UPMC SORBONNE UNIVERSITÉS

Département informatique

Algorithmique Avancée

Devoir de programmation: Tries

Morane Otilia GRUENPETER 3500709

06/12/2015

Table des matières

Introduction

La représentation d'un dictionnaire de mots consiste à créer des structures de données appropriées au stockage de mots sans répétition et de manière à économiser le temps de recherche ou d'ajout d'un mot à la structure. Les structures, proposées dans ce devoir, sont l'arbre de la Briandais et le Trie Hybride. Celle-ci seront présentées dans le premier chapitre avec leurs primitives . Ensuite, au deuxième et au troisième chapitres seront démontrer les fonctions avancées et les fonctions complexes pour chacune des structures en comparant leur implémentation. Ainsi, au quatrième chapitre , les calcules de complexité seront présentés. Finalement, au cinquième chapitre, le comparaison expérimentale sera apportée afin de visualiser la construction de chaque structure.

L'implémentation a été réaliser en langage objet (Java) et les fonctions sont appliquées sur les instances créées dynamiquement. Cela dit, au lieu de passer la structure en argument, la plupart des fonctions s'effectue sur l'instance courante. Mis à part les fonctions complexes, fusion et conversion. Ci dessous, le partage en dossiers du devoir.

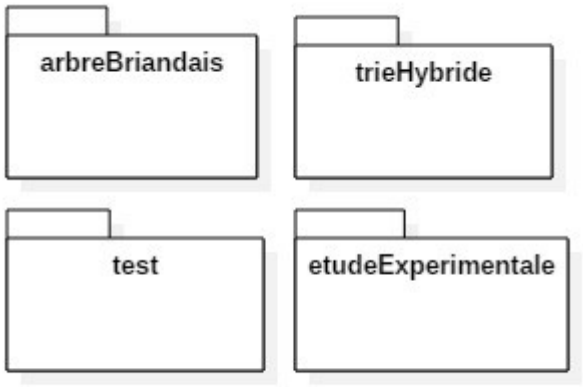


Illustration 1: le diagramme des packages

Chapitre 1 : Présentation

1.1 Structure 1: Arbres de la Briandais

1.1.1 Définition et le choix d'un caractère de fin de mot

Un arbre binaire est un arbre dans lequel chaque noeud contient un caractère, un pointeur vers un noeud frère et un pointeur vers un noeud fils. Le frère représente la lettre alternative d'un autre mot et le fils représente la lettre suivante du même mot.

Le caractère de fin de mot est * de valeur décimale dans la table de ascii 42.

1.1.2 Les primitives

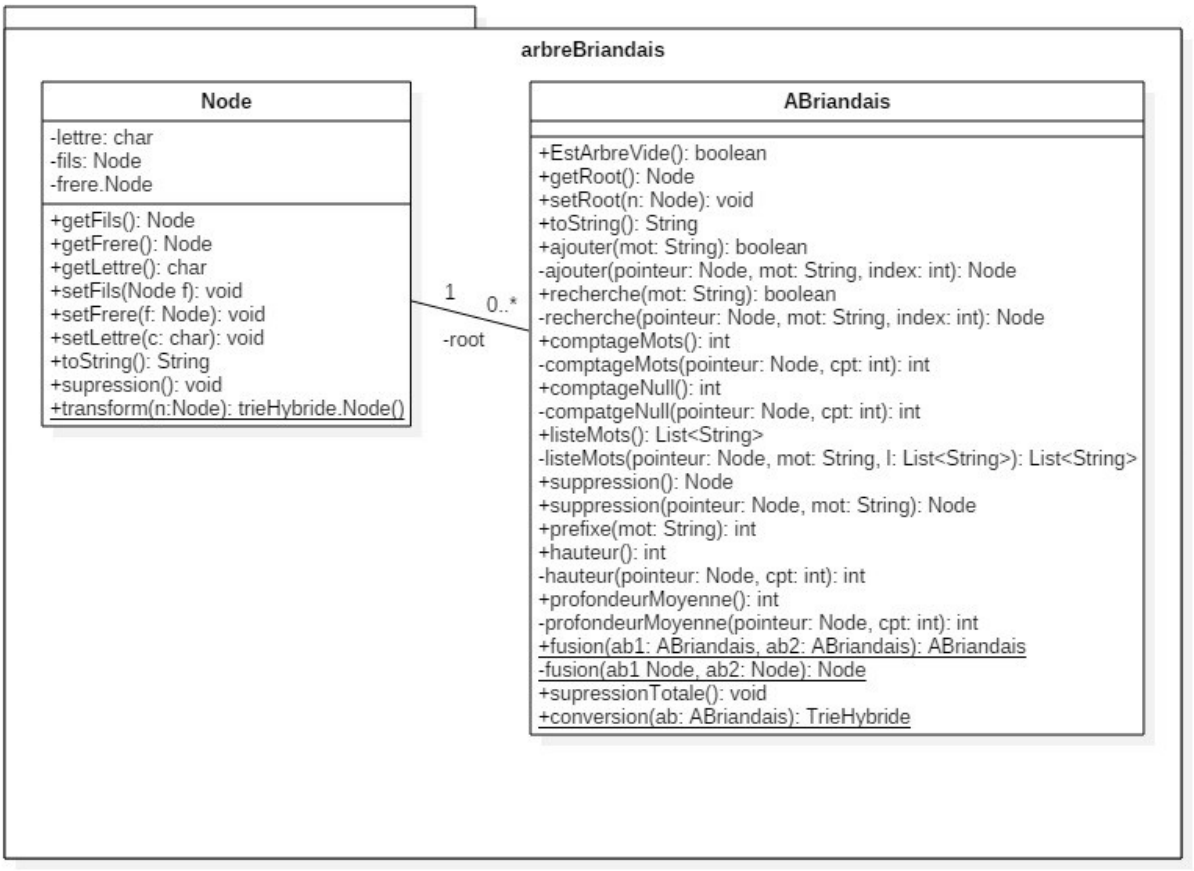


Illustration 2: diagramme de classe de la structure Arbre de la Briandais

Afin de modéliser de la structure de manière cohérente avec la programmation orienté objet, j'ai défini les deux classes suivantes. La classe Node définie les noeuds qui contiennent un caractère et deux

pointeurs.Celle-ci contient les primitives (voir illustration 2). La classe Abriandais est l'entrée au noeud racine. Celle-ci contient toutes les méthodes qui parcourent la structure.

1.1.3 l'exemple de base

Voici l'implémentation de l'exemple de base:

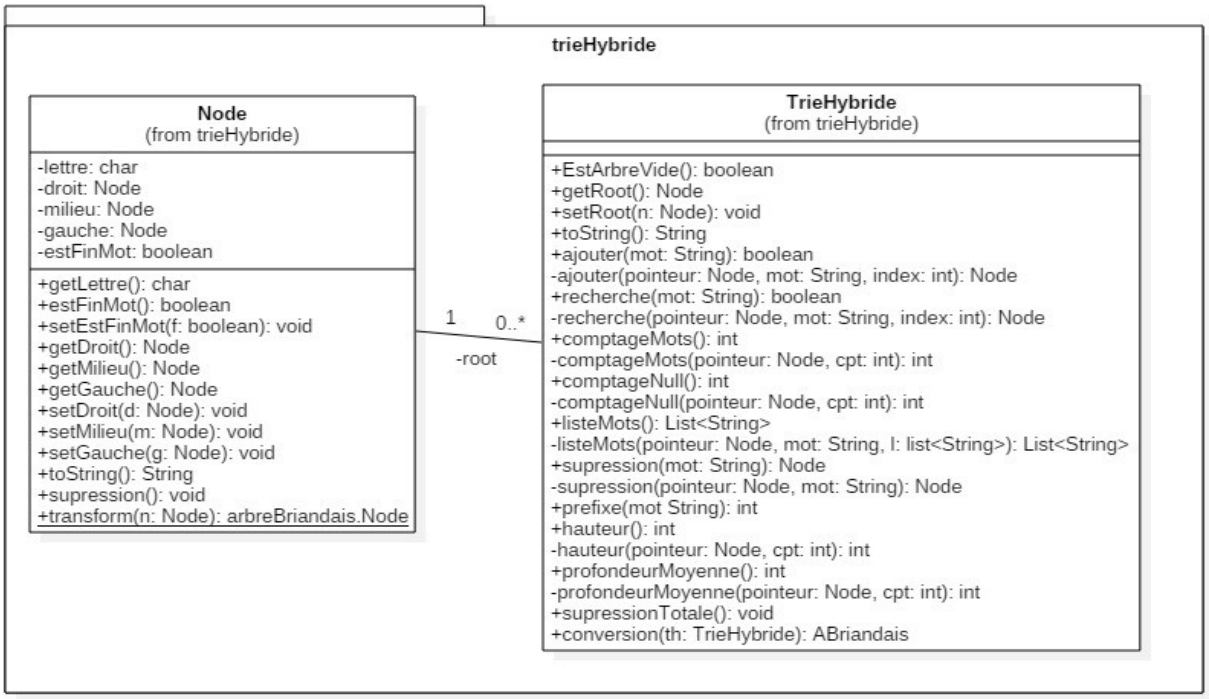
```
public void exempleDeBase(){

            ab = new arbreBriandais.ABriandais();

            System.out.println("Exemple De base: en test");

            String exemple = "A quel genial professeur de " +

                        "dactylographie sommes nous redevables de la superbe
phrase "+

                                    "ci dessous, un modele du genre, que
toute dactylo connait "+

                                    "par coeur puisque elle fait appel a
chacunne des touchesc "+

                                    "du clavier de la machine a ecrire";

            String[] decoupage = exemple.split(" ");

            for(int i= 0; i<decoupage.length; i++){

                  ab.ajouter(decoupage[i]);

            }
```

1.2 Structure 2: Tries Hybrides

1.2.1 Les primitives

Selon la modélisation choisit pour l'arbre de la Briandais, j'ai défini de la même manière la structure Trie Hybride. La classe Node définie les noeuds avec leur caractère et leurs pointeurs.Celle-ci contient les primitives (voir illustration 3). La classe Abriandais est l'entrée au noeud racine. Celle-ci contient toutes les méthodes qui parcourent la structure.

Illustration 3: diagramme de classe de la structure Trie Hybride



1.2.2 l'exemple de base

```
public void ExempleDeBase(){

            TrieHybride test;

            test = new TrieHybride();

            String exemple = "A quel genial professeur de " +

                        "dactyolgraphie sommes nous redevables de la superbe
phrase "+

                                    "ci dessous, un modele du genre, que
toute dactylo connait "+

                                    "par coeur puisque elle fait appel a
chacunne des touchesc "+

                                    "du clavier de la machine a ecrire";

            String[] decoupage = exemple.split(" ");

            for(int i= 0; i<decoupage.length; i++){

                  test.ajouter(decoupage[i]);
```

```
            }
```

Chapitre 2 : Fonctions avancées pour chacune des structures

2.1 Recherche

La recherche dans les deux structures ab pour l'Arbre de la Briandais et th pour le Trie hybride est lancée par une première fonction ab.recherche(mot) ou th.recherche(mot) respectivement. Puis, une fonction interne qui visite les nœuds récursivement.

a.Arbre de La Briandais

```
private Node recherche(Node pointeur, String mot, int index){

            if (pointeur == null) return null;

            char c = mot.charAt(index);

            if(mot.length()-1==index){

                  return pointeur;

            }else if(pointeur.getLettre()==c){

                  return recherche(pointeur.getFils(), mot, index+1);

            }else {

                  return recherche(pointeur.getFrere(), mot,index);

            }

      }
```

b.Trie Hybride

```
private Node recherche(Node pointeur, String mot, int index){

            if (pointeur == null) return null;



            if (index == mot.length()-1) {

                  return pointeur;

            }

            char c = mot.charAt(index);
```

```
            if ( c== pointeur.getLettre()){
                  return recherche(pointeur.getMilieu(),   mot, index+1);
            }else if(c < pointeur.getLettre())   {
                  return recherche(pointeur.getGauche(),  mot, index);
            }
            else  {
                  return recherche(pointeur.getDroit(), mot, index);
            }
      }
```

2.2 ComptageMots

Cette opération compte les mots dans les deux structures ab pour l'Arbre de la Briandais et th pour le Trie hybride est lancée par une première fonction ab.comptageMots() ou th.comptageMots () respectivement. Puis, une fonction interne qui visite les nœuds récursivement et augmente le compteur passé en paramètre en rencontrant la fin du mot .

a. Arbre de La Briandais

Avec une structure ab, on augmente le compteur chaque rencontre avec un noeud avec le caractère *, et on continue le parcours vers les freres, si c'est le cas.

b.Trie Hybride

Avec une structure th, on augmente le compteur si le caractère courant retourne le booléen estFinMot. Puis on continu le parcours recursivement vers les trois branches de la structure.

2.3 ComptageNil

Cette opération compte les pointeurs vide (NULL) dans les deux structures ab pour l'Arbre de la Briandais et th pour le Trie hybride. Celle-ci est lancée par une première fonction ab.comptageNull() ou th.comptageNull() respectivement. Puis, une fonction interne qui visite les nœuds récursivement et augmente le compteur passé en paramètre en rencontrant les pointeurs vide .

a. Arbre de La Briandais

Dans une structure ab, on augmente le compteur à chaque rencontre avec un pointeur vers un frêre ou un fils non instancié (null).

b.Trie Hybride

Dans une structure th, on augmente le compteur à chaque rencontre avec un pointeur vers un droit, un milieu ou un gauche non instancié (null).

2.4 Hauteur

La fonction qui calcule la hauteur de la structure. Celle-ci est lancée premièrement par une fonction ab.hauteur() ou th.hauteur(). Puis, une fonction interne qui visite les nœuds récursivement et retourne le maximum de chaque branche.

a. Arbre de La Briandais

Dans une structure ab, la hauteur de l'arbre est la longueur du plus long mot, donc on compare le maximum de chaque frêre dans la structure.

b.Trie Hybride

Dans une structure th, la hauteur de l'arbre est la plus longue branche, donc à chaque appel recursive on remonte la valeur maximale parmi les trois branches, au final cette fonction retourne la longueur de la branche la plus longue.

2.5 ProfondeurMoyenne

Cette opération permet de calculer la profondeur moyenne d'une feuille. La première fonction, ab.profondeurMoyenne() ou th.profondeurMoyenne (), récupère deux sommes, la somme du nombre de feuilles et la somme des profondeur de ces feuilles en visitant de manière récursive tous les nœuds de la structure. Puis, elle exécute le calcule de moyenne = profondeur/nombre de feuille.

2.6 ListeMots

Cette opération permet retourner la liste des mots stockés dans la structure. La première fonction, ab.listeMots() ou th.listeMots(), lance la fonction interne listeMots(pointeur,mot,liste) qui récupère, à chaque fin de mot, le mot dans la liste.

a. Arbre de La Briandais

```
      private ArrayList<String> listeMots(Node pointeur, String mot,
ArrayList<String> mots) {

            if(pointeur==null){

                  return null;

            }

            if(pointeur.getLettre()=='*'){

                  mots.add(mot);
```

```
            }
            //sans ajout de la lettre
            listeMots(pointeur.getFrere(),mot,mots);
            //avec l'ajout de la lettre en parcourant les fils
            mot+=pointeur.getLettre();
            listeMots(pointeur.getFils(),mot,mots);
            return mots;
      }
```

b.Trie Hybride

```
public      ArrayList<String>      listeMots(Node      pointeur,String      mot,
ArrayList<String> mots){
            if(pointeur==null){
                  return mots;
            }
            if(pointeur.estFinMot()){
                  //if(pointeur.getMilieu()==null){
                        mot+= pointeur.getLettre();
                  //}
                  mots.add(mot);
            }
            //sans ajout de la lettre
            listeMots(pointeur.getGauche(), mot,mots);


            String motMil=mot;
            //avec l'ajout de la lettre si ce n'était pas la fin du mot
            //car ajout double dans ce cas
            if(!pointeur.estFinMot()){
```

```
                    motMil= mot+ pointeur.getLettre();

            }

            listeMots(pointeur.getMilieu(), motMil,mots);

            listeMots(pointeur.getDroit(), mot,mots);

            return mots;

      }
```

2.7 Préfixe

Cette opération permet de compter le nombre de mots desquels le mot passé en paramètre est le préfixe. La fonction, ab.prefixe(mot) ou th.prefixe(mot), lance la recherche du préfixe qui retourne le nœud de la fin du mot s'il existe, sinon cette fonction retourne null.

a. Arbre de La Briandais

Avec un arbre de la Briandais, on utilise le noeud retourné comme point de départ de la procédure comptageMots.

b.Trie Hybride

Avec un Trie Hybride, on compte le nœud d'arrivé puis on passe au nœud milieu suivant afin de lancer la même procédure de comptageMots.

2.8 Suppression

Cette opération supprime le mot passé en paramètre s'il existe dans la structure. La fonction d'entrée est ab.suppression(mot) ou th.suppression(mot) .

a. Arbre de La Briandais

La suppression est fait recursivement dans le parcours postfixe, si le noeud à supprimer n'est pas lié à des noeuds qu'il faut sauvagrader, on supprime le noeud, sinon on crée un nouveau noeud qui lie les partie à sauvegarder.

b.Trie Hybride

On crée une Pile vide qui passe en tant que paramètre dans une fonction interne th.suppression(pointeur,mot,index,pile). Pendant le parcours on ajoute les nœuds visités à partir du moment où l'on rencontre le première lettre du mot. En retournant de ce parcours, on vérifie si le dernier nœud estFinMot, si c'est le cas, et si ces pointeurs sont vide, on supprime ce nœud et on recommence cette vérification avec tous les autres nœuds dans la pile. Sinon, on redéfini à faux le booléen estFinMot.

Chapitre 3 : Fonctions complexes

3.1 Fusionner

Ci-dessous la fusion entre deux arbre, qui fait appel à la fusion entre deux noeuds.

```
public static ABriandais fusion(ABriandais ab1, ABriandais ab2){

            ab1.setRoot(fusion(ab1.getRoot(),ab2.getRoot()));

            return ab1;

      }
```

```
private static Node fusion(Node ab1, Node ab2){

            if(ab1 ==null){

                  return ab2;

            }

            if(ab2==null){

                  return ab1;

            }

            if(ab1.getLettre()> ab2.getLettre()){

                  ab2.setFrere(fusion(ab1,ab2.getFrere()));

            }else if(ab1.getLettre()==ab2.getLettre()){

                  ab1.setFrere(fusion(ab1.getFrere(), ab2.getFrere()));

                  ab1.setFils(fusion(ab1.getFils(),ab2.getFils()));

            }else{

                  ab1.setFrere(fusion(ab1.getFrere(),ab2));

            }

            return ab1;

      }
```

3.2 Conversion

3.2.1 Conversion d'un Arbre de la Briandais vers un Trie Hybride

La conversion de l'arbre fait appel à la fonction transform défini dans les noeuds:

```
public static TrieHybride conversion(ABriandais ab){

            TrieHybride th= new TrieHybride();

            if(ab.EstArbreVide()){

                  return new TrieHybride();

            }


            //transforamtion du noeud racine en

            trie.Node tn = Node.transform(ab.getRoot());

            th.setRoot(tn);


            return th;

      }
```

```
 public static trie.Node transform(Node n){

      trie.Node tn= new trie.Node(n.getLettre());

      if(n.getFils()!=null)

            tn.setMilieu(transform(n.getFils()));

      Node frere = n.getFrere();

      if(frere!=null){

            if(frere.getLettre()<n.getLettre()){

                  tn.setGauche(transform(frere));

            }else{

                  tn.setDroit(transform(frere));

            }

      }
```

```
        return tn;

    }
```

3.2.2Conversion d'un Trie Hybride vers un Arbre de la Briandais

La fonction de conversion est statique. Celle-ci transforme chaque noeud du Trie Hybride en Noeud Arbre de la Briandais.

```
public static ABriandais conversion(TrieHybride th){

            ABriandais ab= new ABriandais();

            if(th.EstArbreVide()){

                  return ab;

            }

            arbreBriandais.Node nab= Node.transform(th.getRoot());

            ab.setRoot(nab);

            return ab;

      }
```

```
public static arbreBriandais.Node transform(Node n){

            arbreBriandais.Node abn = new arbreBriandais.Node(n.getLettre());

            if(n.getMilieu()!= null){

                  abn.setFils(transform(n.getMilieu()));

            }

            if(n.getDroit()!=null){

                  abn.setFrere(transform(n.getDroit()));

            }

            if(n.getGauche()!=null){

                  abn.setFrere(transform(n.getGauche()));

            }

            return abn;
```

```
        }
```

Chapitre 4 : Complexités

fonction	AB	TH	explication
ajouter	O(L+$\log_2 n$)	O(L+$\log_2 n$)	L est la longueur du mot, qui est ajouté au parcours d'une branche, si il faut créer tous les noeuds du mot en les ajoutant à la structure. Pour TH le parcours est plus court, en divisant par 3 la structure qu'il reste à parcourir.
primitives	O(1)	O(1)	Toutes les primitives sont en temps constant, un accès direct .
recherche	O(L+$\log_2 n$)	O(L+$\log_2 n$)	L est la longueur du mot
comptageMots	O(ln)	O(ln)	Parcours de toute la structure en comptant les mots
listeMots	O(n)	O(n)	Parcours de toute la structure en ajoutant les mots rencontrés à une liste passées en tant que paramètre
comptageNil (compatgeNull)	O(2n)	O(3n)	Parcours de toute la structure, puis à chaque visite d'une feuille, compter les pointeurs vers nil (ou null en java). Pour l'AB il s'agit de 2 appels par noeud vers frêre et vers fils. Pour le TH il s'agit de 3 appels vers droit, vers milieu et vers gauche.
hauteur	O(n)	O(n)	Parcours de toute la structure, puis retourne le max des branches parcouru
profondeurMoyenne	O(n)	O(n)	Parcours de toute la structure en comptant les feuilles et leurs profondeurs

préfixe	O(*n)*	O(*n)*	A partir d'une branche donc d'une recherche du préfixe, initialiser un comptage de mots. Au pire des cas, le préfixe est une lettre et celle-ci est la racine, donc il faudra compter les mots de toute la structure.
supression	O(L+$\log_2 n$)	O(L+$\log_2 n$)	Parcours de la structure en recherche du mot, si ce mot existe, recursivement supprmier les pointeur à ce mot dans lesquels tous les pointeurs sont null. L est la longueur du mot, dans le pire des cas il faut supprimer tout les noeuds qui appartiennent au mot.
Fusion	O(n+m)	X	Parcours des deux structures AB, avec n la taille de ab1 et m la taille de ab2 .A chaque appel recursif, comparer les deux noeuds, en ajoutant les noeuds manquant à ab1. Dans les cas il faut visiter tous les noeuds des deux structures.
conversion	De AB à TH O(n)	De TH à AB O(n)	Parcours de toutes la structure en transforament chaque noeud

Chapitre 5: Étude expérimentale

5.1. Comparaison entre Trie Hybride et Arbre de la Briandais

Dans la classe Shakespeare appartenant au dossier etudeExperimentale , la méthode main est chargé de créer plusieurs structures et de tester les temps d'exécutions sur chaque structure.

Ci dessous l'illustration 4 compare le temps de construction en ajoutant successivement tous les mots des différents fichiers de l'archive de Shakespeare dans un AB (arbre de la Briandais) et dans un TH (Trie Hybride)

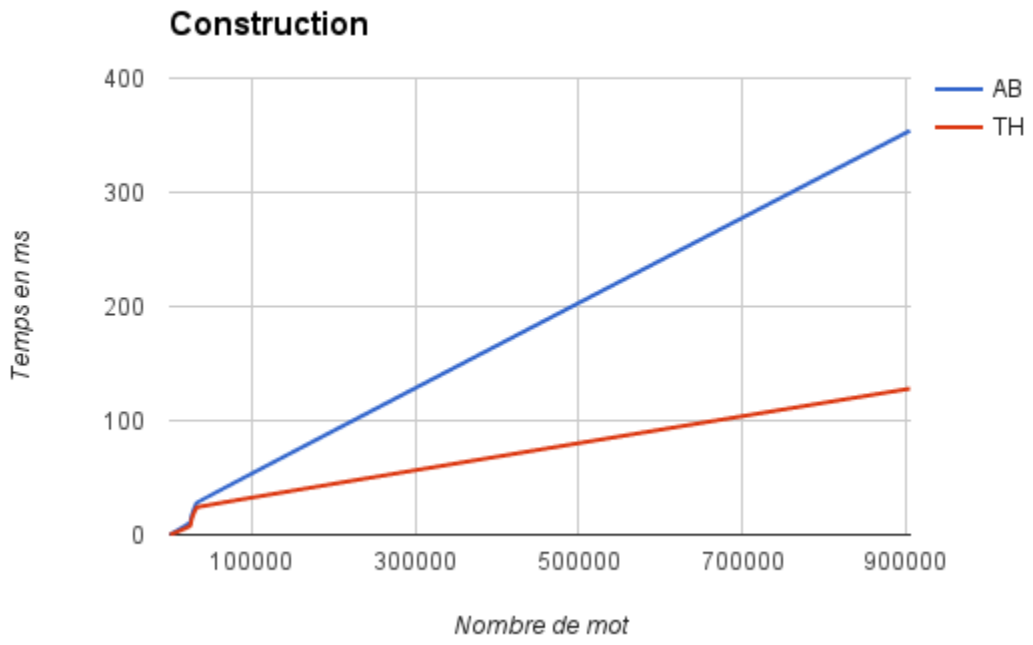


Illustration 4: construction des deux structures avec tous les mots

Conclusion

La construction de ces structures m'a permit de comprendre en profondeur leurs fonctionnement. Ainsi, j'ai pu apprécier les avantages d'utiliser des arbres et des tries pour la définition d'un dictionnaire. En commencent par les primitives et en ajoutant petit à petit des fonctions aux deux structures, j'ai pu estimer les points comparables et les divergences. La comparaison du temps de construction des structures, montre que le Trie Hybride est plus efficace que l'Arbre de la Briandais, si celui-ci est équilibré, sinon les deux sont presque identiques. Par ailleurs, l'ajout d'un nœud « fin de mo »t spécialement dans l'arbre de la Briandais est un inconvénient pour tous les algorithmes qui parcourent la structure.