons sera donc l'algorithme qui aura fait le moins de comparaisons et le moins de tours de boucle tout au long de la recherche de l'élément.

Recherche avec-rupture

Une recherche avec rupture est une recherche qui consiste à trouver un élément dans un tableau et lorsque le programme tombe sur un élément égal, celui-ci sort donc de la boucle et arrête de parcourir le tableau. Il s'arrête donc toujours au premier élément égal d'un tableau et sort ensuite de celui-ci.

élément au début	Trouver	nombres de tours	nombres de comparaisons	indice de recherche
10	true	1	2	0
100	true	1	2	0
1 000	true	1	2	0
5 000	true	1	2	0
10 000	true	1	2	0
100 000	100 000	1	2	0

élément à la fin	Trouver	nombres de tours	nombres de comparaisons	indice de recherche
10	true	10	11	9
100	true	100	101	99
1 000	true	1 000	1 001	999
5 000	true	5 000	5 001	4 999
10 000	true	10 000	10 001	9 999
100 000	true	100 000	100 001	99 999

élément au milieu	Trouver	nombres de tours	nombres de comparaisons	indice de recherche
10	true	5	6	4
100	true	50	51	49
1 000	true	500	501	499
5 000	true	2 500	2501	2 499
10 000	true	5 000	5 001	4 999
100 000	true	50 000	50 001	49 999

pas l'élément	Trouver	nombres de tours	nombres de comparaisons	indice de recherche
10	false	10	10	-1
100	false	100	100	-1
1 000	false	1 000	1 000	-1
5 000	false	1 000	1 000	-1
10 000	false	10 000	10 000	-1
100 000	false	100 000	100 000	-1

On peut observer que si il y a plusieurs même nom dans la liste donné le programme effectuera plus de comparaison car il comparera des prénoms en plus (sauf si le premier nom a le bon prénom).

Aussi il y a donc en général plus de comparaisons que de tours de boucles et le nombre de tour de boucle est plus grand que le nombre de l'indice de Recherche. Cela est normal puisque lorsque l'élément est trouvé nous faisons une comparaison en plus avec le prénom.

Cependant si l'élément n'est pas trouvé il y a autant de tour de boucle que de comparaisons car nous n'avons jamais comparé avec un prénom en plus.

Aussi pour 100 000 éléments si l'on utilise le tri avant la recherche le programme ne marche pas car il y a trop de calcul à faire.

Recherche Sans Rupture



TEST 10K d'Étudiant

Voici un test du programme RechercheSansRupture qui permet donc de chercher des étudiants dans une liste

```
nombre de lignes/etudiants : 10000
nombre de lignes : 10000
Erreur : L'étudiant n'a pas été trouvé !
Le nombre de tour de boucle a été de 10000 Le nombre de comparaisons est : 10001
#####
L'étudiant a été trouvé au tour de boucle 0 etu : ABADIA KURT 10
Le nombre de tour de boucle a été de 10000 Le nombre de comparaisons est : 10001
#####
L'étudiant a été trouvé au tour de boucle 4999 etu : JAUFFRET ANDERS 20
Le nombre de tour de boucle a été de 10000 Le nombre de comparaisons est : 10001
#####
L'étudiant a été trouvé au tour de boucle 9999 etu : ŠIMKONIS NANOU 4
Le nombre de tour de boucle a été de 10000 Le nombre de comparaisons est : 10001
```



TEST 5k d'Étudiant

```
nombre de lignes/etudiants : 5000
nombre de lignes : 5000
Erreur : L'étudiant n'a pas été trouvé !
Le nombre de tour de boucle a été de 5000 Le nombre de comparaisons est : 5001
#####
L'étudiant a été trouvé au tour de boucle 0 etu : ABADIA KURT 10
Le nombre de tour de boucle a été de 5000 Le nombre de comparaisons est : 5001
#####
L'étudiant a été trouvé au tour de boucle 2499 etu : JALICOT FILIP 14
Le nombre de tour de boucle a été de 5000 Le nombre de comparaisons est : 5001
#####
L'étudiant a été trouvé au tour de boucle 4999 etu : ÖRNDAHL DUNCAN 8
Le nombre de tour de boucle a été de 5000 Le nombre de comparaisons est : 5001
```



TEST 1K d'Étudiant

```
nombre de lignes/etudiants : 1000
nombre de lignes : 1000
Erreur : L'étudiant n'a pas été trouvé !
Le nombre de tour de boucle a été de 1000 Le nombre de comparaisons est : 1002
#####
L'étudiant a été trouvé au tour de boucle 0 etu : ABDALLAH Béatrice 13
Le nombre de tour de boucle a été de 1000 Le nombre de comparaisons est : 1001
#####
L'étudiant a été trouvé au tour de boucle 499 etu : HOURTANE Robin 3
Le nombre de tour de boucle a été de 1000 Le nombre de comparaisons est : 1002
#####
L'étudiant a été trouvé au tour de boucle 999 etu : ÂGESEN ANNABELLE 18
Le nombre de tour de boucle a été de 1000 Le nombre de comparaisons est : 1001
```



TEST 100 d'Étudiant

```
nombre de lignes/etudiants : 100
nombre de lignes : 100
Erreur : L'étudiant n'a pas été trouvé !
Le nombre de tour de boucle a été de 100 Le nombre de comparaisons est : 101
#####
L'étudiant a été trouvé au tour de boucle 0 etu : AIELLO Lisa 4
Le nombre de tour de boucle a été de 100 Le nombre de comparaisons est : 101
#####
L'étudiant a été trouvé au tour de boucle 49 etu : KURTKOWIAK YOANN 4
Le nombre de tour de boucle a été de 100 Le nombre de comparaisons est : 101
#####
L'étudiant a été trouvé au tour de boucle 99 etu : ZIEMNIAK ALEKSANDRA 3
Le nombre de tour de boucle a été de 100 Le nombre de comparaisons est : 101
```



TEST 10 d'Étudiant

```
nombre de lignes/etudiants : 10
nombre de lignes : 10
Erreur : L'étudiant n'a pas été trouvé !
Le nombre de tour de boucle a été de 10 Le nombre de comparaisons est : 11
#####
L'étudiant a été trouvé au tour de boucle 0 etu : FIEVEZ SUZETTE 4
Le nombre de tour de boucle a été de 10 Le nombre de comparaisons est : 11
#####
L'étudiant a été trouvé au tour de boucle 4 etu : PERRONNET Richard 4
Le nombre de tour de boucle a été de 10 Le nombre de comparaisons est : 11
#####
L'étudiant a été trouvé au tour de boucle 9 etu : ZARSKA LEO 4
Le nombre de tour de boucle a été de 10 Le nombre de comparaisons est : 11
```

On peut constater que chaque test a été un succès, que ce soit pour 10 000 étudiants ou encore pour 1 000 étudiants. Chacun des tests nous fournit le nombre de tours effectués pour trouver l'étudiant ainsi que le nombre de comparaisons effectuées. La fonction de recherche sans rupture est alors opérationnelle, car à chaque test, elle analyse l'ensemble du tableau sans interruption. Cependant, il faut savoir que j'ai tenté d'exécuter le test avec 100 000 étudiants, mais ce test était voué à l'échec, car le pc ne pouvait pas supporter

Recherche Dichotomique

La recherche dichotomique est une technique de recherche rapide qui fonctionne sur des listes triées. Elle consiste à diviser continuellement la liste en deux parties égales jusqu'à ce que l'élément recherché soit trouvé ou que la sous-liste devienne vide. Cette méthode est efficace car elle réduit considérablement le nombre d'éléments à examiner à chaque étape, ce qui permet de trouver l'élément recherché beaucoup plus rapidement que si nous devions examiner chaque élément un par un. C'est pourquoi on l'appelle souvent "recherche binaire".

Interprétation des résultats :

- Pour un nombre d'étudiants plus élevé, le nombre de tours de boucle et de comparaisons augmente généralement, ce qui est attendu car il y a plus de données à traiter.
- Cependant, si l'étudiant recherché se trouve au début ou au milieu de la liste, le nombre de tours de boucle et de comparaisons peut être significativement réduit.
- Dans certains cas où l'étudiant n'est pas trouvé, le nombre de tours de boucle et de comparaisons peut être assez élevé, car l'algorithme doit parcourir l'ensemble des données.

Conclusion tirée :

- La recherche dichotomique est un algorithme efficace pour la recherche dans une grande liste triée, car elle réduit le nombre de comparaisons nécessaires en divisant la liste en deux à chaque itération.
- Cependant, son efficacité dépend de la position de l'élément recherché. Si l'élément est au début ou au milieu de la liste, la recherche peut être plus rapide.
- Par rapport à un algorithme de recherche simple ou avec rupture, la recherche dichotomique peut offrir de meilleures performances, surtout lorsque la taille de la liste est grande. Cependant, si la liste est petite, la différence de performance peut ne pas être significative.

Tableau de synthèse des expérimentations

	Recherche	Liste étudiant	nombre de tour de boucle	Nombre Comparaison	Etudiant Trouvé
Pas trouvé	Dichotomie	10 000	13	39	Non
Début	Dichotomie	10 000	13	38	Oui
Milieux	Dichotomie	10 000	1	2	Oui
FIN	Dichotomie	10 000	14	28	Oui
Pas trouvé	Dichotomie	1000	10	24	non
Début	Dichotomie	1000	9	26	oui
Milieux	Dichotomie	1000	1	2	Oui
FIN	Dichotomie	1000	10	20	Oui
Pas trouvé	Dichotomie	10	4	10	Non
Début	Dichotomie	10	3	8	Oui
Milieux	Dichotomie	10	1	2	Oui

FIN	Dichotomie	10	4	8	Oui
Pas trouvé	Recherche SansRupture	10 000	10 000	10 001	Non
Début	Recherche SansRupture	10 000	10 000	10 001	Oui
Milieux	Recherche SansRupture	10 000	10 000	10 001	Oui
FIN	Recherche SansRupture	10 000	10 000	10 001	Oui
Pas trouvé	Recherche SansRupture	1000	1000	1001	Non
Début	Recherche SansRupture	1000	1000	1001	Oui
Milieux	Recherche SansRupture	1000	1000	1001	Oui
FIN	Recherche SansRupture	1000	1000	1001	Oui
Pas trouvé	Recherche SansRupture	10	10	11	Non
Début	Recherche SansRupture	10	10	11	Oui

Milieux	Recherche SansRupture	10	10	11	Oui
FIN	Recherche SansRupture	10	10	11	Oui
Pas trouvé	Recherche Rupture	10000	10000	10000	Non
Début	Recherche Rupture	10000	1	2	Oui
Milieux	Recherche Rupture	10000	5000	5001	Oui
FIN	Recherche Rupture	10000	10000	10001	Oui
Pas trouvé	Recherche Rupture	1000	1000	1000	Non
Début	Recherche Rupture	1000	1	2	Oui
Milieux	Recherche Rupture	1000	500	501	Oui
FIN	Recherche Rupture	1000	1000	1001	Oui
Pas trouvé	Recherche Rupture	10	10	10	Non
Début	Recherche Rupture	10	1	2	Oui
Milieux	Recherche Rupture	10	5	6	Oui

FIN	Recherche Rupture	100	10	11	Oui
-----	-------------------	-----	----	----	-----

Une fois que l'on analyse le tableau de synthèse de l'expérimentation, on peut voir la grande différence entre les trois recherches, que ce soit entre la dichotomie et la recherche avec rupture, ou encore entre la dichotomie et la recherche sans rupture. On constate donc que la dichotomie a une performance nettement meilleure que la recherche sans rupture ou encore avec rupture. Cependant, lorsque la recherche se fait dans un tableau avec beaucoup d'éléments, la recherche sans rupture est plus optimale que la recherche dichotomique. À part cette exception, la recherche dichotomique est la plus adaptée.

Conclusion

Ce jeu est donc pertinent pour connaître la complexité des différents algorithmes. En effet, nous pouvons observer dans un premier temps que l'algorithme le moins complexe est donc celui sans-rupture, car dans tous les cas il parcourt tous les éléments du tableau même s'il a déjà trouvé l'élément. Ensuite en comparant les recherches dichotomique et avec-rupture, on peut constater que la recherche dichotomique est en général la plus optimale. En effet, elle fait moins de comparaisons et moins de tours de boucle.

Cependant, pour une recherche dans un très grand tableau, la recherche avec rupture peut être meilleure que la recherche dichotomique si l'élément recherché se trouve au début du tableau.

Pour conclure, si vous souhaitez rechercher un élément dans un tableau nous vous recommandons donc d'utiliser la recherche dichotomique pour avoir une recherche plus optimale.

Annexes

Dichotomie

Tests pour 10000 étudiants :

Etudiant non existant :

```
L'étudiant JAUFFRET KURT n'a pas été trouvé !
Infos d'exécution :
-10000 étudiants
-13 tour(s) de boucle
-39 comparaisons
```

Etudiant au début :

```
L'étudiant ABADIA KURT a été trouvé à la position : 0
Il est dans le Groupe : 10
Infos d'exécution :
-10000 étudiants
-13 tours de boucle
-38 comparaisons
```

Etudiant au milieu :

```
L'étudiant JAUFFRET ANDERS a été trouvé à la position : 4999
Il est dans le Groupe : 20
Infos d'exécution :
-10000 étudiants
-1 tour(s) de boucle
-2 comparaisons
```

Etudiant à la fin :

```
L'étudiant À IMKONIS NANOU a été trouvé à la position : 9999
Il est dans le Groupe : 4
Infos d'exécution :
-10000 étudiants
-14 tour(s) de boucle
-28 comparaisons
```

Tests pour 1000 étudiants :

Etudiant non existant :

```
L'étudiant HOURTANE ANNABELLE n'a pas été trouvé !
Infos d'exécution :
-1000 étudiants
-10 tour(s) de boucle
-24 comparaisons
```

Etudiant au début :

```
L'étudiant ABDALLAH BÃ©atrice a été trouvé à la position : 0
Il est dans le Groupe : 13
Infos d'exécution :
-1000 étudiants
-9 tour(s) de boucle
-26 comparaisons
```

Etudiant au milieu :

```
L'étudiant HOURTANE Robin a été trouvé à la position : 499
Il est dans le Groupe : 3
Infos d'exécution :
-1000 étudiants
-1 tour(s) de boucle
-2 comparaisons
```

Étudiant à la fin :

```
L'étudiant ÆGESEN ANNABELLE a été trouvé à la position : 999
Il est dans le Groupe : 18
Infos d'exécution :
-1000 étudiants
-10 tour(s) de boucle
-20 comparaisons
```

Tests pour 10 étudiants :

Etudiant non existant :

```
L'étudiant PERRONNET LEO n'a pas été trouvé !
Infos d'exécution :
-10 étudiants
-4 tour(s) de boucle
-10 comparaisons
```

Etudiant au début :

```
L'étudiant FIEVEZ SUZETTE à été trouvé à la position : 0
Il est dans le Groupe : 4
Infos d'exécution :
-10 étudiants
-3 tour(s) de boucle
-8 comparaisons
```

Etudiant au milieu :

```
L'étudiant PERRONNET Richard à été trouvé à la position : 4
Il est dans le Groupe : 4
Infos d'exécution :
-10 étudiants
-1 tour(s) de boucle
-2 comparaisons
```

Etudiant à la fin :

```
L'étudiant ZARSKA LEO à été trouvé à la position : 9
Il est dans le Groupe : 4
Infos d'exécution :
-10 étudiants
-4 tour(s) de boucle
-8 comparaisons
```

Comparaison d'Algos

Bilan personnel

SAE 1.02



BOULOUIHA GNAOUI Yassir

Informatique 1^{ière} Année

Table des matières :

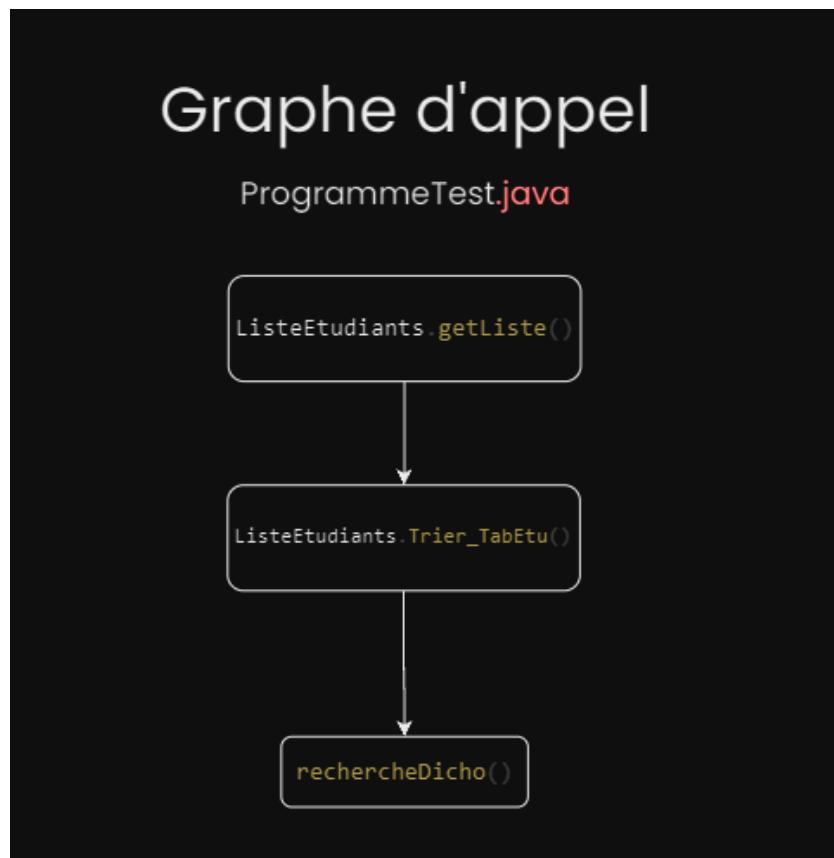
Contexte.....	2
Sous Programmes.....	3
Articulation des programmes.....	3
nbEtudiant().....	3
getListe().....	5
Trier_TabEtu().....	6
rechercheDicho().....	8
Bilan.....	10
Interprétation des résultats.....	10
Conclusion tirée :.....	10
Difficultés & Progression.....	12
Annexes.....	14
Tests pour 10000 étudiants.....	14
Tests pour 5000 étudiants.....	15
Tests pour 1000 étudiants.....	16
Tests pour 100 étudiants.....	17
Tests pour 10 étudiants.....	18

Contexte

Dans le cadre de la SAE 1.02 nous avons été amenés à développer une solution permettant de charger depuis un fichier csv une liste de X étudiants, et de mettre en place des algorithmes permettant la recherche d'étudiants dans cette liste avec des approches différentes, notamment une approche dichotomique pour ma part. Dans un premier temps je vais vous présenter les différents programmes que j'ai effectué lors de cette saé, puis dans un second temps je vais vous donner mon bilan personnel, ce que j'ai appris et les compétences que j'ai pu développer grâce à cette saé.

Sous Programmes

Articulation des programmes



nbEtudiant()

```
/** Retourne le nombre d'étudiants de du TNPEtu pfTNPEtu
 * @param pfTNPEtu IN : Objet de type TNPEtu
 * @return le nombre d'étudiants contenus dans le TNPEtu
 * @author Modifié par BOULOUIHA GNAOUI Yassir
 */
public static int nbEtudiant(TNPEtu pfTNPEtu)
```

Le sous programme nbEtudiant est une fonction publique statique qui prend un objet de type TNPEtu en paramètre et retourne un entier.

L'objet pfTNPEtu est utilisé dans le corps du sous-programme pour accéder à son attribut nbEtu, qui représente le nombre d'étudiants. Le sous programme retourne simplement cette valeur.

En termes d'algorithme non raffiné, le sous-programme fait ce qui suit :

1. Prend un objet TNPEtu en entrée.
2. Accède à l'attribut nbEtu de cet objet, qui représente le nombre d'étudiants.
3. Retourne cette valeur.

Cela signifie que ce sous programme est utilisé pour obtenir le nombre d'étudiants d'un objet TNPEtu. Il ne modifie pas l'objet en question, il se contente d'en extraire une information.

getListe()

```
/** Retourne un tableau à une entrée contenant des objets de type etudiant
chargés grâce au fichier pfFilename et le délimiteur défini par pfDelimiter,
met à jour le nombre d'étudiants du pfTNPEtu passé en paramètres
* @param pfFileName IN : le nom du fichier à lire
* @param pfDelimiter IN : le délimiteur de champs dans le fichier csv
* @param pfTNPEtu IN-OUT : le TNPEtu qui va contenir le tableau d'étudiants
* @return le tableau
* @author Modifié par BOULOUIHA GNAOUI Yassir
*/
public static Etudiant [] getListe(String pfFileName, String pfDelimiter,
TNPEtu pfTNPEtu)
```

Le sous-programme `getListe` est une fonction publique statique qui prend trois paramètres : une chaîne de caractères `pfFileName` qui représente le nom du fichier à lire, une autre chaîne de caractères `pfDelimiter` qui représente le délimiteur de champs dans le fichier csv, et un objet `pfTNPEtu` de type `TNPEtu` qui va contenir le tableau d'étudiants. Le sous-programme retourne un tableau d'objets de type `Etudiant`.

Le code commence par ouvrir le fichier spécifié par `pfFileName` et compte le nombre de lignes, qui correspond au nombre d'étudiants. Ce nombre est ensuite assigné à l'attribut `nbEtu` de l'objet `pfTNPEtu`.

Ensuite, un tableau `TabEtu` de type `Etudiant` est créé avec une taille égale au nombre d'étudiants.

Le fichier est à nouveau lu ligne par ligne. Chaque ligne est divisée en tokens en utilisant le délimiteur spécifié par `pfDelimiter`. Un nouvel objet `Etudiant` est créé avec ces tokens et est ajouté au tableau `TabEtu`.

Finalement, le tableau TabEtu est retourné. Ce sous-programme est donc utilisé pour lire un fichier csv contenant des informations sur des étudiants, créer un tableau d'objets Etudiant à partir de ces informations, et mettre à jour le nombre d'étudiants dans un objet TNPEtu. Il lance une exception FileNotFoundException si le fichier spécifié par pfFileName n'est pas trouvé.

Trier_TabEtu()

```
/** Trie un tableau à une entrée contenant des objets de type Etudiant en
fonction des noms et prenoms
* @param pfTabEtu IN-OUT : Tableau d'une dimension de type Etudiant
* @author BOULOUIHA GNAOUI Yassir
*/
public static void Trier_TabEtu(Etudiant pfTabEtu[])
```

Le sous-programme Trier_TabEtu est une fonction publique statique qui prend un tableau d'objets de type Etudiant en paramètre.

Le sous-programme commence par initialiser une variable tab_triee à false, indiquant que le tableau n'est pas trié. Il crée également une variable temp_etu de type Etudiant pour stocker temporairement un objet Etudiant lors de l'échange des éléments du tableau.

Ensuite, il entre dans une boucle while qui continue jusqu'à ce que le tableau soit trié. À l'intérieur de cette boucle, il suppose d'abord que le tableau est trié en définissant tab_triee à true.

Il parcourt ensuite le tableau avec une boucle for. Pour chaque paire d'objets Etudiant consécutifs dans le tableau, il compare leurs noms. Si le nom du premier objet est alphabétiquement supérieur au nom du second, il échange leurs positions dans le tableau et définit tab_triee à false, indiquant que le tableau n'est pas encore trié.

Si les noms sont identiques, il compare ensuite leurs prénoms. Si le prénom du premier objet est alphabétiquement supérieur au prénom du second, il échange également leurs positions dans le tableau et définit tab_triee à false.

Le sous-programme continue à parcourir et à trier le tableau jusqu'à ce qu'il n'y ait plus d'échanges nécessaires, c'est-à-dire jusqu'à ce que le tableau soit trié par ordre alphabétique des noms et des prénoms. À ce moment-là, tab_triee reste true et la boucle while se termine.

Ce sous-programme est donc utilisé pour trier un tableau d'objets Etudiant par ordre alphabétique des noms et des prénoms. Il ne retourne rien car il modifie directement le tableau passé en paramètre.

rechercheDicho()

```
/** Fonction qui permet de rechercher un étudiant dans une liste donnée avec
la méthode de la recherche dichotomique
* @param pfTNPEtu IN : parametre de type tableau non plein d'étudiants
(TNPEtu)
* @param pfnom IN : parametre de type chaine de charactères représentant le
nom de l'étudiant à rechercher
* @param pfprenom IN : parametre de type chaine de charactères représentant
le prenom de l'étudiant à rechercher
* @return indice_etu : La position à la quelle l'étudiant a été trouvé ou -1
si celui-ci n'a pas été trouvé
* @author BOLOUIHA GNAOUI Yassir
*/
public static int rechercheDicho(TNPEtu pfTNPEtu, String pfnom, String
pfprenom)
```

Le sous-programme rechercheDicho prend trois paramètres : un objet pfTNPEtu de type TNPEtu, une chaîne de caractères pfnom représentant le nom de l'étudiant à rechercher, et une autre chaîne de caractères pfprenom représentant le prénom de l'étudiant à rechercher. Le sous-programme retourne un entier qui est l'indice de l'étudiant recherché dans le tableau d'étudiants de l'objet pfTNPEtu, ou -1 si l'étudiant n'est pas trouvé.

Le sous-programme utilise la méthode de la recherche dichotomique pour trouver l'étudiant. Il initialise d'abord les variables debut, fin, milieu et indice_etu, ainsi que les variables nb_comp et nb_tour pour mesurer la pseudo-complexité de la recherche.

Ensuite, il entre dans une boucle while qui continue tant qu'il reste des éléments à parcourir dans le tableau d'étudiants.

À chaque tour de boucle, il met à jour la variable milieu et compare le nom et le prénom de l'étudiant à la position milieu avec les noms et prénoms recherchés.

Si le nom et le prénom correspondent, il affecte milieu à indice_etu et retourne cette valeur. Sinon, il ajuste les variables debut et fin pour continuer la recherche dans la partie appropriée du tableau.

Si après l'exécution de la recherche, indice_etu n'a pas changé, cela signifie que l'étudiant recherché n'a pas été trouvé, et le sous-programme retourne -1.

Ce sous-programme est donc utilisé pour rechercher un étudiant dans un tableau non plein d'étudiants en utilisant la méthode de la recherche dichotomique. Il ne modifie pas le tableau passé en paramètre. Il retourne l'indice de l'étudiant recherché dans le tableau, ou -1 si l'étudiant n'est pas trouvé. Il affiche également des informations sur l'exécution de la recherche.

Bilan

Interprétation des résultats

D'après le tableau ci-dessous fait grâce aux données des tests (voir [annexes](#)), voici une interprétation des résultats :

- Pour un nombre d'étudiants plus élevé, le nombre de tours de boucle et de comparaisons augmente généralement, ce qui est attendu car il y a plus de données à traiter.
- Cependant, si l'étudiant recherché se trouve au début ou au milieu de la liste, le nombre de tours de boucle et de comparaisons peut être significativement réduit.
- Dans certains cas où l'étudiant n'est pas trouvé, le nombre de tours de boucle et de comparaisons peut être assez élevé, car l'algorithme doit parcourir l'ensemble des données.

Ces résultats concordent avec les informations que j'ai trouvé sur internet.

Conclusion tirée :

- La recherche dichotomique est un algorithme efficace pour la recherche dans une grande liste triée, car elle réduit le nombre de comparaisons nécessaires en divisant la liste en deux à chaque itération.
- Cependant, son efficacité dépend de la position de l'élément recherché. Si l'élément est au début ou au milieu de la liste, la recherche peut être plus rapide.
- Par rapport à un algorithme de recherche simple ou avec rupture, la recherche dichotomique peut offrir de meilleures performances, surtout lorsque la taille de la liste est grande. Cependant, si la liste est petite, la différence de performance peut ne pas être significative.

Comparaison d'Algos

2023/2024

Nombre d'étudiants	Nom de l'étudiant	Position	Tours de boucle	Comparaisons
10	PERRONNET LEO	N/A	4	10
10	FIEVEZ SUZETTE	0	3	8
10	PERRONNET Richard	4	1	2
10	ZARSKA LEO	9	4	8
1000	HOURTANE ANNABELLE	N/A	10	24
1000	ABDALLAH BÃ©atrice	0	9	26
1000	HOURTANE Robin	499	1	2
1000	Ã?GESEN ANNABELLE	999	10	20
10000	JAUFFRET NANOU	N/A	13	39
10000	ABADIA KURT	0	13	38
10000	JAUFFRET ANDERS	4999	1	2
10000	Å IMKONIS NANOU	9999	14	28

Difficultés & Progression

J'ai rencontré quelques petites difficultés au cours de cette saé, il y'a eu dans un premier temps avec le programme getListe(), une soucis quant à la définition du nombre d'étudiants dans l'objet TNPETU, en effet vu que je n'utilisais pas un TNPEtu en paramètres initialement, je remplissais les étudiants dans un tableau tableau à une entrée de type Etudiant que je retournais et que j'affectais ensuite au champ concerné de mon TNPETU via l'appel de cette même fonction, cependant le champ nbEtu censé contenir le nombre d'étudiants actuels dans le TNPEtu n'était pas changé et restait à zéro, j'ai donc ajouté un paramètre TNPEtu et mis à jour le champ nbEtu dans le sous programme getListe afin de régler ce probleme.

J'ai également dû changer le type des champs de ma structure Etudiant une fois les tests du sous programme getListe commencés. Initialement j'avais mis des types int, et char pour groupeTD et groupeTP respectivement, cependant lorsque je venais à insérer les données lues sur le fichier csv celles-ci étaient forcément insérées au type chaîne de caractères (String), j'ai donc modifiée en conséquence les types de ces deux champs en String afin de régler le problème.

A part cela je n'ai pas vraiment eu de gros soucis qui m'ont freiné un bon bout de temps, il y a eu le code du sous programme de recherche dichotomique qui était un peu délicat à faire notamment sur la recherche des prénoms lorsque les noms correspondent, je pensais initialement faire des indices différents de recherche pour le nom et le prénom, cependant après un peu de réflexion j'ai réussi à le faire après avoir compris qu'il n'était pas du tout nécessaire d'utiliser des indices de recherche différents pour le nom et le prenom.

Compétence 2

Appréhender et construire des algorithmes

AC 1 Analyser un problème avec méthode (découpage en éléments algorithmiques simples, structure de données...)

AC 2 Comparer des algorithmes pour des problèmes classiques (tris simples, recherche...)

Mis à part ces difficultés, faire la structure de données m'a permis de monter en compétences notamment en AC 1. Faire le sous-programme de tri, ainsi que celui de la recherche dichotomique m'a permis de monter en compétences en AC 2. Je pense avoir vraiment progressé en raisonnement algorithmique bien plus qu'en programmation au sens stricte au cours de cette saé, me permettant d'être sensibilisé à la complexité algorithmique, et de développer mon sens critique en matière de conception d'algorithmes, cela m'a appris à me poser des questions sur ce qui serait peut être plus raisonnable d'implémenter lorsque j'aurais le choix entre deux algorithmes différents et permis progresser de façon non négligeable dans la compétence "Appréhender et construire des algorithmes".

Bien que j'ai gagné en compétences grâce à cette saé, il me reste beaucoup à apprendre en termes de programmation, notamment en termes de décomposition efficace des problèmes et en raisonnement algorithmique, cette saé m'a permis de faire émerger ces deux points sur lesquels il me reste encore à progresser à l'avenir.

Annexes

Tests pour 10000 étudiants

Etudiant non existant :

```
L'étudiant JAUFFRET KURT n'a pas été trouvé !
Infos d'exécution :
-10000 étudiants
-13 tour(s) de boucle
-39 comparaisons
```

Etudiant au début :

```
L'étudiant ABADIA KURT à été trouvé à la position : 0
Il est dans le Groupe : 10
Infos d'exécution :
-10000 étudiants
-13 tours de boucle
-38 comparaisons
```

Etudiant au milieu :

```
L'étudiant JAUFFRET ANDERS à été trouvé à la position : 4999
Il est dans le Groupe : 20
Infos d'exécution :
-10000 étudiants
-1 tour(s) de boucle
-2 comparaisons
```

Etudiant à la fin :

```
L'étudiant À IMKONIS NANOU à été trouvé à la position : 9999
Il est dans le Groupe : 4
Infos d'exécution :
-10000 étudiants
-14 tour(s) de boucle
-28 comparaisons
```

Tests pour 5000 étudiants

Etudiant non existant :

```
L'étudiant JALICOT KURT n'a pas été trouvé !  
Infos d'exécution :  
-5000 étudiants  
-12 tour(s) de boucle  
-36 comparaisons
```

Etudiant au début :

```
L'étudiant ABADIA KURT a été trouvé à la position : 0  
Il est dans le Groupe : 10  
Infos d'exécution :  
-5000 étudiants  
-12 tour(s) de boucle  
-35 comparaisons
```

Étudiant au milieu :

```
L'étudiant JALICOT FILIP a été trouvé à la position : 2499  
Il est dans le Groupe : 14  
Infos d'exécution :  
-5000 étudiants  
-1 tour(s) de boucle  
-2 comparaisons
```

Etudiant à la fin :

```
L'étudiant Æ?GESEN ANNABELLE a été trouvé à la position : 4999  
Il est dans le Groupe : 18  
Infos d'exécution :  
-5000 étudiants  
-13 tour(s) de boucle  
-26 comparaisons
```

Tests pour 1000 étudiants

Etudiant non existant :

```
L'étudiant HOURTANE ANNABELLE n'a pas été trouvé !
Infos d'exécution :
-1000 étudiants
-10 tour(s) de boucle
-24 comparaisons
```

Etudiant au début :

```
L'étudiant ABDALLAH BÃ©atrice a été trouvé à la position : 0
Il est dans le Groupe : 13
Infos d'exécution :
-1000 étudiants
-9 tour(s) de boucle
-26 comparaisons
```

Etudiant au milieu :

```
L'étudiant HOURTANE Robin a été trouvé à la position : 499
Il est dans le Groupe : 3
Infos d'exécution :
-1000 étudiants
-1 tour(s) de boucle
-2 comparaisons
```

Etudiant à la fin :

```
L'étudiant ÆGESEN ANNABELLE a été trouvé à la position : 999
Il est dans le Groupe : 18
Infos d'exécution :
-1000 étudiants
-10 tour(s) de boucle
-20 comparaisons
```

Tests pour 100 étudiants

Etudiant non existant :

```
L'étudiant KURTKOWIAK ALEKSANDRA n'a pas été trouvé !
Infos d'exécution :
-100 étudiants
-7 tour(s) de boucle
-16 comparaisons
```

Etudiant au début :

```
L'étudiant AIELLO Lisa à été trouvé à la position : 0
Il est dans le Groupe : 4
Infos d'exécution :
-100 étudiants
-6 tour(s) de boucle
-17 comparaisons
```

Etudiant au milieu :

```
L'étudiant KURTKOWIAK YOANN à été trouvé à la position : 49
Il est dans le Groupe : 4
Infos d'exécution :
-100 étudiants
-1 tour(s) de boucle
-2 comparaisons
```

Étudiant à la fin :

```
L'étudiant ZIEMNIAK ALEKSANDRA à été trouvé à la position : 99
Il est dans le Groupe : 3
Infos d'exécution :
-100 étudiants
-7 tour(s) de boucle
-14 comparaisons
```

Tests pour 10 étudiants

Etudiant non existant :

```
L'étudiant PERRONNET LEO n'a pas été trouvé !
Infos d'exécution :
-10 étudiants
-4 tour(s) de boucle
-10 comparaisons
```

Etudiant au début :

```
L'étudiant FIEVEZ SUZETTE à été trouvé à la position : 0
Il est dans le Groupe : 4
Infos d'exécution :
-10 étudiants
-3 tour(s) de boucle
-8 comparaisons
```

Etudiant au milieu :

```
L'étudiant PERRONNET Richard à été trouvé à la position : 4
Il est dans le Groupe : 4
Infos d'exécution :
-10 étudiants
-1 tour(s) de boucle
-2 comparaisons
```

Etudiant à la fin :

```
L'étudiant ZARSKA LEO à été trouvé à la position : 9
Il est dans le Groupe : 4
Infos d'exécution :
-10 étudiants
-4 tour(s) de boucle
-8 comparaisons
```