# BUT_Info_2_SAE



## Dictionnaire de définitions JDict

Nous souhaitons réaliser un gestionnaire de dictionnaire développé en langage Java.
Ce projet nous permettra de créer un dictionnaire qui associe des mots à leurs définitions.
Nous aurons la possibilité d'interroger celui-ci afin d'obtenir les définitions de mots dont nous ignorons le sens.

## Wiktionary

Afin de créer notre dictionnaire, nous utiliserons comme source le projet Wiktionary de Wikimedia qui est décliné en de nombreuses langues.

La fondation Wikimedia publie régulièrement des *dumps* (fichiers sérialisés de toutes les données présentes) des Wiktionary de chacune des langues supportées. Ils sont trouvables à l'adresse https://dumps.wikimedia.org/ à laquelle on ajoute le nom du Wikitionary (enwiktionary, frwiktionary, dewiktionary, itwiktionary...).

Notre objectif est de récupérer le *dump* correspondant à la langue qui nous intéresse (fichier `xxwiktionary-latest-pages-articles.xml.bz2`) puis de l'analyser pour extraire les définitions de chaque mot qui nous intéresse.

Un *dump* est un fichier XML compressé au format bzip2. Chaque page du dictionnaire est incluse entre des balises XML `<page ...>...</page>` et concerne le mot spécifié entre les balises `<title>...</title>`. Les informations relatives au mot sont présentes entre les balises `<text>...</text>` et utilisent le formattage MediaWiki. Notons que certaines pages concernent des méta-informations qui concernent des discussions ou des utilisateurs : seules les pages dans le *namespace* 0 nous intéressent (indiquées par `<ns>0</ns>`).

Les informations sont organisées en sections hiérarchiques dont le titre est entouré d'un nombre variable de symboles =. Par exemple si nous nous intéressons au Wiktionary français, la section la plus intéressante est notée `== {{langue|fr}} ==`. En son sein on pourra trouver des sous-sections pour l'étymologie ainsi que potentiellement plusieurs sous-sections pour les différents usages du mot : nom, nom propre, pronom personnel, verbe, adjectif, locution...

L'objectif est de récupérer pour ces différents usages les définitions présentes qui débutent généralement par un caractère # (on pourra laisser de côté les exemples débutant par #*).

Prenons l'exemple du mot *plage*, dont on pourra extraire les définitions suivantes :

```
plage
    nom
        - [[étendue|Étendue]] de [[sable]] ou de [[galet]]s bordant un plan d'[[eau]].
        - {{lexique|mathématiques|fr}} Partie d'un tout constituant un sous-ensemble cohérent et contigu.
        - {{par ext}} {{lexique|informatique|fr}} [[objet|Objet]] des [[tableur]]s correspondant à un ensemble de [[cellule]]s.
        - Chacun des [[morceau]]x enregistrés sur un [[disque]] [[vinyle]] ou un [[CD]].
        - Partie plane située à l'arrière d'une [[voiture]] et couvrant la [[malle]] arrière.
        - {{lexique|informatique|fr}} {{ellipse|fr}} Groupe d'[[adresse IP|adresses IP]] consécutives.
        - {{lexique|informatique|fr}} {{ellipse|fr}} Ensemble de [[cellule]]s d'une [[feuille de calcul]] formant une zone
rectangulaire.
    verbe
        - ''Première personne du singulier de l'indicatif présent de'' [[plager]].
        - ''Troisième personne du singulier de l'indicatif présent de'' [[plager]].
        - ''Première personne du singulier du subjonctif présent de'' [[plager]].
        - ''Troisième personne du singulier du subjonctif présent de'' [[plager]].
        - ''Deuxième personne du singulier de l'impératif présent de'' [[plager]].
```

On pourra laisser le balisage *MediaWiki* indiquant certaines méta-informations ou alors des liens vers d'autres mots.

Notons qu'il existe d'autres sections pour le mot *plage* dans le Wiktionary français concernant d'autres langues que le français : par exemple, nous pouvons apprendre que *plage* est aussi la deuxième personne du singulier au présent de l'indicatif du verbe *plagen* en allemand (mais cela ne nous intéresse pas). Il n'est pas demandé non plus d'inclure les exemples et d'autres informations (tels que synonymes, antonymes. traductions...).

## Fichier des définitions extraites

Le fichier compilant toutes les définitions débute par une ligne au format JSON indiquant des méta-données concernant ce fichier dont la date de création ainsi que la langue représentée :

```
{"description": "definition file", "created_on": "20221014T145610Z", "language": "fr"}
```

Nous analysons le fichier XML du *dump* compressé et nous extrayons les définitions qui nous intéressent et que nous exprimons sous la forme d'une chaîne JSON ajouté dans le fichier de définitions. Par exemple, pour *plage*, nous pourrions écrire une ligne qui ressemblerait à ceci :

```
{"title":"plage","definitions":{"verbe":["'''Première personne du singulier de l’indicatif présent de'' [[plager]].","'''Troisième personne du singulier de l’indicatif présent de'' [[plager]].","'''Première personne du singulier du subjonctif présent de'' [[plager]].","'''Troisième personne du singulier du subjonctif présent de'' [[plager]].","'''Deuxième personne du singulier de l’impératif présent de'' [[plager]]."],"nom":["[[étendue|Étendue]] de [[sable]] ou de [[galet]]s bordant un plan d’[[eau]].","{{lexique|mathématiques|fr}} Partie d’un tout constituant un sous-ensemble cohérent et contigu.","{{par ext}} {{lexique|informatique|fr}} [[objet|Objet]] des [[tableur]]s correspondant à un ensemble de [[cellule]]s.","Chacun des [[morceau]]x enregistrés sur un [[disque]] [[vinyle]] ou un [[CD]].","Partie plane située à l’arrière d’une [[voiture]] et couvrant la [[malle]] arrière.","{{lexique|informatique|fr}} {{ellipse|fr}} Groupe d’[[adresse IP|adresses IP]] consécutives.","{{lexique|informatique|fr}} {{ellipse|fr}} Ensemble de [[cellule]]s d'une [[feuille de calcul]] formant une zone rectangulaire.","{{lexique|électronique|fr}} Zone sur un circuit dédiée aux [[contact]]s et aux autres types d’[[interconnexion]] électrique (brasure, câblage filaire, etc.)."]}}
```

Le fichier des définitions que nous constituons comprend donc (après l'en-tête des données) pour chaque mot une ligne en JSON avec les définitions extraites. Il nous faut néanmoins pouvoir trouver rapidement un mot dans ce fichier. En l'état actuel, ceci nécessiterait de lire une par une les lignes du fichier jusqu'à trouver le mot recherché ce qui serait trop coûteux étant donné que nous avons affaire à des centaines de milliers de mots. Il nous faut donc créer un fichier d'index accélerant la recherche.

## Creation de l'index

Pour créer l'index qui va accélérer nos recherches, nous associons pour chaque mot la position de la ligne JSON dans le fichier de définitions (i.e. position du 1er octet débutant la ligne et position du dernier octet la terminant). Nous trions ensuite cette table avec un ordre spécifique sur les mots :

- on prend tout d'abord en considération l'ordre lexicographique des mots normalisés dans une casse unique (en majuscules par exemple) avec suppression des diacritiques (*congres* est par exemple normalisé en *CONGRES* de même que *congrès*)
- pour deux mots de même normalisation (e.g. congres et congrès), on réalise un départage par l'ordre lexicographique lié à la comparaison des codes Unicodes (ordre naturel des String en Java) ; dans cet ordre, congres est inférieur à congrès (puisque le code de *e* est `0x0065` alors que le code de *è* est `0x00E8`)

La table étant triée, on enregistre dans le fichier index uniquement la suite des couples de position des définitions. Les positions sont exprimées sous la forme de deux entiers de 4 octets en grand-boutiste (en débutant par l'octet de poids fort). Si par exemple le fichier de définitions ne contient que les définitions de congrès de l'octet 0 à l'octet 514 et de congres de l'octet 515 à 1100, nous enregistrons d'abord l'emplacement de congres puis de congrès :

```
515 1100 0 514
```

Soit 16 octets au total que l'on peut ainsi décomposer en hexadécimal :

```
00 00 02 03 00 00 04 4C 00 00 00 00 00 00 02 02
```

Afin de reconnaître facilement un fichier d'index, on pourra faire débuter le fichier par l'en-tête "DICTINDX" (8 caractères ASCII).

## Recherche dans l'index

La recherche d'un mot donné se fait par dichotomie de la façon suivante avec l'aide de l'index :

- On récupère la ligne JSON du mot médian (en récupérant sa position dans l'index) et on compare ce mot au mot recherché
- Si le mot récupéré est égal, nous avons trouvé le mot recherché
- Si le mot récupéré est inférieur, nous récupérons le mot à la position 3/4 de l'index
- Si le mot récupéré est supérieur, nous récupérons le mot à la position 1/4 de l'index
- On continue ainsi jusqu'à trouver (ou non) le mot recherché

Afin de lire le fichier de définitions, on utilisera `RandomAccessFile` qui permet de récupérer des morceaux de fichier avec un accès aléatoire.

## Programmes demandés

Nous devons pouvoir créer un fichier de définitions et son index avec la commande suivante (on suppose que le *dump* téléchargé en langue française est `frwikitionary.xml.bz2` et que l'on enregistre le résultat dans le fichier `definitions` et `definitions.index`) :

```
cat frwikitionary.xml.bz2 | java fr.uge.DictionaryMaker fr definitions
```

On supportera l'extraction des définitions pour les Wiktionary en langue française et anglaise (ce qui suppose de bien examiner les titres de sections et sous-sections afin de ne sélectionner que celles qui sont pertinentes).

Pour rechercher une définition, nous utilisons la commande suivante (par exemple pour le mot *congres*) :

```
java fr.uge.jdict.DictionarySearcher definitions congres
```


```
{"title":"congres","definitions":{"nom":["'''Pluriel de'' [[congre#fr|congre]]."]}}
```


Cette commande retourne la lignes JSON correspondant au mot congres. Si le mot est écrit en majuscules, on suppose qu'il s'agit d'une forme normalisée et l'on retournera tous les mots correspondant à cette forme normalisée. Par exemple la commande suivante retourne les définitions à la fois de congres, congrès et Congrès (trois lignes JSON) :

```
java fr.uge.jdict.DictionarySearcher definitions CONGRES
```


```
{"title":"Congrès","definitions":{"nom":["{{lexique|diplomatie|fr}} [[assemblée|Assemblée]] de plusieurs [[représentant]]s de [[différent]]es [[puissance]]s, qui se sont [[rendre|rendus]] dans un même [[lieu]] pour y [[conclure]] la [[paix]] ou pour y [[délibérer]] sur les [[intérêt]]s généraux de divers états.","{{lexique|politique|fr}} {{France|fr}} [[réunion|Réunion]] des [[député]]s et [[sénateur]]s [[français]] en [[parlement]].","{{lexique|politique|fr}} {{États-Unis|fr}} [[corps|Corps]] [[législatif]] des [[États-Unis]] d'[[Amérique]]."]}}
{"title":"congres","definitions":{"nom":["'''Pluriel de'' [[congre#fr|congre]]."]}}
{"title":"congrès","definitions":{"nom":["[[assemblée|Assemblée]] de plusieurs [[représentant]]s de [[différent]]es [[puissance]]s, qui se sont [[rendre|rendus]] dans un même [[lieu]] pour y [[conclure]] la [[paix]] ou pour y [[délibérer]] sur les [[intérêt]]s généraux de divers états.","[[réunion|Réunion]] des [[député]]s et [[sénateur]]s [[français]] en [[parlement]].","[[corps|Corps]] [[législatif]] des [[États-Unis]] d'[[Amérique]].","Assemblée de plusieurs personnes qui se [[réunir|réunissent]] pour se [[communiquer]] les [[résultat]]s de [[leur]]s [[étude]]s et [[échanger]] [[leur]]s [[idée]]s sur des [[point]]s de [[religion]], de [[science]], de [[littérature]], de [[politique]], etc.","{{lexique|histoire|fr}} [[épreuve|Épreuve]] que devait [[subir]] un [[mari]] que son [[épouse]], [[en vue de]] [[divorce]], [[réputait]] [[impuissant]], afin de [[prouver]] la [[fausseté]] des [[allégation]]s de la [[femme]]."]}}
```


On proposera également une option permettant de retourner le résultat non pas en JSON mais en format YAML :

```
java fr.uge.jdict.DictionarySearcher definitions yaml:CONGRES
```

Nous souhaitons pouvoir extraire sur la sortie standard les formes normalisées (mots en majuscules sans diacritiques avec un mot par ligne) triées de tous les mots dont les définitions sont présentes dans le fichier :

```
java fr.uge.jdict.NormalizedExtractor definitions > words.txt
```

Si plusieurs mots ont la même forme normalisee, on ne garde qu'un seul exemplaire de celle-ci. Le fichier `words.txt` peut ensuite être utilisé pour créer un trie servant un solveur de grilles Boggle.

On pourra aussi filtrer les mots dont on retourne les formes normalisées en ajoutant les types de mots qui nous intéressent. Par exemple, si nous ne souhaitons extraire que les formes normalisées correspondant à des noms propres et des adjectifs, on pourra utiliser la commande qui suit :

```
java fr.uge.jdict.NormalizedExtractor definitions "nom propre,adjectif" > words.txt
```

On souhaite également pouvoir générer une table de fréquences de lettres pour une langue donnée. Pour cela, on examine toutes les entrées du dictionnaire et on compte le nombre d'occurrences de chaque lettre en prenant en considération le titre des entrées ainsi que les définitions. On pourra aussi considérer des multigrammes (combinaisons de lettres) pour certaines langues. Par exemple en français, on s'intéressera au digramme "qu".

```
java fr.uge.jdict.CharTableMaker definitions > chartable.txt
```

## Réduction de la taille du fichier de définitions

Cette partie est optionnelle pour les apprentis (mais obligatoire pour les initiaux). Elle consiste à réduire la taille du fichier de définitions en utilisant l'algorithme de compression de *zlib*.
A cet effet, vous pouvez transformer chaque ligne JSON de définitions en un `byte[]` compressé avant de l'ajouter au fichier de définitions. On utilisera pour cela la classe `Deflater` pour la compression et `Inflater` pour la décompression.

Nous n'activerons la compression que si le nom du fichier de définition se termine par l'extension `.z`. Nous ajoutons dans les métadonnées d'en-tête une entrée `{..., "compression": "zlib"}`. Lors de la lecture du fichier de définitions, on pourra vérifier la présence de cette information pour savoir si l'on doit ou non activer le décompresseur pour chaque entrée lue.

## Interface graphique JavaFX

Cette partie est optionnelle pour tout le monde. Elle consiste (si vous avez le temps) à réaliser une interface graphique JavaFX permettant d'interroger le dictionnaire (répliquant les fonctionnalités de `DictionarySearcher`).

## APIs et bibliothèques nécessaires

Pour mener à bien ce projet, vous aurez besoin des APIs suivantes :

- L'API XML StAX (partie intégrante du JDK depuis Java 6) qui permet de réaliser une analyse du dump XML du Wiktionary en utilisant une approche événementielle *pull* par itérateur. On obtient ainsi un flux d'événement ; pour chaque événement intéressant, nous pouvons réaliser une action qui met à jour un état. Nous pouvons ainsi extraire les balises *page* intéressantes avec leur *title*, *namespace* et *text*
- Apache Commons Compress pour gérer la décompression du dump qui utilise le format de compression bzip2 (vous aurez plus exactement besoin de la classe `BZip2CompressorInputStream`)
- Une bibliothèque gérant le format de sérialisation JSON ; il peut s'agir de :
  - org.json (bibliothèque de base)
  - Gson (bibliothèque de sérialisation JSON développée par Google assez simple à utiliser mais qui ne supporte pas encore nativement les `Record`)
  - Jackson (bibliothèque JSON la plus complète)
- La classe `RandomAccessFile` du JDK qui nous permet de lire aléatoirement dans un fichier
- Les classes `Inflater` et `Deflater` du JDK pour gérer la compression zlib des entrées du dictionnaire

Pour les APIs qui ne font pas partie du JDK, n'oubliez pas d'ajouter les dépendances nécessaires pour Maven dans votre fichier `pom.xml`.

L'utilisation des commandes indiquées dans l'énoncé présuppose la génération d'un jar lourd (fat jar) contenant toutes les dépendances. On ajoute ensuite le chemin vers ce jar dans la variable d'environnement `CLASSPATH`.

Modifié le: vendredi 14 octobre 2022, 17:03