INGI1122 — Projet 2016 : Smart highlight

Guillaume Maudoux, Charles Pecheur

20 avril 2016

Modalités

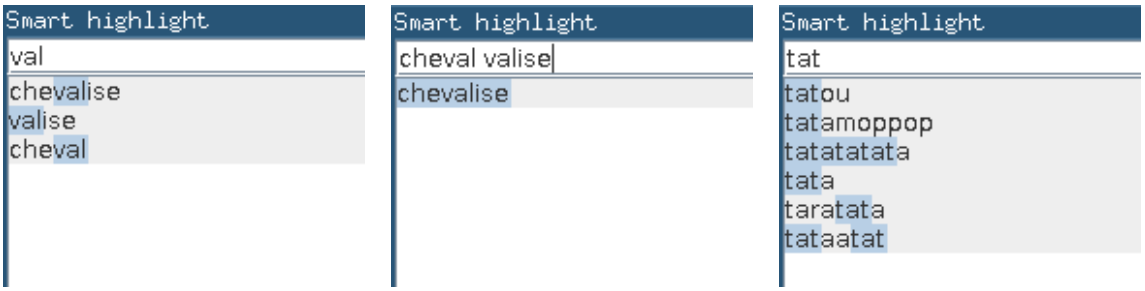
Le projet se divise en trois parties de deux semaines et s'étale donc sur six semaines. Le projet se fera en groupes fixés de trois étudiants. La répartition en groupes est disponible sur Moodle.

Chaque partie du projet se fera en deux temps : une semaine et demie pour rédiger votre solution, et trois jours pour commenter celle d'autres groupes. Il y aura donc trois échéances de remise de travaux à rendre et deux échéances de relecture croisée.

lundi 18 avril midi Remise de la partie 1 ;
jeudi 21 avril midi Relecture croisée de la partie 1 ;
lundi 2 mai midi Remise de la partie 2 ;
jeudi 5 mai midi Relecture croisée de la partie 2 ;
vendredi 13 mai midi Remise de la partie 3 ;
(pas de relecture croisée pour la partie 3).

Description du projet

Le but de ce projet est de spécifier, d'implémenter et de prouver la validité d'un programme de recherche interactif. Ce programme permettra de rechercher dans une liste de textes et d'afficher les lignes qui correspondent aux mots-clés souhaités. De plus, l'application devra surligner les parties de texte qui correspondent à ces mots-clés pour montrer à l'utilisateur pourquoi les lignes choisies correspondent. La figure 1 montre le résultat final attendu. Le code de la partie graphique de l'application vous sera fourni pour que vous puissiez vous concentrer sur les différents algorithmes.

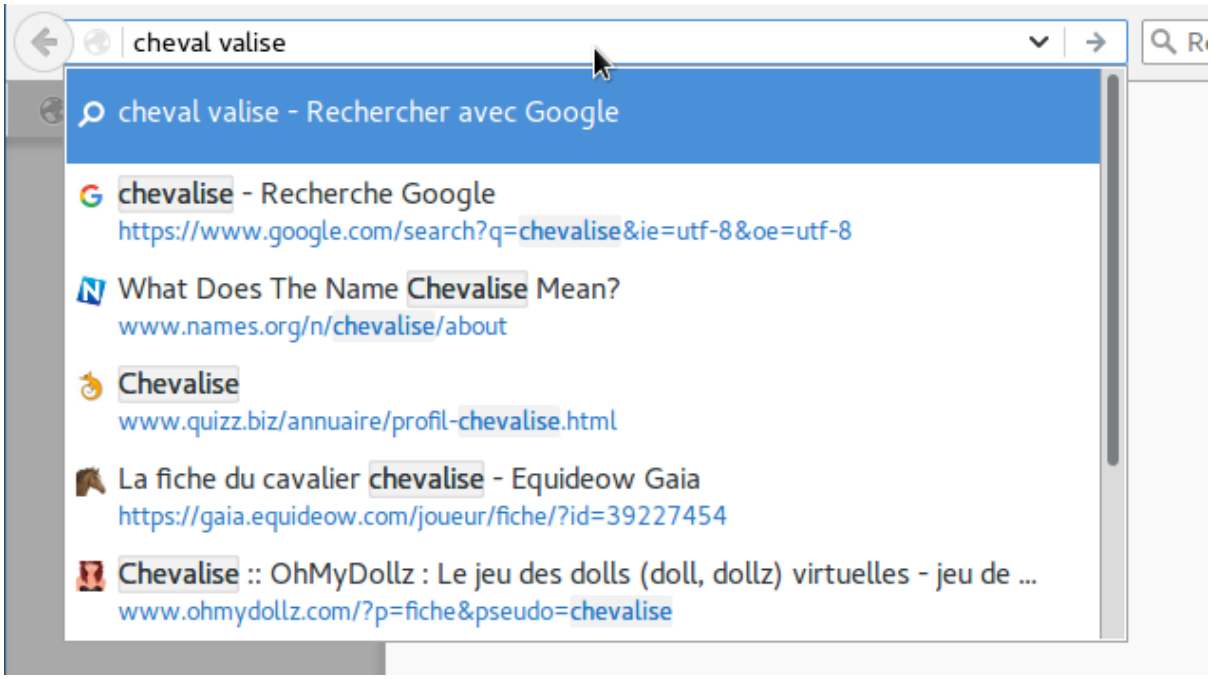


(a) Recherche simple (b) Termes superposés (c) Terme simple superposé

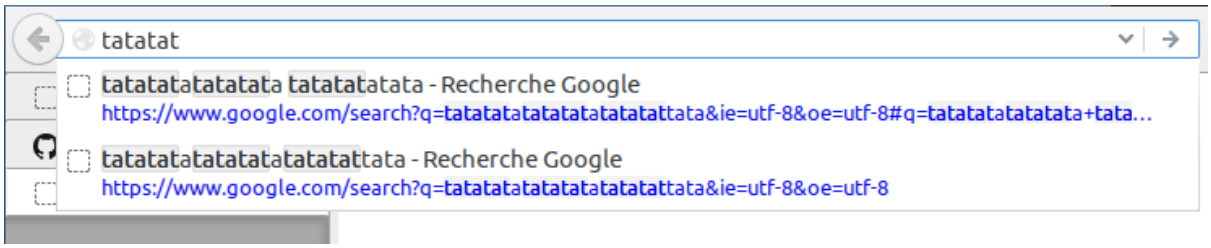
FIGURE 1 – Exemples de l'application finale

Cette application est largement inspirée du comportement de la barre d'adresse des navigateurs internet. La figure 2 montre deux captures d'écran de Firefox. Nous allons cependant un peu plus loin que ce comportement puisque nous allons surligner toutes les occurrences des mots cherchés, et pas seulement la première (comme Chrome), ou seulement les occurrences disjointes (comme Firefox, cf. figure 2b).

L'application comporte deux composantes principales : la **recherche** et le **surlignage**.



(a) Avec des mots différents, Firefox surligne toutes les parties.



(b) Si le même mot est superposé, Firefox ne souligne que les parties disjointes.

FIGURE 2 – Captures d'écran du comportement de firefox.

Recherche L'application a besoin d'une méthode pour réaliser le filtrage des textes sur base des mots-clés. Un texte correspond à un ensemble de mots-clés s'il contient chacun des mots-clés de l'ensemble. Pour simplifier, on suppose que tous mots-clés sont non-vides. En particulier, tous les textes correspondent à l'ensemble de mots-clés vide.

Surlignage L'objectif est donc de construire un algorithme capable de calculer les parties de texte à surligner en fonction des mots-clés recherchés. Ces résultats seront utilisés pour insérer les balises HTML qui modifieront l'apparence du texte pour le surligner. Comme les balises HTML doivent toujours être appariées, il faut donc calculer des portions strictement disjointes du texte. Deux portions sont strictement disjointes si il y a au moins un caractère entre chaque portion. De plus il faut que les portions ne soient pas vides, car surligner une portion vide n'a aucun sens. Enfin, il faut que les portions soient fournies dans l'ordre du texte.

En pratique L'application finale sera implémentée en Java. Nous allons nous limiter à l'usage de tableaux, d'entiers et de booléens. Ces tableaux pourront contenir des objets Java que vous avez définis. Vous ne pouvez pas utiliser de de collections Java (`java.util`) ni faire appel aux méthodes statiques de la classe `Collections` ou `Arrays`. Techniquement, le code Java que vous allez écrire ne pourra pas faire appel à des librairies tierces et ne contiendra donc pas l'instruction **`import`**. En particulier, les chaines de caractères seront représentées par des tableaux d'entiers de type **`char[]`**.

Première partie

La description ci dessus est volontairement vague. La première étape de ce projet consistera donc à établir la théorie du problème, puis à spécifier les méthodes à implémenter.

La théorie du problème Afin de comprendre et d'expliquer le problème posé, on vous demande de décrire formellement le problème posé et la tactique qui sera utilisée pour le résoudre. Il ne s'agit pas de programmer, mais bien d'analyser et décomposer le problème. Vous devez décrire :

- le problème posé en termes formels et précis ;
- la stratégie utilisée pour le résoudre, en particulier la décomposition en sous-problèmes ;
- la représentation des données intermédiaires manipulées, et les opérations nécessaires sur cette représentation ;
- les fonctions, prédicats et propriétés utiles à la résolution.

Par exemple, le problème qui consiste à donner l'étendue d'un tableau a ($\max(a) - \min(a)$) peut se décomposer comme suit :

> Pour trouver l'étendue d'un tableau donné, nous allons d'abord trouver sa valeur maximale et sa valeur minimale, puis renvoyer leur différence.
>
> Pour calculer le maximum (resp. minimum) d'un tableau, nous allons le parcourir entièrement en mémorisant à chaque étape la plus grande (resp. petite) valeur rencontrée.

N'hésitez pas à vous aider de schémas et de dessins pour représenter vos idées, et à joindre ceux-ci à votre rapport. Vous pouvez aussi vous inspirer de la théorie du problème de la solution du devoir 2.

La spécification En plus de la théorie du problème, il vous est demandé de spécifier formellement les éléments du programme à développer. Votre programme devra comporter :

- une classe `Portion` qui représente une portion à surligner dans un texte donné (en termes d'indice dans ce texte),
- une classe `PortionSet` qui représente un ensemble de portions,
- une méthode `boolean correspond(char[] texte, char[][] mots)` qui réalise la recherche de `mots` dans `texte`,
- une méthode `Portion[] quoiSurligner(char[] texte, char[][] mots)` qui réalise le calcul du surlignage de `mots` dans `texte`,

Vous devez donc donner, sous forme mathématique (pas en JML) :

- des spécifications formelles pour les méthodes `quoiSurligner` et `correspond`,
- les fonctions d'abstraction et les invariants de représentation pour les classes `Portion` et `PortionSet`,
- les spécifications formelles des opérations de `Portion` et `PortionSet`.

Remarque : une classe `Pair` qui contient deux entiers `a` et `b` s'écrit en java :

```
public class Pair { public int a ; public int b ; }
```

et correspond au type `type Pair ::= {Int a, Int b}` dans le pseudo-code utilisé dans le cours et les exercices.

Vous serez attentifs à développer les éléments demandés de manière complète, rigoureuse et claire. Pour la spécification, il est recommandé d'introduire des prédicats intermédiaires pour alléger vos pré- et post-conditions.

Délivrables

Vous devez remettre **un document pdf** sur le site Moodle du cours, dans la partie *Projet : Partie 1*, pour l'échéance reprise en tête de ce document. Veillez à inclure en couverture l'identification du cours, de l'année, du projet et les noms, prénoms et NOMA des étudiants. La relecture croisée se fera également au départ du site Moodle, dès l'échéance de la soumission.

Deuxième partie

Maintenant que vous avez posé et compris le problème à résoudre, nous allons passer à l'implémentation du programme. Concrètement, nous vous fournissons un fichier `sources.zip` contenant une interface graphique java composée d'une zone de texte et d'une liste de résultats.

En entrant des mots dans la zone de texte, vous affinez les résultats affichés dans la fenêtre. Si vous pressez la touche "entrée", la phrase affichée dans la zone de texte est ajoutée aux résultats possibles, et la zone de texte est vidée.

Cette application utilise la classe `Tools` qui contient les deux méthodes mentionnées dans la partie 1. Celles-ci sont définies sur base de l'interface Portion qui représente une portion de texte à sélectionner. Le code fourni prend pour convention de surligner le texte entre l'indice `getStart()` inclu et `getEnd()` exclu. Dans cette convention, une portion n'est non-vide que si `getStart()` est strictement inférieur à `getEnd()`. Si votre spécification utilise une autre convention, pensez à convertir vos portions au moment de renvoyer la valeur de retour de `quoiSurligner` mais ne changez pas vos spécifications.

Vous devez implémenter les méthodes manquantes de l'application en respectant au mieux les spécifications et la théorie du problème réalisés durant la phase 1. Vous ne pouvez y déroger que si elles contiennent une incohérence empêchant le bon fonctionnement de l'application ou pour tenir compte d'une remarque de la correction croisée. À l'inverse, si une imprécision vous permet plusieurs implémentations, pensez à la rapporter.

Par souci de simplicité, vous pouvez utiliser les conteneurs de listes comme `java.util.ArrayList`.

Vous devez aussi rédiger un rapport contenant

- La spécification de chacune de vos méthodes ;
- Les corrections que vous avez apportées à vos spécifications antérieures pour les rendre valides ;
- Les choix d'implémentation additionnels visant à résoudre les ambiguïtés de votre spécification ;
- La preuve formelle de la méthode et des sous-méthodes qui calculent une liste de portions disjointes, à l'exception des sous-méthodes réalisant un tri, si elles existent ;
- La preuve formelle de la méthode `quoiSurligner` et de ses sous-méthodes, à l'exception des sous-méthodes de recherche d'un mot dans un texte et du code qui traduit la représentation des portions pour l'interface graphique.

Pour les deux preuves, il est permis et même encouragé d'écrire de nombreuses petites méthodes et de décomposer le problème dans différentes classes afin de diminuer la taille et le nombre de preuves. N'oubliez pas que vous pouvez aussi introduire des prédicats logiques afin d'alléger vos spécifications.

Délivrables

Vous devez remettre une archive zip contenant votre rapport et les sources complètes de votre application (avec celles fournies) en respectant la structure suivante :

```
groupe_XX.zip
+-- Rapport.pdf
`-- src
    `-- be
        `-- ucl
            `-- info
                `-- ingi1122
                    `-- highlight
                        +-- gui
                        |   +-- FilteredListModel.java
                        |   +-- HighlightCellRenderer.java
                        |   `-- SearchWidget.java
                        `-- tools
                            +-- Portion.java
                            +-- Tools.java
                            `-- ... (vos fichiers supplémentaires)
```