Un correcteur orthographique efficace

Comment cela peut-il être mis en oeuvre ?

Paul IANNETTA

Définition (Distance de Levenshtein)

La distance de Levenshtein $\mathcal{L}$ est une application de $\mathcal{A}^* \times \mathcal{A}^* \longmapsto \mathbb{N}$ définie par :

$$\mathcal{L}(w_1, w_2) = \min(\begin{array}{ll} c + \mathcal{L}(w_1', w_2'), & \text{substitution} \\ 1 + \mathcal{L}(w_1, w_2'), & \text{ajout} \\ 1 + \mathcal{L}(w_1', w_2) & \text{suppression} \end{array})$$

Avec $w_1 = aw_1'$, $w_2 = bw_2'$ et $c = 0$ si $a = b$ ou 1 sinon.

Proposition

Complexité temporelle : $\mathcal{L}(w_1, w_2) \in \mathcal{O}(|w_1||w_2|)$

Théorème

Le couple $(\mathcal{A}^*, \mathcal{L})$ est un espace métrique.

Démonstration

- Positivité : $\mathcal{L}(w_1, w_2) \geq 0$
- Séparation : $\mathcal{L}(w_1, w_2) = 0 \Longrightarrow w_1 = w_2$
- Symétrie : $\mathcal{L}(w_1, w_2) = \mathcal{L}(w_2, w_1)$
- Inégalité triangulaire : $\mathcal{L}(w_1, w_3) \leq \mathcal{L}(w_1, w_2) + \mathcal{L}(w_2, w_3)$

Le cas d'égalité a lieu si $w_2 = w_1$ ou $w_2 = w_3$.
L'autre cas se prouve par récurrence sur la longueur de w_2 que l'on note $n = |w_2|$.
Pour $n = 0$, c'est vrai : $\mathcal{L}(w_1, w_3) \leq |w_1| + |w_3|$
Soit $n \in \mathbb{N}$ tel que l'inégalité triangulaire soit vraie.

$$\begin{aligned} \mathcal{L}(w_1, w_2) + \mathcal{L}(w_1, w_3) &= \mathcal{L}(w_1, w_2') + \mathcal{L}(w_2', w_3) \\ &\geq \mathcal{L}(w_1, w_3) \end{aligned}$$

L'inégalité triangulaire est vraie pour tout $w_2 \in \mathcal{A}^*$.

```c
#define MAX_LEN 42
#define min3(a,b,c) ((a)<(b)?((a)<(c)?(a):(c)):((b)<(c)?(b):(c)))

int
levenshtein (char * src, char * dst) {
    int dist[MAX_LEN + 1][MAX_LEN + 1] = {0};
    int i, j, len_src, len_dst;

    len_src = strlen(src);
    len_dst = strlen(dst);

    for (i = 0 ; i < MAX_LEN ; ++i)
        dist[i][0] = i;
    for (j = 0 ; j < MAX_LEN ; ++j)
        dist[0][j] = j;

    for (i = 0 ; i < len_src ; ++i)
        for (j = 0 ; j < len_dst ; ++j) {

            dist[i + 1][j + 1] = min3(
                                    dist[i][j + 1] + 1,
                                    dist[i + 1][j] + 1,
                                    dist[i][j] + (src[i] != dst[j])
                                );
        }

    return dist[len_src][len_dst];
}
```

Il s'agit de calculer la distance de Levenshtein d'un mot avec tous les autres mots du dictionnaire.

Avantage :

- Mise en oeuvre aisée et rapide

Inconvénient :

- Cette méthode est très lente en raison d'un grand nombre de calculs.

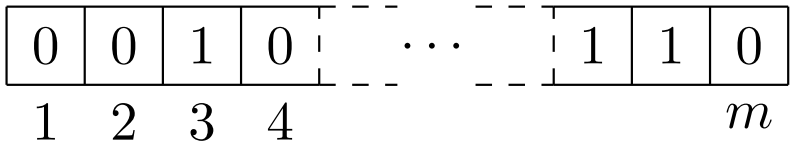
Objectifs : Limiter les accès au dictionnaire.

Définition (Filtre de Bloom)

Un filtre de Bloom est un vecteur ligne à valeurs dans $\mathbb{Z}/2\mathbb{Z}$.

Principe de construction :

- La longueur du vecteur vaut m.
- Il utilise k fonctions de hachage. ($h_i : \mathcal{A}^* \longrightarrow \mathbb{N}$)
- Il représente n éléments.



Théorème

La probabilité d'obtenir un faux positif est de l'ordre de $(1 - e^{-\frac{kn}{m}})^k$.

Démonstration

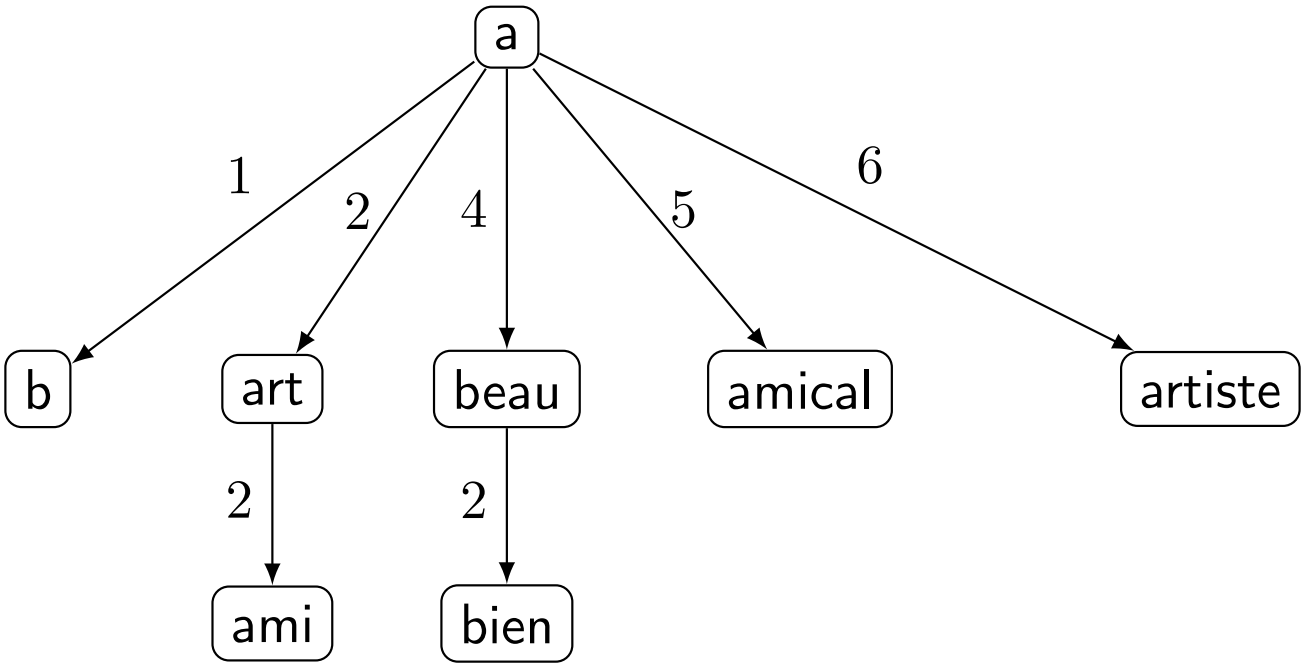
On suppose qu'une fonction de hashage donne un nombre entre 0 et m de manière équiprobable.
La probabilité qu'un bit soit à 1 est $\frac{1}{m}$ et celle qu'il soit à 0 est de $1 - \frac{1}{m}$.
Or, on utilise k fonctions de hashage que l'on applique à n éléments.
Cette probabilité devient donc $\left(1 - \frac{1}{m}\right)^{kn}$.

La probabilité d'avoir un faux positif est donc de :

$$\left(1 - \left(1 - \frac{1}{m}\right)^{kn}\right)^k = \left(1 - \left(1 - \frac{\frac{kn}{m}}{kn}\right)^{kn}\right)^k \underset{kn\infty}{\sim} \left(1 - e^{-\frac{kn}{m}}\right)^k$$

Définition (Burkhard-Keller Trees)

Les BK-Trees sont des arbres n-aires à valeurs dans un espace métrique. Ils vérifient la propriété suivante :
Tous les mots d'un sous-arbre sont à la même distance de la racine du sous-arbre au sens de la distance de Levenshtein.



`dict = {a, art, ami, amical, artiste, b, beau, bien}`.

```
1  LIST_NAME(string)
2  bk_tree_search (const char * word, int range, struct bk_tree * tree) {
3      LIST_NAME(string) res = NULL;
4      int dist, i;
5
6      if (!tree) return NULL;
7
8      dist = levenshtein(word, tree->word);
9
10     if (dist <= range)
11         LIST_CONS(char *, string, tree->word, res);
12
13     for (i = 0 ; i < NB_SONS ; ++i) {
14         int son_dist = tree->sons[i].dist;
15         if (dist - range <= son_dist && son_dist <= dist + range) {
16             LIST_NAME(string) tmp = NULL;
17
18             tmp = bk_tree_search(word, range, tree->sons[i].son);
19
20             if (!res) res = tmp;
21             else LIST_CONCAT(char *, string, res, tmp);
22         }
23     }
24
25     return res;
26 }
```

Démonstration (Preuve de correction)

Un raisonnement par récurrence montre que ce parcours en profondeur de l'arbre termine effectivement.

Montrons pour cela que les sous-arbres éludés ne contenaient pas de potentiels candidats.
On cherche les mots à une distance **inférieure ou égale à** r d'un mot w.

Soit $\mathfrak{M}(A)$ l'ensemble des mots contenus dans l'arbre A.
Soit $d = \mathcal{L}(w, A.mot)$.
Soit A_1 un sous-arbre de A tel que :

$$\forall m \in \mathfrak{M}(A_1),\ \mathcal{L}(m, A.mot) = d + r + 1.$$

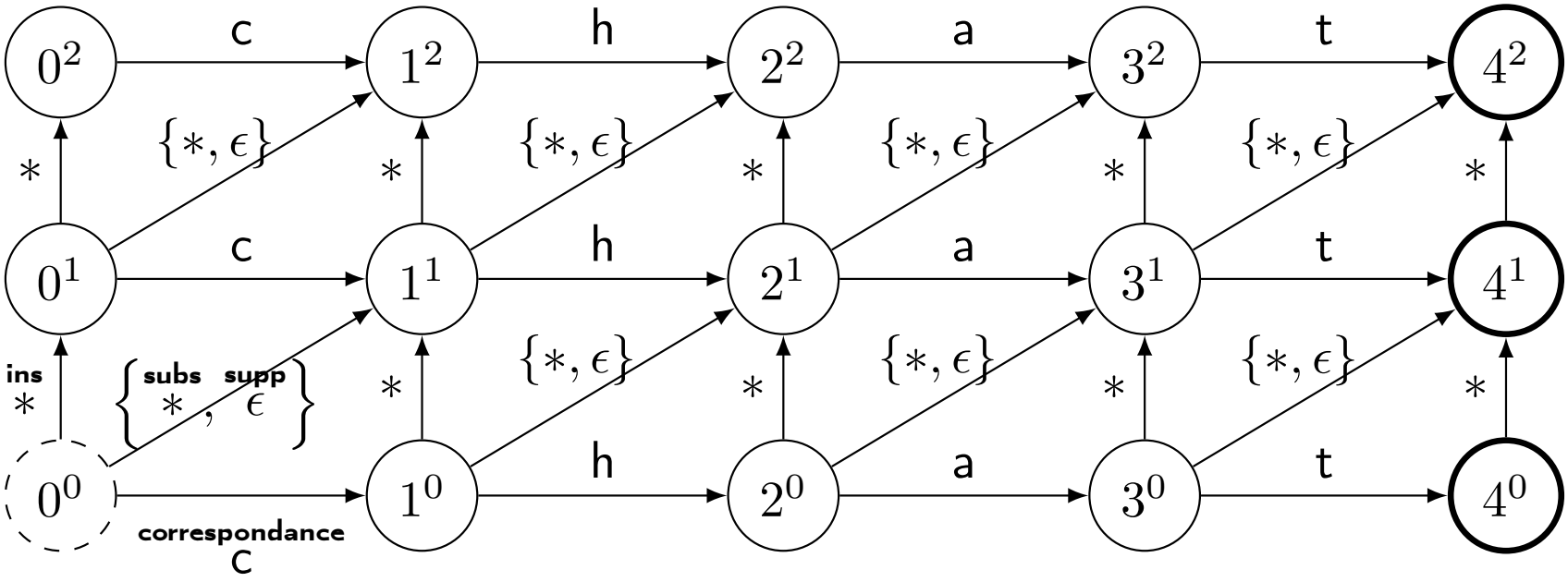
$\forall m \in \mathfrak{M}(A_1),\ d + r + 1 = \mathcal{L}(m, A.mot)$
$\forall m \in \mathfrak{M}(A_1),\ d + r + 1 \leq \mathcal{L}(m, w) + \mathcal{L}(w, A.mot)$
$\forall m \in \mathfrak{M}(A_1),\ \mathcal{L}(m, w) \geq r + 1$

Si $\mathcal{L}(m, A.mot) = d - (r + 1)$, $|\mathcal{L}(a.mot, w) - \mathcal{L}(w, m)| \leq \mathcal{L}(a.mot, m)$.

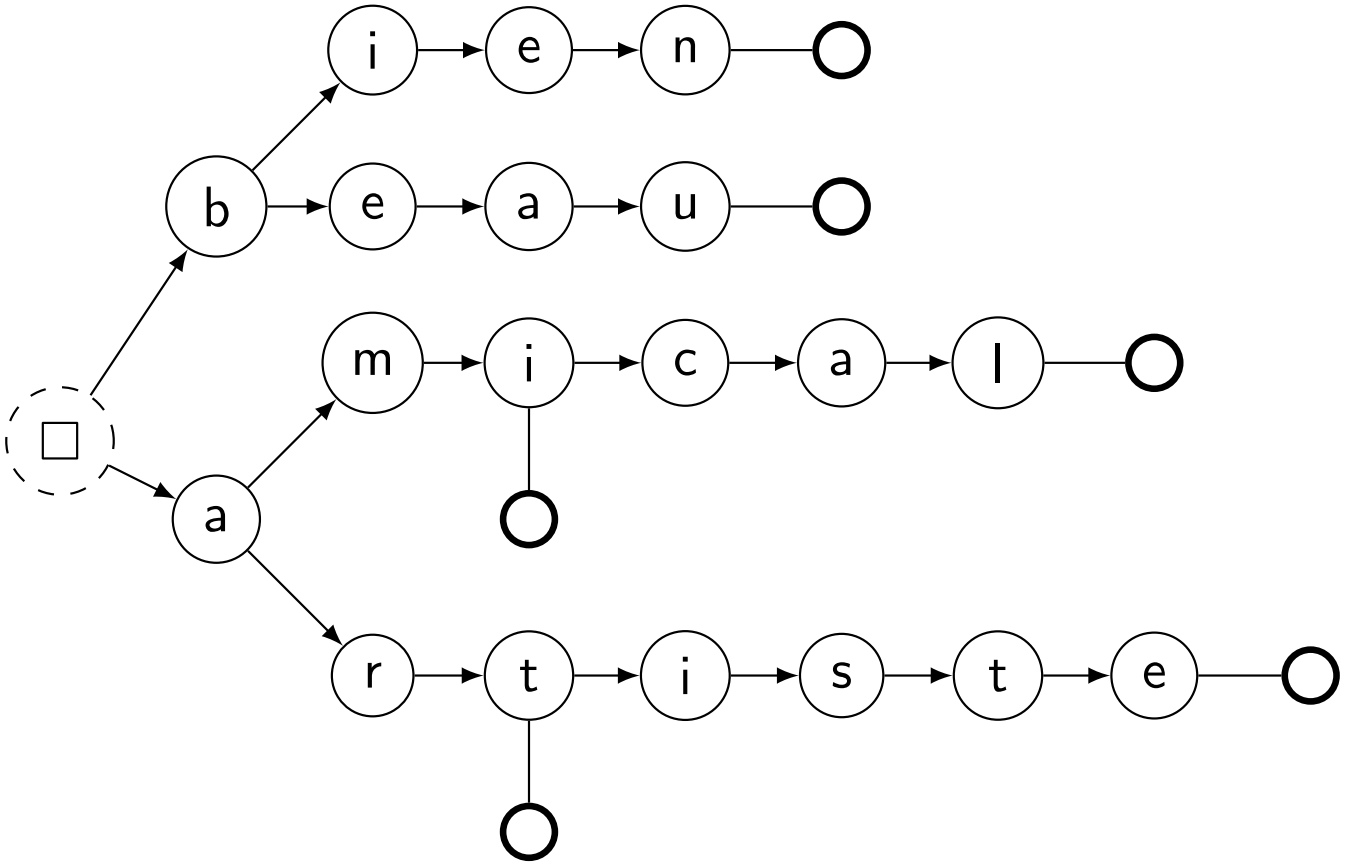
Définition (Automate de Levenshtein)

L'automate de Levenshtein d'un mot est un automate qui reconnait seulement les mots ayant une distance de Levenshtein inférieure ou égale à un entier n fixé.

$$\mathsf{Lev}_n(m) = \{w \in \mathcal{A}^* \ : \ \mathcal{L}(m, w) \leq n\}$$



dict = {a, art, ami, amical, artiste, b, beau, bien}.



Nombre de mots du dictionnaire : $324\ 476$ mots.

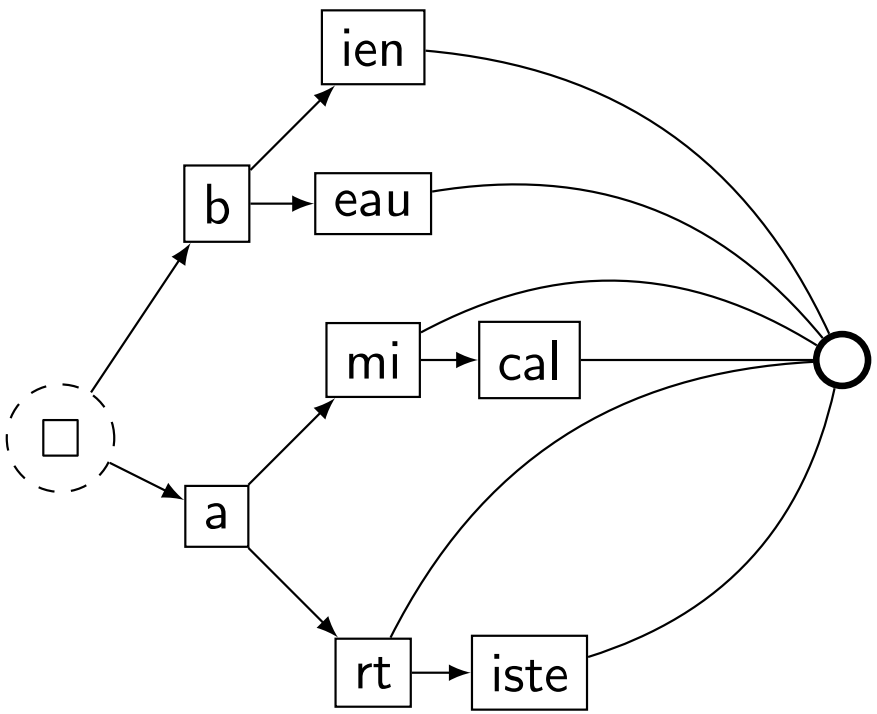
	Filtre de Bloom	BK-Tree	Naïve
Création	0 s	5 s	-
Place	(300 Ko) Paramétrable	150 Mo	-
Recherche et suggestions (mots/s)			
$\mathcal{L} = 1$	-	1000 - 10000	1
$\mathcal{L} = 2$	-	100 - 1000	”
$\mathcal{L} = 3$	-	75 - 100	”
$\mathcal{L} = 4$	-	30 - 40	”

Modifier la fonction de distance :

- Transposition.
- Lettres adjacentes sur un clavier.
- Lettres ayant des prononciations similaires.

Réduire la taille du dictionnaire pour les automates de Levenshtein :

- Utilisation d'un graphe acyclique orienté PATRICIA.



`dict = {a, art, ami, amical, artiste, b, beau, bien}`.

- Compilateurs, techniques, principes et outils. 2^{e} édition
- Wikipédia (Angaise et Française) :
  - Distance de Levenshtein.
  - Filtre de Bloom.
- Nick's Blog (blog.notdot.net) – Damn Cool Algorithms :
  - Levenshtein Automata
  - BK-Trees