Projet de Programmation en Langage C

Webscraping et Structuration de Données
Word Embedding

Auteurs : Romain CARRIÈRE & Sofiane BADDOU

Filière : 1A IR
École : Télécom Physique Strasbourg

Février 2026

Table des matières

PARTIE 1 : Extraction et Structuration de Données HTML

PARTIE 2 : Word Embedding

PARTIE 1

Extraction et Structuration de Données HTML

1. Introduction

Cette première partie du projet porte sur l'extraction et la structuration de données à partir de pages HTML. L'objectif est de développer un programme en langage C capable d'analyser du code HTML, d'extraire le contenu informatif et de le restructurer dans différents formats exploitables (texte brut, CSV et HTML simplifié).

Le programme traite des pages web complexes en filtrant les éléments non informatifs (scripts, styles, balises de mise en forme) pour ne conserver que le contenu pertinent. Cette extraction constitue une étape préalable essentielle pour la phase 2 du projet (word embedding).

2. Architecture du programme

2.1. Organisation des fichiers

Le projet est structuré selon l'organisation suivante :

Fichier	Rôle
html-read.c	Programme principal contenant le *main()* et la gestion des fichiers
fichier.c et fichier.h	Module d'analyse et de parsing du code HTML
csv-to-html.c et csv-to-html.h	Module de conversion CSV vers HTML
mise_en_page.c et mise_en_page.h	Module de formatage et amélioration de la lisibilité
Makefile	Automatisation de la compilation et tests
test.txt	Fichier HTML source (page TPS de l'école depuis Git)
test1.txt	Texte extrait non structuré (mode texte)
test1.csv	Structure tabulaire intermédiaire (mode structuré)
test1.html	Fichier HTML restructuré en sortie
test.exe	Exécutable final

2.2. Compilation et exécution

```
# Compilation avec Makefile
make test.exe

# Test avec valgrind (détection fuites mémoire)
make valgrind
```

Un Makefile gère les dépendances entre modules, compile les fichiers sources en fichiers objets (.o), puis génère l'exécutable. Cette approche permet une compilation incrémentale où seuls les fichiers modifiés sont recompiles.

3. Modes de fonctionnement

Le fichier html-read.c constitue le point d'entrée du programme. Il orchestre l'ensemble du processus d'extraction et de structuration des données. Son fonctionnement repose sur une variable de contrôle txt_or_csv qui détermine le mode de sortie :

Valeur	Comportement
txt_or_csv = 0	Génération d'un fichier .txt contenant uniquement le texte extrait
txt_or_csv = 1	Génération d'un fichier .csv (structure tabulaire) et d'un fichier .html restructuré

Le module principal gère l'ensemble des opérations d'entrée/sortie : ouverture (*fopen*), lecture (*fread*), écriture et fermeture (*fclose*) des fichiers.

4. Modules fonctionnels

4.1. Module de parsing HTML : fichier.c

Ce module constitue le cœur du système d'extraction. Il implémente les algorithmes nécessaires pour analyser la structure HTML et extraire le contenu pertinent grâce à une structure de pile (stack) permettant de gérer l'imbrication des balises.

4.1.1 Gestion de la pile

L'analyse repose sur une **structure de pile** gérant l'imbrication des balises :

- **creer_pile()** : Initialise une pile vide
- **enfile()** : Ajoute une balise ouvrante au sommet
- **defile()** : Retire la balise lors d'une balise fermante
- **free_pile()** : Libère la mémoire allouée

Cette approche par pile permet de :

- Vérifier la cohérence de la structure HTML (correspondance ouvrante/fermante)
- Déterminer le contexte de chaque élément de contenu
- Filtrer efficacement les balises non informatives (scripts, styles)

4.1.2.Fonction principales

- **html_copy(char* s)** : Charge le contenu HTML source depuis le dépôt Git et le copie dans un buffer. C'est à ce niveau qu'il est possible d'effectuer des prétraitements sur le code HTML.
- **balise(char* s, int i)** : Identifie le type de balise correspondant au caractère < situé à l'indice i.
- **b_a_suppr(char* s, size_t N)** : Parseur déterminant quelles balises doivent voir leur contenu supprimé (retourne une structure balises_t).
- **f_wo_in_b(char* s, balises_t balises)** : Supprime le contenu des balises identifiées par b_a_suppr().
- **suppr_bal(char* s)** : Supprime les balises HTML, les \t, les \n et tout le contenu après la balise </main>. Génère une chaîne de texte brut.
- **suppr_bal_csv(char* s)** : Même fonction que suppr_bal() mais formate la sortie pour correspondre à la structure CSV (première ligne = types de balises, deuxième ligne = contenus).
- **copy(FILE* f, FILE* g)** : Copie le contenu du fichier f dans le fichier g.

4.2. Module de mise en page : mise_en_page.c

Ce module améliore la lisibilité du texte extrait en appliquant des transformations sur les chaînes de caractères.

4.2.1. Fonctions utilitaires

- **next_8_are_space(char* s, size_t i)** : Détermine si 8 caractères consécutifs à partir de l'indice i sont des espaces.

- **quote(char* s, size_t i)** : Détecte la présence de l'entité HTML " à partir de l'indice i.

4.2.2. Fonction principale

- **espace_a_entree(char* s)** : Analyse les espaces consécutifs dans le texte extrait. Lorsqu'un nombre important d'espaces successifs est détecté (seuil défini par next_8_are_space), ceux-ci sont remplacés par des retours à la ligne (\n), améliorant la structure visuelle.

Cette fonction gère également la conversion des entités HTML courantes, notamment " → ". Cette normalisation évite les incohérences dans le texte final.

4.3. Module de restructuration : csv-to-html.c

Ce Ce module reconstruit un fichier HTML simplifié à partir du format CSV intermédiaire.

4.3.1. Structure du fichier CSV

Le fichier CSV généré suit une structure spécifique :

- **Première ligne** : Types des balises HTML dans l'ordre d'apparition (séparés par ,)
- **Deuxième ligne** : Contenus correspondants dans la même colonne

Exemple :

```
h1,p,h2,p
Titre principal,Paragraphe introductif,Sous-titre,Contenu détaillé
```

4.3.2. Fonctions de parsing CSV

- **int_to_tile(char* s, int k)** : Retourne l'indice du premier caractère de la k-ième case (cellule) dans la chaîne CSV.
- **nb_tile(char* s)** : Détermine le nombre de cellules par ligne.
- **gettile(char* s, size_t tile, char* buf)** : Extrait le contenu de la cellule commençant à l'indice tile dans la première ligne (type de balise) et le stocke dans buf.
- **gettile_content(char* s, size_t tile, char* buf)** : Extrait le contenu de la cellule dans la deuxième ligne (contenu de la balise) et le stocke dans buf.

4.3.3. Fonctions de génération HTML

- **make_balise(char* s1, char* s0, int k, int end_b, int nxl)** : Inscrit dans s0 à partir de l'indice k une balise de type s1. Les paramètres end_b et nxl indiquent respectivement s'il s'agit d'une balise fermante et si un retour à la ligne doit suivre.
- **make_cont(char* s1, char* s0, int k)** : Inscrit le contenu s1 dans s0 à partir de l'indice k.
- **csv_to_html(char* s)** : Fonction principale qui reconstruit la chaîne HTML complète à partir de la chaîne CSV. Elle parcourt le CSV, extrait types et contenus, puis génère les balises correspondantes avec leur contenu.

5. Chaîne de traitement complète

Le processus global d'extraction et de structuration se déroule selon les étapes suivantes :

1) Le fichier HTML brut (test.txt) est chargé en mémoire via la fonction html_copy()

2) Le code HTML est analysé balise par balise. Les balises informatives (h1, h2, p, article,section) sont identifiées et leur contenu est extrait. Les balises non pertinentes (script, style, meta) et tout le contenu après </main> sont ignorés.

3) Le contenu textuel est extrait en tenant compte du contexte (type de balise, niveau d'imbrication). La pile maintient la cohérence de cette extraction.

4) Le texte extrait est normalisé :

- Conversion des entités HTML (" → ")
- Gestion des espaces multiples → retours à la ligne
- Suppression des caractères de formatage (\t, \n superflus)

5) Selon le mode sélectionné, le programme génère soit un fichier texte brut (.txt), soit un fichier CSV structuré suivi d'un fichier HTML restructure

Mode texte (txt_or_csv = 0) :

- Génération de test1.txt : texte brut débarrassé de toutes les balises, paragraphes correctement séparés.

Mode structuré (txt_or_csv = 1) :

- Génération de test1.csv : structure tabulaire (balises/contenus) ligne
- Génération de test1.html : HTML restructuré et épuré conservant la sémantique du document original

6. Difficultés rencontrées et solutions

6.1. Compréhension de la structure HTML

Problème : La principale difficulté initiale a été de comprendre en profondeur la structure d'un document HTML. Les pages web actuelles contiennent de nombreux éléments non informatifs (scripts JavaScript, feuilles de style, balises de mise en page) qu'il faut identifier et filtrer.

Solution adoptée : Après analyse de plusieurs pages HTML, une taxonomie des balises a été établie. Les balises de contenu (h1-h6, p, article, section) sont systématiquement traitées, tandis que les balises techniques (script, style, meta) sont ignorées. Tout le contenu après la balise </main> est également supprimé.

6.2. Confusion entre slash et antislash

Problème : Un problème qui a persisté pendant plusieurs séances concernait la confusion entre le slash (/) et le backslash (\) dans les balises fermantes HTML. Cette confusion, bien que simple, a été à l'origine de nombreux bugs difficiles à diagnostiquer :

- Mauvaise détection des balises fermantes → pile déséquilibrée
- Extraction incorrecte du contenu (mélange entre sections)
- Erreurs de segmentation lors du parcours de la chaîne HTML
- Incohérences dans le fichier de sortie (balises mal fermées)

6.3. Gestion de la mémoire

Problème : Le travail avec des chaînes de caractères en C nécessite une gestion rigoureuse de la mémoire : fuites mémoire lors de l'allocation dynamique, dépassements de buffer lors de la concaténation de chaînes, et pointeurs invalides après libération. Il est arrivé un moment où le nombre d'erreur a dépassé 10 millions. Un message inattendu de la part de valgrind, notant le nombre immense d'erreurs et la fausseté du programme, est alors apparue.

7. Résultats et validation

Le programme a été testé sur la page HTML de l'école (section TPS) fournie dans le dépôt Git. Les tests ont permis de valider le bon fonctionnement de l'ensemble de la chaîne de traitement.

7.1. Format de sortie texte

En mode texte (txt_or_csv = 0), le programme extrait uniquement le contenu informatif sans conserver la structure HTML. Le résultat (test1.txt) est un fichier texte brut, débarrassé de toutes les balises, où les paragraphes sont correctement séparés et les espaces superflus éliminés.

7.2. Format de sortie structuré

En mode structuré (txt_or_csv = 1), deux fichiers sont générés :

- **test1.csv** : Fichier intermédiaire contenant la structure tabulaire des données extraites. Chaque élément HTML est représenté par une paire balise/contenu, facilitant les traitements ultérieurs.
- **test1.html** : Fichier HTML restructuré conservant la sémantique du document original tout en éliminant le bruit (scripts, styles inline, attributs non essentiels). Il offre une version épurée et plus lisible du document source.

8. Conclusion Partie 1

Cette première partie du projet a permis de développer un outil fonctionnel d'extraction et de structuration de données HTML en langage C. Le programme répond aux objectifs fixés en offrant une solution modulaire, robuste et extensible.

L'architecture modulaire adoptée facilite la maintenance et l'évolution du code. Chaque module a une responsabilité clairement définie et peut être testé indépendamment. L'utilisation d'un Makefile assure une compilation efficace et reproductible.

Les données extraites par ce programme constituent une base solide pour la phase 2 du projet (word embedding). Le format texte produit est directement exploitable pour la génération de vecteurs one-hot encoding.

PARTIE 2

Word Embedding

9. Objectifs de la phase 2

La phase 2 vise à réaliser le word embedding sur les pages HTML du site de l'école. Le programme transforme le texte extrait en représentations vectorielles exploitables par des modèles NLP. La méthode retenue est le One-Hot Encoding : chaque mot du vocabulaire est représenté par un vecteur de dimension N (taille du vocabulaire) avec un 1 à sa position et des 0 ailleurs.

Justification : Approche simple et fondamentale, idéale pour une implémentation en C et la compréhension des concepts de vectorisation.

10. Architecture du programme

10.1. Pipeline de traitement

Le programme we.c implémente un pipeline séquentiel en 4 étapes :

HTML → Extraction texte → Normalisation → Filtrage → Vocabulaire → Export CSV

1. Extraction : Suppression des balises HTML (fichier → 01_texte_brut.txt)

2. Normalisation : Minuscules, UTF-8, suppression ponctuation (→ 02_texte_normalise.txt)

3. Filtrage : Suppression des déterminants français (→ 03_texte_filtre.txt)

4. Export : Génération du vocabulaire et vecteurs One-Hot (→ vocabulaire_embedding.csv)

10.2. Organisation modulaire

Module	Fonctions principales
Traitement UTF-8	traiter_caractere_special()
Vocabulaire	write(), structure motVocab
Prétraitement	decouper_et_filtrer(), normaliser_et_filtrer(), nettoyer_mot()
Export	exporter_vocabulaire_csv()
Principal	main(), est_dans_balise_html()

11. Stratégies et algorithmes adoptés

11.1. Extraction du contenu HTML

Algorithme : Machine à états avec flag in_tag.

Parcours caractère par caractère du fichier HTML. Lorsqu'un '<' est détecté, le flag est activé et le contenu est ignoré jusqu'au '>'.

11.2. Traitement des caractères UTF-8

Stratégie : Décodage manuel basé sur l'analyse du premier octet

- 1 octet (< 0x80) : ASCII standard → conversion minuscule
- 2 octets (0xC0-0xDF) : Caractères accentués → normalisation ASCII (é → e, à → a, ç → c)
- 3 octets (0xE0-0xEF) : Symboles spéciaux (€ → "euro", guillemets « » → suppression)

Justification : Uniformisation du vocabulaire et réduction de sa taille.

11.3. Construction du vocabulaire

On utilise une structure `motVocab` qui est composée de :

- ➔ Un mot (`char mot[MAX_LONGUEUR_MOT]`)
- ➔ Un identifiant unique (`int id`)
- ➔ Un nombre d'occurrence (`int frequence`)

Algorithme d'insertion :

1. Recherche linéaire dans le vocabulaire existant.
2. Si trouvé : on incrémente la fréquence,
3. Sinon : ajout en fin de tableau.

Choix : Tableau statique pour simplicité de gestion mémoire.

11.4. Filtrage linguistique

Déterminants supprimés : l, d, le, la, de, du, un, les, des, une.

Implémentation : Comparaisons de chaînes dans *write()* avant insertion.

11.5. Normalisation du texte

Transformations appliquées dans *normaliser_et_filtrer()* :

- Ponctuation supprimée : , . ; ! ? : ( ) / \
- Apostrophes et tirets → espaces
- Conversions UTF-8 → ASCII
- Majuscules → minuscules

11.6. Génération des vecteurs One-Hot

Algorithme : Pour chaque mot i, création d'un vecteur de dimension N (le nombre de mots uniques) :

vecteur[i][j] = 1 si i == j, 0 sinon.

Format CSV généré :

```
id;mot;frequence;v0;v1;v2;...;vN
0;telecom;38;1;0;0;...;0
1;ingenieur;29;0;1;0;...;0
```

12. Description des fonctions

➔ `int` **`est_dans_balise_html(`**`char c, int* in_tag`**`)`** : Détecte si un caractère fait partie d'une balise HTML. Gère l'état de lecture pour ignorer le contenu entre < et >.

➔ `void` **`traiter_caractere_special(`**`FILE *dest,FILE *source, unsigned char premier_octet`**`)`** : Normalise un caractère UTF-8 en ASCII. Convertit les caractères accentués en leur équivalent ASCII (ô → o).

➔ `void` **`nettoyer_mot(`**`char* mot`**`)`** : Supprime les espaces et retours à la ligne en début/fin de mot.

➔ `int` **`write(`**`char mot[],FILE *file`**`)`** : Ajoute un mot au vocabulaire ou incrémente sa fréquence sauf si le mot est un determinant, auquel cas il est ignoré.

➔ `int` **`decouper_et_filtrer(`**`char *fichier_entree,char *fichier_sortie`**`)` : Découpe le texte mot à mot et envoie chaque mot à *write()*.

➔ `int` **`normaliser_et_filtrer(`**`char *fichier_source, char *fichier_tampon,char *fichier_destination`**`)` : Normalise le texte (minuscules, ponctuation, UTF-8). Supprime les caractères spéciaux.

➔ `void` **`exporter_vocabulaire_csv(`**`const char* fichier_csv`**`)` : Exporte le vocabulaire et les vecteurs One-Hot en CSV. Génère un fichier contenant l'ID, le mot, sa fréquence et son vecteur.

13. Paramètres d'exécution

13.1. Compilation et utilisation

```
# Compilation
gcc we.c -o we

# Exécution
./we
```

Note : Le programme va directement chercher le fichier test1.html déposé en partie 1, il n'y a donc rien à passer en paramètre.

13.2. Fichiers d'entrée/sortie

Fichier	Type	Description
../Part1/test1.html	Entrée	Page HTML source
data/01_texte_brut.txt	Intermédiaire	Texte sans balises HTML
data/02_texte_normalise.txt	Intermédiaire	Texte normalisé
data/03_texte_filtre.txt	Intermédiaire	Texte sans déterminants
output/vocabulaire_embedding.csv	Sortie finale	Vocabulaire + vecteurs

13.3. Constantes configurables

#define MAX_LONGUEUR_MOTS 50
#define MAX_MOTS 10000
#define FICHIER_HTML "../Part1/test1.html"
#define FICHIER_TEXTE_BRUT "data/01_texte_brut.txt"
#define FICHIER_TEXTE_NORMALISE "data/02_texte_normalise.txt"
#define FICHIER_TEXTE_FILTRE "data/03_texte_filtre.txt"
#define FICHIER_SORTIE_CSV "output/vocabulaire_embedding.csv"

14. Résultats obtenus

14.1. Performances sur fichier de test

Résultats obtenus sur le fichier *https__www.telecom-physique.fr_formation_presentation-des-diplomes_.html* :

- Taille du fichier HTML : 85 Ko
- Vocabulaire généré : 304 mots uniques
- Temps d'exécution : 3,220 ms (moyenne sur 24 essais)
- Taille du CSV : 186 Ko

14.2. Statistiques du vocabulaire

- 75,66% des mots : 1 occurrence (hapax)
- 20,72% des mots : 2-5 occurrences
- 3,32% des mots : > 5 occurrences
- Mot le plus fréquent : "et" avec 32 occurrences

14.3. Exemple de transformation

```
Entrée HTML : <p>L'École Télécom Physique forme des ingénieurs.</p>
              ↓
Extraction : L'École Télécom Physique forme des ingénieurs.
              ↓
Normalisation : l ecole telecom physique forme des ingenieurs
              ↓
Filtrage : ecole telecom physique forme ingenieurs
              ↓
Vocabulaire : {ecole, telecom, physique, forme, ingenieurs}
              ↓
Vecteurs : ecole → [1,0,0,0,0]
           telecom → [0,1,0,0,0]
           …
```

15. Conclusion générale

15.1. Objectifs atteints

- ✓ Extraction complète et structuration de données HTML
- ✓ Normalisation (UTF-8, ponctuation, conversion minuscules)
- ✓ Construction d'un vocabulaire avec fréquences
- ✓ Génération des vecteurs One-Hot Encoding
- ✓ Export au format CSV exploitable pour modèles NLP
- ✓ Architecture modulaire facilitant la maintenance

15.2. Améliorations possibles

- Gestion dynamique du vocabulaire (malloc/realloc) pour lever la limite des 10 000 mots
- Format d'export plus compact (JSON, binaire, format sparse)
- Filtrage étendu (prépositions, conjonctions, adverbes)
- Test sur un modèle de language pour verifier sa comprehension du word embedding implémenté
- Utilisation d’un word embedding dense (deux mot de sens proches auront un vecteur similaire)

16. Bilan du projet

Ce projet a permis de développer une chaîne complète de traitement de données web en langage C, allant de l'extraction HTML brute jusqu'à la génération de représentations vectorielles exploitables en NLP. L'architecture modulaire adoptée favorise la réutilisabilité et l'évolution du code. Les performances obtenues (traitement en moins de 0.004 seconde) démontrent l'efficacité de l'implémentation. Ce travail constitue une base solide pour des applications plus avancées de traitement du langage naturel et d'analyse de contenu web.