Carry, un algorithme de désuffixation pour le français

M. Paternostre, P. Francq, J. Lamoral, D. Wartel et M. Saerens

Juillet 2002

Résumé

Dans le cadre du projet de recherche GALILEI (Generic Analyser and Listener for Indexed and Linguistics Entities of Information), nous avons voulu apporter un traitement linguistique au regroupement de documents électroniques. Partant du principe que des mots partageant un même radical avaient de fortes chances de partager également une unité de sens, nous avons entrepris une analyse morphologique des mots contenus dans les documents. Pour ce faire, nous avons adopté l'algorithme de désufixation de PORTER pour traiter de la langue anglaise et nous avons entrepris son adaptation simple pour le français. Une fois créé, cet algorithme fut comparé à celui de PORTER à l'aide des mesures de précision, "recall" et "RAND index". Par ces mesures, nous avons pu constater l'efficacité de notre algorithme.

1 Introduction

Depuis quelques années, les documents électroniques se multiplient aussi bien sur le réseau internet que dans les intranets d'entreprises. Retrouver l'information que l'on cherche s'avère donc de plus en plus difficile et bien souvent les utilisateurs sont découragés par la lenteur et le manque d'efficacité des moteurs de recherche traditionnels mis à leur disposition. Une des causes fréquentes des échecs de ceux-ci est l'absence presque totale d'analyse linguistique dans le traitement des requêtes et des documents. Ainsi, les documents contenant le mot "voiture" et ceux contenant le mot "automobile" ne seront pas liés entre eux alors que l'utilisateur cherchant des renseignements sur l'un sera sans aucun doute intéressé par l'autre. Un moteur de recherche verra donc son efficacité augmenter si il intègre une dimension sémantique à son traitement de l'information.

De cette même manière, les résultats qu'il obtiendra seront bien meilleurs si la morphologie de la langue est prise en compte. Il s'agira alors de mettre en évidence le rapport fort qui unit des mots comme "chanter", "chanteront", "chanteur", "chanteuses", ... Pour ce faire deux méthodes sont le plus souvent employées. D'un côté il existe la possibilité de créer un lexique comprenant toutes les formes fléchies de tous les mots. Cependant, on voit aisément qu'un tel lexique prend énormément de place en mémoire et augmente donc le temps de recherche. L'autre façon de faire, qui est aussi celle employée dans notre projet, GALILEI, est l'application de règles permettant de relier les mots entre eux sur base d'un radical commun. Ceci suppose bien entendu établie l'existence d'un lien sémantique fort entre des mots dont le radical est le même et qui diffèrent seulement dans leurs terminaisons. Il est toutefois important de noter ici que le but de l'analyseur morphologique étant la mise en relief de ce lien sémantique, le radical trouvé n'est pas nécessairement celui qui aurait été sélectionné par une analyse strictement linguistique.

Divers algorithmes de désuffixation ont été mis au point pour l'anglais, qui est une langue à morphologie relativement pauvre se prêtant donc bien à ce genre d'analyses. Les plus connus sont ceux de LOVINS [9], de PORTER [13] et de PAICE [12]. Nous avons choisi d'utiliser l'algorithme de PORTER dans nos travaux car celui-ci est assez efficace tout en étant moins lourd que celui de PAICE.

En ce qui concerne le traitement du français, nous n'avons pu trouver d'algorithmes similaires à ceux développés en anglais. Il est un fait que la langue française ayant une morphologie assez complexe, la simple suppression des suffixes ne peut suffire à regrouper des familles de mots. Le plus souvent, les outils s'intéressant à cette langue font donc appel à de lourds dictionnaires ou à un étiquetage morphosyntaxique préalable des mots présents dans les documents (NAMER [11]). Ne voulant ni utiliser de dictionnaire, ni d'analyseur syntaxique dans le cadre de notre projet, nous avons entrepris de créer nous-même un algorithme de désuffixation pour le français, que nous appellerons Carry, basé sur l'algorithme de PORTER. Les

erreurs de surracinisation ou de sous-racinisation, c'est-à -dire celles qui consistent à retrouver trop ou trop peu de racines différentes, ne sont évidemment pas négligeables avec un tel outil. Cependant, l'utilisation que nous en faisons, à savoir l'aide au regroupement de documents jugés intéressants par des utilisateurs, amoindrit leurs effets néfastes. En effet, que les mots "marmaille" et "marmite", pour reprendre l'exemple de DAL et NAMER [2], soient considérés comme appartenant à une même famille de radical "marm" n'est pas dramatique, étant donné que les chances de les voir apparaître dans des documents similaires sont assez faibles.

Dans le présent article, après une brève introduction à l'algorithme de Porter, nous proposerons l'adaptation française que nous avons mise au point. Nous présenterons ensuite les modes de validation que nous avons employés, aussi bien sur l'algorithme anglais que sur le français.

2 Les algorithmes de désuffixation

2.1 L'algorithme de Porter (anglais)

L'algorithme développé par PORTER [13] se compose d'une cinquantaine de règles de désuffixation classées en sept phases successives (traitement des pluriels et verbes à la troisième personne du singulier, traitement du passé et du progressif,...). Les mots à analyser passent par tous les stades et, dans le cas où plusieurs règles pourraient leur être appliquées, c'est toujours celle comprenant le suffixe le plus long qui est choisie. La désuffixation est accompagnée, dans la même étape, de règles de recodage. Ainsi, par exemple, "troubling" deviendra "troubl" par enlèvement du suffixe marqueur du progressif -ing et sera ensuite transformé en "trouble" par application de la règle "bl" devient "ble". Cet algorithme comprend aussi cinq règles de contexte, qui indiquent les conditions dans lesquelles un suffixe devra être supprimé. La terminaison en -ing, par exemple, ne sera enlevée que si le radical comporte au moins une voyelle. De cette manière, "troubling" deviendra "troubl", nous l'avons vu, alors que "sing" restera "sing".

2.2 L'algorithme Carry (français)

Tout comme celui de PORTER, l'algorithme que nous avons créé se déroule en diverses étapes par lesquelles les mots à traiter passent successivement [1]. Selon les règles, quand l'analyseur reconnaît un suffixe de la liste, soit il le supprime, soit il le transforme. C'est ici aussi le suffixe le plus long qui détermine la règle à appliquer.

En appendice se trouve l'ensemble des règles que nous proposons pour l'étude de la morphologie de français, dans la version 1.0 de notre algorithme. Cette version est téléchargeable gratuitement sur le site du projet GALILEI [2]. Les modifications éventuelles apparaîtront dans les versions suivantes.

Nous avons adopté les mêmes conventions que PORTER dans la première publication de son algorithme. Le lecteur intéressé est invité à en prendre connaissance dans PORTER [13]. Nous présenterons cependant ici un bref résumé de ces conventions.

Les règles se présentent comme suit : à gauche de la flèche se trouve la terminaison à modifier, à droite, le résultat obtenu après passage par l'analyseur. Le signe ε correspond à la disparition de la terminaison. Ainsi, après application de la règle "$e \rightarrow \varepsilon$", "chienne" deviendra "chienn". Ensuite, en phase trois et suivant la règle "$nn \rightarrow n$", "chienn" deviendra "chien".

Chaque règle de l'algorithme est précédée d'une condition indiquant si oui ou non la règle de désuffixation peut être appliquée. Si nous posons que C est une consonne ou une suite de consonnes, que V est une voyelle ou une suite de voyelles, et que les crochets ([]) marquent un élément optionnel, alors chaque mot du français peut être réduit à cette formule : $[C](VC)^m[V]$, où (VC) est répété un nombre m de fois. Ainsi, si la valeur contextuelle est $(m > 0)$, le radical trouvé pour un mot après application d'une règle de désuffixation doit avoir une valeur m strictement supérieure à 0. Par exemple, le mot "tissaient" ne deviendra pas "t" par application de la règle "$(m > 0)issaient \rightarrow \varepsilon$" car ce radical ne comprendrait pas au moins une séquence VC.

Encore une fois, nous invitons le lecteur intéressé par les détails à consulter PORTER [13].

1. Le nom "Carry" étant un jeu de mot avec la signification française de "PORTER".
2. `http://www.otlet-institute.org/galilei`

3 Validation

L'algorithme de PORTER étant une référence dans le domaine de la désuffixation, nous avons voulu comparer ses résultats sur des textes anglais avec ceux que nous obtenions par application de notre algorithme, Carry, sur le français. A cet effet, nous nous sommes servis des lexiques français et anglais de l'analyseur Mmorph [10], analyseur morphologique reconnu dans le monde de la linguistique computationnelle et développé à l'université de Genève.

Ces lexiques sont composés des formes fléchies de nombreux noms, verbes, adjectifs et de leurs formes de base (infinitif des verbes et masculin singulier des noms et adjectifs).

Exemple 3.1 *Entrées du lexique français de Mmorph*

"chante" = "chanter" Verb[mode=indicative\|subjunctive tense=present number=singular person=1\|3 form=surface type=1 derivation=ant]

"personnelles" = "personnel" Adjective[gender=feminine number=plural degree=positive form=surface]

"échographies" = "échographie" Noun[gender=feminine number=plural form=surface]

Vous trouverez en FIG. 1 la procédure suivie pour évaluer les algorithmes à partir des lexiques Mmorph. Les formes de base définies dans Mmorph ont tout d'abord été utilisées pour la génération des formes fléchies. Les algorithmes, Carry pour le français et PORTER pour l'anglais, ont ensuite été appliqués à ces inflexions et les radicaux ainsi trouvés furent comparés aux formes de base initiales.

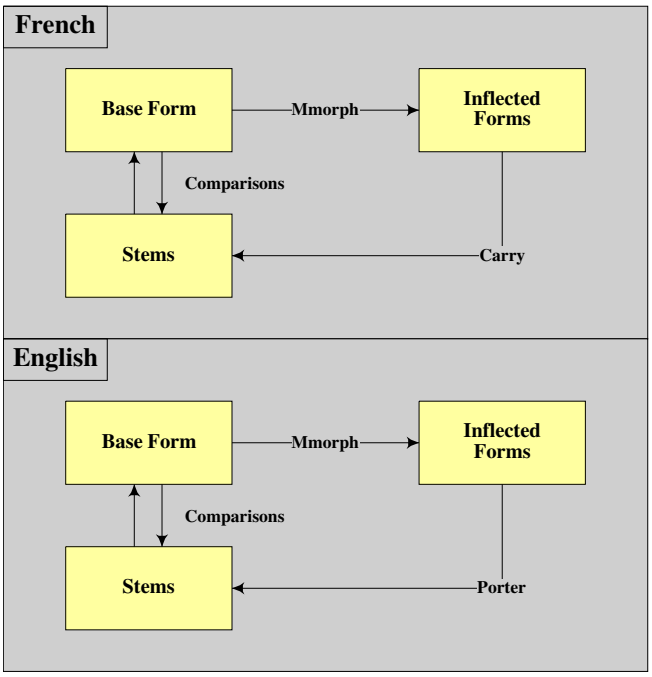

FIGURE 1 – Procédure de Validation

Nous avons appliqué l'analyseur Mmorph et notre algorithme au lexique français afin de comparer leurs performances. Le lexique français contient 225508 mots (chantera et chanteront étant comptés comme des mots différents). Ces mots sont traités par l'analyseur Mmorph, qui les groupe en 21568 formes de base (chanter pour chantera et chanteront), c'est-à-dire 9.5% du lexique. L'application de l'algorithme français sur le lexique réduit celui-ci à un nombre de 20707 formes de base, c'est-à-dire 9.1% du lexique.

Nous avons ensuite effectué la même opération sur le lexique anglais à l'aide de Mmorph et de l'algorithme de PORTER. Le lexique anglais contient 197820 mots dont Mmorph retient 98008 formes de base (49.5% du lexique) et l'algorithme de Porter 87845 (44.4% du lexique).

Nous constatons que le traitement des mots par les algorithmes de désuffixation permet de réduire considérablement la taille des dictionnaires de formes fléchies. Les deux algorithmes, PORTER et Carry, opèrent des réductions quasi identiques à celles effectuées par l'analyseur Mmorph. Les différences notées entre les résultats pour les deux langues pourraient être dues à la moindre qualité du lexique anglais par rapport au lexique français.

Cependant, ces chiffres ne peuvent nous donner d'idée précise sur la qualité des groupements de formes fléchies en formes de base. Ainsi, nous aimerions non seulement savoir (1) si toutes les formes du verbe "chanter" se retrouvent bien avec la même racine ("chantera", "chantais", "chantâmes",... donnant "chan"), mais encore (2) si n'interviennent pas dans ce groupe des flexions de mots n'ayant aucun lien sémantique avec le verbe "chanter" (nous n'aimerions, par exemple, pas retrouver dans ce groupe le terme "chanterelle"). Deux mesures ont donc été calculées afin d'évaluer les performances des algorithmes de désuffixation pour ces deux cas de figures. Dans nos calculs, nous tiendrons pour corrects les résultats obtenus par Mmorph. En d'autres termes, nous considérerons que Mmorph associe de manière adéquate les formes fléchies à leurs formes de base.

Vous trouverez ci-dessous les définitions des notions de recall et de précision (BAEZA-YATES & RIBEIRO-NETO [1]).

Recall — Mesure permettant de définir si tous les termes correspondants à une même forme de base dans Mmorph sont bien regroupés sous un même radical par les algorithmes. Pour une forme de base donnée, le recall est calculé comme suit : nombre de mots avec lesquels il partage le même radical et présents sous la même forme de base / nombre de mots avec lesquels il partage la même forme de base. Le recall est moyenné sur toutes les formes de base présentes dans Mmorph.

Precision — Mesure permettant de définir si tous les termes regroupés sous un même radical correspondent à une même forme de base dans Mmorph. Pour un radical donné, le recall est calculé comme suit : Nombre de mots avec lesquels il partage la même forme de base (définie par Mmorph) et présents sous le même radical (après désuffixation) / nombre de mots avec lesquels il partage le même radical. La précision est moyennée sur tous les radicaux découverts par les algorithmes de désuffixation.

Une troisième mesure, le RAND index (HUBERT & ARABIE [6] ou EVERITT, LANDAU & LEESE [4]), a été calculée afin d'obtenir une comparaison entre les résultats des groupements effectués par les algorithmes et ce que nous appellerons les groupements idéaux, qui correspondent aux groupements de toutes les formes fléchies en leurs formes de bases. Dans le cas présent, deux stratégies de groupements peuvent être définies : le groupement des mots partageant une même forme de base (considérée comme groupement idéal fourni par Mmorph) et le groupement des mots partageant un même radical. Cette mesure globale calcule le niveau de concordance entre chaque paire de groupement (même radical, même forme de base) de mots. Pour ce faire, elle se base sur le rapport de paires qui concordent (la forme de base correspond au radical) et le nombre total de paires.

Les résultats de nos tests sont présentés au TAB. 1. Un résultat de 1 correspond à un groupement parfait entre les formes de base et les radicaux.

	Anglais (PORTER)	Français (Carry)
Recall	0.972	0.917
Presicion	0.777	0.905
Adjusted RAND Index	0.606	0.897

TABLE 1 – Recall, Precision et Adjusted RAND Index

Les résultats obtenus par l'algorithme Carry sont moins bons que ceux de PORTER en termes de recall mais meilleurs en termes de précision. Cependant, la qualité des deux lexiques utilisés n'étant pas équivalente, nous pouvons difficilement comparer les deux algorithmes sur cette base uniquement. Si nous envisageons donc les résultats de Carry seul, nous remarquons néanmoins qu'ils sont proches de 1, c'est-à-dire que les radicaux obtenus par application de notre algorithme correspondent largement aux formes de

base définies par Mmorph. Le RAND Index nous confirme cette tendance, nous permettant de conclure que les groupement effectués par Carry et par Mmorph sont non seulement aussi nombreux, mais encore très similaires.

4 Conclusions

Le présent article visait à présenter l'algorithme de désuffixation du français réalisé par nos soins, et ce suivant le modèle de l'algorithme de PORTER pour l'anglais. Nous avons entrepris d'appliquer les deux algorithmes aux lexiques de l'analyseur morphologique Mmorph, afin de tester leur efficacité et de tenter de les comparer.

Deux conclusions peuvent être tirées des résultats des calculs effectués sur les lexiques de Mmorph. Premièrement, l'utilisation d'algorithmes de désuffixation permet de réduire considérablement la taille des dictionnaires de formes fléchies, et ce pour l'anglais, avec l'algorithme de Porter, comme pour le français, avec l'algorithme Carry. Cette diminution de taille est due au fait que les mots sont assemblés en petits groupes partageant la même forme de base ou radical.

Deuxièmement, le calcul des mesures de précision et recall nous montre que les groupes ainsi constitués correspondent relativement bien aux groupes de Mmorph. Si, comme nous l'avons déjà dit, il est délicat de tirer des conclusions à partir des résultats de l'algorithme de PORTER, nous pouvons cependant remarquer que les résultats obtenus par Carry sont très proches de ceux obtenus à l'aide de l'analyseur Mmorph.

Comme nous l'avons dit dans notre introduction, les méthodes de désuffixation n'ayant recours ni à des dictionnaires, ni à des analyseurs syntaxiques sont très peu fréquentes, voire inexistantes, pour le français. L'algorithme que nous avons réalisé répond cependant à ce besoin. En effet, tout en étant simple et léger, il permet de regrouper les mots liés sémantiquement presque aussi bien que ne le ferait un analyseur morphologique plus lourd et donc plus lent.

Références

[1] BAEZA-YATES, R. et RIBEIRO-NETO, B. (1999) : *Modern Information Retrieval*, ACM Press, New York.

[2] DAL, G. et NAMER, F. (2000) : "Génération et analyses automatiques de ressources lexicales construites utilisables en recherche d'information", *T.A.L* ,42 (2),pp. 423–446, Paris.

[3] DAL, G. (1997) : "Un point de vue sur la morphologie dérivationnelle du français", Ecole d'Eté en Morphologie (COLEX), Nantes, http ://www.mshs.univ-poitiers.fr/Forell/COLEX/ETELEX.HTM

[4] EVERITT, B.S., LANDAU, S et LEESE, M. (2001) : *Cluster Analysis*, Arnold, London.

[5] GAUSSIER, E. (2001) : "Unsupervised learning of derivational morphology from inflectional lexicons"*, Computational Linguistics*.

[6] HUBERT, L. et ARABIE, P. (1985) : *Comparing partitions*, Journal of Classification, pp 193–218.

[7] HUOT, H. (1994) : "Sur la notion de racine", *T.A.L*, 35 (2), pp. 49–76, Paris.

[8] JURAFSKY, D. et MARTIN, J.H. (2000) : *Speech and language processing : an introduction to natural language processing, computational linguistics, and speech recognition*, Prentice Hall, Inc., New Jersey.

[9] LOVINS J. B. (1968) : *Development of a Stemming Algorithm*, Mechanical Translation and Computational Linguistics, 11 (1–2), pp. 22–31.

[10] PETITPIERRE, D. et RUSSEL, G. (1994). "Mmorph - The Multext Morphology Program. Technical Report", ISSCO. http ://issco-www.unige.ch/tools/

[11] NAMER, F. (2000) : "Flemm : Un analyseur Flexionnel du Français à base de règles", *Traitement automatique des langues pour la recherche d'information*, *numéro spécial de la revue T.A.L*, Paris.

[12] PAICE, C. (1996) : "Method for evaluation of stemming algorithms based on error counting", *Journal of the American Society for Information Science*, 47 (8), pp. 632–349.

[13] PORTER, M. (1980) : "An algorithm for suffix stripping", *Program*, 14 (3), pp.130–137.

[14] ROBERT, G. (1994) : "Amélioration de recherches full-text sur base de connaissances morphologiques", Mémoire de cours postgrade en informatique : gestion moderne des documents électroniques, EPFL http ://www.issco.unige.ch/staff/robert/memoire1/

A Étapes de l'algorithme de Carry

A.1 Étape 1

$(m > 0)$ `issaient` ε
$(m > 0)$ `ellement` *el*
$(m > 0)$ `issement` ε
$(m > 0)$ `alement` *al*
$(m > 0)$ `eraient` ε
$(m > 0)$ `iraient` ε
$(m > 0)$ `eassent` ε
$(m > 0)$ `ussent` ε
$(m > 0)$ `amment` ε
$(m > 0)$ `emment` ε
$(m > 0)$ `issant` ε
$(m > 0)$ `issent` ε
$(m > 0)$ `assent` ε
$(m > 0)$ `eaient` ε
$(m > 0)$ `issait` ε
$(m > 0)$ `èrent` ε
$(m > 0)$ `erent` ε
$(m > 0)$ `irent` ε
$(m > 0)$ `erait` ε
$(m > 0)$ `irait` ε
$(m > 0)$ `iront` ε
$(m > 0)$ `eront` ε
$(m > 0)$ `ement` ε
$(m > 0)$ `aient` ε
$(m > 0)$ `îrent` ε
$(m > 0)$ `eont` ε
$(m > 0)$ `eant` ε
$(m > 0)$ `eait` ε
$(m > 0)$ `ient` ε
$(m > 0)$ `ent` ε
$(m > 0)$ `ont` ε
$(m > 0)$ `ant` ε
$(m > 0)$ `eât` ε
$(m > 0)$ `ait` ε
$(m > 0)$ `at` ε
$(m > 0)$ `ât` ε
$(m > 0)$ `it` ε
$(m > 0)$ `ît` ε
$(m > 0)$ `t` ε
$(m > 0)$ `uction` ε
$(m > 1)$ `ication` ε
$(m > 1)$ `iation` ε

$(m > 1)$ `ation` ε
$(m > 0)$ `ition` ε
$(m > 0)$ `tion` ε
$(m > 0)$ `er` ε
$(m > 0)$ `ir` ε
$(m > 0)$ `r` ε
$(m > 0)$ `eassiez` ε
$(m > 0)$ `issiez` ε
$(m > 0)$ `assiez` ε
$(m > 0)$ `ussiez` ε
$(m > 0)$ `issez` ε
$(m > 0)$ `assez` ε
$(m > 0)$ `eriez` ε
$(m > 0)$ `iriez` ε
$(m > 0)$ `erez` ε
$(m > 0)$ `irez` ε
$(m > 0)$ `iez` ε
$(m > 0)$ `ez` ε
$(m > 0)$ `erai` ε
$(m > 0)$ `irai` ε
$(m > 0)$ `eai` ε
$(m > 0)$ `ai` ε
$(m > 0)$ `i` ε
$(m > 0)$ `ira` ε
$(m > 0)$ `era` ε
$(m > 0)$ `ea` ε
$(m > 0)$ `a` ε
$(m > 0)$ `f` *v*
$(m > 0)$ `yeux` *oeil*
$(m > 0)$ `eux` ε
$(m > 0)$ `aux` *al*
$(m > 0)$ `x` ε
$(m > 0)$ `issante` ε
$(m > 0)$ `eresse` ε
$(m > 0)$ `eante` ε
$(m > 0)$ `easse` ε
$(m > 0)$ `eure` ε
$(m > 0)$ `esse` ε
$(m > 0)$ `asse` ε
$(m > 0)$ `ance` ε
$(m > 0)$ `ence` ε
$(m > 0)$ `aise` ε
$(m > 0)$ `euse` ε
$(m > 0)$ `oise` ε
$(m > 0)$ `isse` ε
$(m > 0)$ `ante` ε
$(m > 0)$ `ouse` *ou*
$(m > 0)$ `ière` ε
$(m > 0)$ `ete` ε
$(m > 0)$ `ète` ε
$(m > 0)$ `iere` ε
$(m > 0)$ `erie` ε
$(m > 0)$ `étude` ε
$(m > 0)$ `etude` ε

$(m > 0)$ `itude` ε
$(m > 0)$ `ade` ε
$(m > 0)$ `isme` ε
$(m > 0)$ `age` ε
$(m > 0)$ `trice` ε
$(m > 0)$ `cque` c
$(m > 0)$ `que` c
$(m > 0)$ `eille` eil
$(m > 0)$ `elle` ε
$(m > 0)$ `able` ε
$(m > 0)$ `iste` ε
$(m > 0)$ `ulle` ul
$(m > 0)$ `gue` g
$(m > 0)$ `ette` ε
$(m > 0)$ `nne` n
$(m > 0)$ `itée` ε
$(m > 0)$ `ité` ε
$(m > 0)$ `té` ε
$(m > 0)$ `ée` ε
$(m > 0)$ `é` ε
$(m > 0)$ `usse` ε
$(m > 0)$ `aise` ε
$(m > 0)$ `ate` ε
$(m > 0)$ `ite` ε
$(m > 0)$ `ee` ε
$(m > 0)$ `e` ε
$(m > 0)$ `issements` ε
$(m > 0)$ `issantes` ε
$(m > 1)$ `ications` ε
$(m > 0)$ `eassions` ε
$(m > 0)$ `eresses` ε
$(m > 0)$ `issions` ε
$(m > 0)$ `assions` ε
$(m > 1)$ `iations` ε
$(m > 0)$ `issants` ε
$(m > 0)$ `ussions` ε
$(m > 0)$ `ements` ε
$(m > 0)$ `eantes` ε
$(m > 0)$ `issons` ε
$(m > 0)$ `assons` ε
$(m > 0)$ `easses` ε
$(m > 0)$ `études` ε
$(m > 0)$ `etudes` ε
$(m > 0)$ `itudes` ε
$(m > 0)$ `issais` ε
$(m > 0)$ `irions` ε
$(m > 0)$ `erions` ε
$(m > 1)$ `ateurs` ε
$(m > 1)$ `ations` ε
$(m > 0)$ `usses` ε
$(m > 0)$ `tions` ε
$(m > 0)$ `ances` ε
$(m > 0)$ `entes` ε
$(m > 1)$ `teurs` ε

$(m > 0)$ eants ε
$(m > 0)$ ables ε
$(m > 0)$ irons ε
$(m > 0)$ irais ε
$(m > 0)$ ences ε
$(m > 0)$ ients ε
$(m > 0)$ ieres ε
$(m > 0)$ eures ε
$(m > 0)$ aires ε
$(m > 0)$ erons ε
$(m > 0)$ esses ε
$(m > 0)$ euses ε
$(m > 0)$ ulles *ul*
$(m > 0)$ cques ε
$(m > 0)$ elles ε
$(m > 0)$ ables ε
$(m > 0)$ istes ε
$(m > 0)$ aises ε
$(m > 0)$ asses ε
$(m > 0)$ isses ε
$(m > 0)$ oises *o*
$(m > 0)$ tions ε
$(m > 0)$ ouses *ou*
$(m > 0)$ ières ε
$(m > 0)$ eries ε
$(m > 0)$ antes ε
$(m > 0)$ ismes ε
$(m > 0)$ erais ε
$(m > 0)$ eâtes ε
$(m > 0)$ eâmes ε
$(m > 0)$ itées ε
$(m > 0)$ ettes ε
$(m > 0)$ ages ε
$(m > 0)$ eurs ε
$(m > 0)$ ents ε
$(m > 0)$ ètes ε
$(m > 0)$ etes ε
$(m > 0)$ ions ε
$(m > 0)$ ités ε
$(m > 0)$ îtés ε
$(m > 0)$ eurs ε
$(m > 0)$ iers ε
$(m > 0)$ iras ε
$(m > 0)$ eras ε
$(m > 1)$ ures ε
$(m > 0)$ ants ε
$(m > 0)$ îmes ε
$(m > 0)$ ûmes ε
$(m > 0)$ âmes ε
$(m > 0)$ ades ε
$(m > 0)$ eais ε
$(m > 0)$ eons ε
$(m > 0)$ ques *c*
$(m > 0)$ gues *g*

$(m > 0)$ nnes n
$(m > 0)$ ttes ε
$(m > 0)$ îtes ε
$(m > 0)$ tés ε
$(m > 0)$ ons ε
$(m > 0)$ ais ε
$(m > 0)$ ées ε
$(m > 0)$ ees ε
$(m > 0)$ ats ε
$(m > 0)$ eas ε
$(m > 0)$ ts ε
$(m > 0)$ rs ε
$(m > 0)$ as ε
$(m > 0)$ es ε
$(m > 0)$ fs v
$(m > 0)$ és ε
$(m > 0)$ is ε
$(m > 0)$ s ε
$(m > 0)$ ea ε
$(m > 0)$ au ε

A.2 Étape 2

$(m > 1)$ ent ε
$(m > 1)$ ation ε
$(m > 1)$ ition ε
$(m > 1)$ tion ε
$(m > 1)$ el ε
$(m > 0)$ i ε

A.3 Étape 3

$(m > 0)$ nn n
$(m > 0)$ ll l
$(m > 0)$ tt t
$(m > 0)$ y ε
$(m > 0)$ t ε
$(m > 0)$ qu c
$(m > 0)$ gu g
$(m > 0)$ issaient ε
$(m > 0)$ ellement el
$(m > 0)$ issement ε
$(m > 0)$ alement al
$(m > 0)$ eraient ε
$(m > 0)$ iraient ε
$(m > 0)$ eassent ε
$(m > 0)$ ussent ε
$(m > 0)$ amment ε
$(m > 0)$ emment ε
$(m > 0)$ issant ε
$(m > 0)$ issent ε
$(m > 0)$ assent ε
$(m > 0)$ eaient ε
$(m > 0)$ issait ε

$(m > 0)$ èrent ε
$(m > 0)$ erent ε
$(m > 0)$ irent ε
$(m > 0)$ erait ε
$(m > 0)$ irait ε
$(m > 0)$ iront ε
$(m > 0)$ eront ε
$(m > 0)$ ement ε
$(m > 0)$ aient ε
$(m > 0)$ îrent ε
$(m > 0)$ eont ε
$(m > 0)$ eant ε
$(m > 0)$ eait ε
$(m > 0)$ ient ε
$(m > 0)$ ent ε
$(m > 0)$ ont ε
$(m > 0)$ ant ε
$(m > 0)$ eât ε
$(m > 0)$ ait ε
$(m > 0)$ at ε
$(m > 0)$ ât ε
$(m > 0)$ it ε
$(m > 0)$ ît ε
$(m > 0)$ t ε
$(m > 0)$ uction ε
$(m > 1)$ ication ε
$(m > 1)$ iation ε
$(m > 1)$ ation ε
$(m > 0)$ ition ε
$(m > 0)$ tion ε
$(m > 0)$ ateur ε
$(m > 1)$ teur ε
$(m > 0)$ ier ε
$(m > 0)$ er ε
$(m > 0)$ ir ε
$(m > 0)$ r ε
$(m > 0)$ eassiez ε
$(m > 0)$ issiez ε
$(m > 0)$ assiez ε
$(m > 0)$ ussiez ε
$(m > 0)$ issez ε
$(m > 0)$ assez ε
$(m > 0)$ eriez ε
$(m > 0)$ iriez ε
$(m > 0)$ erez ε
$(m > 0)$ irez ε
$(m > 0)$ iez ε
$(m > 0)$ ez ε
$(m > 0)$ erai ε
$(m > 0)$ irai ε
$(m > 0)$ eai ε
$(m > 0)$ ai ε
$(m > 0)$ i ε
$(m > 0)$ ira ε

$(m > 0)$ `era` ε
$(m > 0)$ `ea` ε
$(m > 0)$ `a` ε
$(m > 0)$ `f` *v*
$(m > 0)$ `yeux` *oeil*
$(m > 0)$ `eux` ε
$(m > 0)$ `aux` *al*
$(m > 0)$ `x` ε
$(m > 0)$ `issante` ε
$(m > 0)$ `eresse` ε
$(m > 0)$ `eante` ε
$(m > 0)$ `easse` ε
$(m > 0)$ `eure` ε
$(m > 0)$ `esse` ε
$(m > 0)$ `asse` ε
$(m > 0)$ `ance` ε
$(m > 0)$ `ence` ε
$(m > 0)$ `aise` ε
$(m > 0)$ `oise` ε
$(m > 0)$ `isse` ε
$(m > 0)$ `ante` ε
$(m > 0)$ `ouse` *ous*
$(m > 0)$ `ière` ε
$(m > 0)$ `esse` ε
$(m > 0)$ `ete` ε
$(m > 0)$ `ète` ε
$(m > 0)$ `aire` ε
$(m > 1)$ `ure` ε
$(m > 0)$ `erie` ε
$(m > 0)$ `étude` ε
$(m > 0)$ `etude` ε
$(m > 0)$ `itude` ε
$(m > 0)$ `ade` ε
$(m > 0)$ `isme` ε
$(m > 0)$ `age` ε
$(m > 0)$ `trice` ε
$(m > 0)$ `cque` *c*
$(m > 0)$ `que` *c*
$(m > 0)$ `eille` *eil*
$(m > 0)$ `elle` ε
$(m > 0)$ `able` ε
$(m > 0)$ `iste` ε
$(m > 0)$ `ulle` *ul*
$(m > 0)$ `gue` *g*
$(m > 0)$ `ette` ε
$(m > 0)$ `nne` *n*
$(m > 0)$ `itée` ε
$(m > 0)$ `ité` ε
$(m > 0)$ `té` ε
$(m > 0)$ `ée` ε
$(m > 0)$ `é` ε
$(m > 0)$ `usse` ε
$(m > 0)$ `aise` ε
$(m > 0)$ `ate` ε

$(m > 0)$ `ite` ε
$(m > 0)$ `ee` ε
$(m > 0)$ `e` ε
$(m > 0)$ `issements` ε
$(m > 0)$ `issantes` ε
$(m > 1)$ `ications` ε
$(m > 0)$ `eassions` ε
$(m > 0)$ `eresses` ε
$(m > 0)$ `issions` ε
$(m > 0)$ `assions` ε
$(m > 1)$ `iations` ε
$(m > 0)$ `issants` ε
$(m > 0)$ `ussions` ε
$(m > 0)$ `ements` ε
$(m > 0)$ `eantes` ε
$(m > 0)$ `issons` ε
$(m > 0)$ `assons` ε
$(m > 0)$ `easses` ε
$(m > 0)$ `études` ε
$(m > 0)$ `etudes` ε
$(m > 0)$ `itudes` ε
$(m > 0)$ `issais` ε
$(m > 0)$ `trices` ε
$(m > 0)$ `eilles` *eil*
$(m > 0)$ `irions` ε
$(m > 0)$ `erions` ε
$(m > 1)$ `ateurs` ε
$(m > 1)$ `ations` ε
$(m > 0)$ `usses` ε
$(m > 0)$ `tions` ε
$(m > 0)$ `ances` ε
$(m > 0)$ `entes` ε
$(m > 1)$ `teurs` ε
$(m > 0)$ `eants` ε
$(m > 0)$ `ables` ε
$(m > 0)$ `irons` ε
$(m > 0)$ `irais` ε
$(m > 0)$ `ences` ε
$(m > 0)$ `ients` ε
$(m > 0)$ `ieres` ε
$(m > 0)$ `eures` ε
$(m > 0)$ `aires` ε
$(m > 0)$ `erons` ε
$(m > 0)$ `esses` ε
$(m > 0)$ `euses` ε
$(m > 0)$ `ulles` *ul*
$(m > 0)$ `cques` *c*
$(m > 0)$ `elles` ε
$(m > 0)$ `ables` ε
$(m > 0)$ `istes` ε
$(m > 0)$ `aises` ε
$(m > 0)$ `asses` ε
$(m > 0)$ `isses` ε
$(m > 0)$ `oises` *o*

$(m > 0)$ `tions` ε
$(m > 0)$ `ouses` *ou*
$(m > 0)$ `ières` ε
$(m > 0)$ `eries` ε
$(m > 0)$ `antes` ε
$(m > 0)$ `ismes` ε
$(m > 0)$ `erais` ε
$(m > 0)$ `eâtes` ε
$(m > 0)$ `eâmes` ε
$(m > 0)$ `itées` ε
$(m > 0)$ `ettes` ε
$(m > 0)$ `ages` ε
$(m > 0)$ `eurs` ε
$(m > 0)$ `ents` ε
$(m > 0)$ `ètes` ε
$(m > 0)$ `etes` ε
$(m > 0)$ `ions` ε
$(m > 0)$ `ités` ε
$(m > 0)$ `ites` ε
$(m > 0)$ `ates` ε
$(m > 0)$ `îtes` ε
$(m > 0)$ `eurs` ε
$(m > 0)$ `iers` ε
$(m > 0)$ `iras` ε
$(m > 0)$ `eras` ε
$(m > 1)$ `ures` ε
$(m > 0)$ `ants` ε
$(m > 0)$ `îmes` ε
$(m > 0)$ `ûmes` ε
$(m > 0)$ `âmes` ε
$(m > 0)$ `ades` ε
$(m > 0)$ `eais` ε
$(m > 0)$ `eons` ε
$(m > 0)$ `ques` *c*
$(m > 0)$ `gues` *g*
$(m > 0)$ `nnes` *n*
$(m > 0)$ `ttes` ε
$(m > 0)$ `îtes` ε
$(m > 0)$ `tés` ε
$(m > 0)$ `ons` ε
$(m > 0)$ `ais` ε
$(m > 0)$ `ées` ε
$(m > 0)$ `ees` ε
$(m > 0)$ `ats` ε
$(m > 0)$ `eas` ε
$(m > 0)$ `ts` ε
$(m > 0)$ `rs` ε
$(m > 0)$ `as` ε
$(m > 0)$ `es` ε
$(m > 0)$ `fs` *v*
$(m > 0)$ `és` ε
$(m > 0)$ `is` ε
$(m > 0)$ `s` ε
$(m > 0)$ `eau` ε

$(m > 0)$ `au` ε
$(m > 1)$ `ent` ε
$(m > 1)$ `ation` ε
$(m > 1)$ `ition` ε
$(m > 1)$ `tion` ε
$(m > 1)$ `el` ε
$(m > 0)$ `i` ε
$(m > 0)$ `nn` n
$(m > 0)$ `ll` l
$(m > 0)$ `tt` t
$(m > 0)$ `y` ε
$(m > 0)$ `t` ε
$(m > 0)$ `qu` c
$(m > 0)$ `gu` g