Algorithmique du texte. Exemples et Applications.

I. Recherche de motif [CRO 1.3]

Motivation 1 La recherche de motif est le moteur d'actions quotidiennes, comme le fameux CTRL+F pour rechercher dans una page web, ou le post-traitement de sortie en ligne de commande avec `grep`.

Définition 2 **Un alphabet Σ** est un ensemble fini non-vide de **lettres**.

Exemple 3 Par exemple, $\{0,1\}$ pour l'alphabet binaire, $\{A,C,G,T\}$ pour le séquençage ADN, ou l'ensemble de symboles Unicode sont des alphabets.

Définition 4 **Un mot, texte ou motif** (selon le contexte) est une suite finie $u_0...u_{n-1} \in \Sigma^n$.

Exemple 5 Dans le texte « abracadabrant », le motif « bra » apparaît aux positions 1 et 8.

A. Recherche naïve

Intuition 6 Tester pour chaque facteur du texte w s'il est égal à p, c'est-à-dire pour chaque $i \in [\![0, |w| - |p|]\!]$, $w_i...w_{i+|p|-1} = p$.

Algorithme 7 **Approche naïve**

```
tw, tp = len(w), len(p)
for i in range(0, tw - tp):
    for j in range(i, i + tp - 1):
      if w[j] != p[j - i]:
        break
    else:
      print(p, "apparaît en position", i)
```

Complexité 8 Cet algorithme effectue au pire cas $|p|$ $(|w| - |p| - 1)$ comparaisons de symboles, d'où sa complexité en $\mathcal{O}(|w| \times |p|)$.

B. Améliorations de la recherche naïve

Intuition 9 La complexité élevée provient principalement de la comparaison répétée des même symboles du texte. Nous aimerions une méthode permettant de glisser la fenêtre de comparaison à la position suivante en temps constant : cela est possible en utilisant une fonction de hashage bien choisie !

Algorithme 10 **L'algorithme de Rabin-Karp** utilise une fonction de hachage *incrémentale*. Il est alors possible de calculer $\text{hash}(w_{i+1}...w_{i+|p|})$ à partir de $\text{hash}(w_i...w_{i+|p|-1})$.

```
let hash_fact ancien_hash fact fact' =
  ancien_hash - hash fact.[0] + hash fact'.[String.length fact'
- 1]
```

Si les hash sont égaux, on doit tout de même comparer le facteur de w à p afin de détecter les collisions de notre fonction de hachage.

Complexité 11 Dans le meilleur cas, notre fonction de hachage n'a pas de collisions. On ne compare alors jamais les facteurs à notre motif inutilement, cela prendra un temps linéaire en $\mathcal{O}(|w|)$.
Dans le pire cas, il est nécessaire d'effectuer les mêmes comparaisons que l'algorithme naïf. On retombe ainsi sur la complexité $\mathcal{O}(|w| \times |p|)$ en pire cas.

Pratique 12 En pratique, il est rare que des collisions se produisent, ce qui justifie l'utilisation de cet algorithme.

Algorithme 13 **L'algorithme de Boyer-Moore** utilise un pré-traitement sur le motif afin de faire un tableau de saut, évitant les comparaisons répétées des mêmes symboles du texte. Cet algorithme a une complexité $\mathcal{O}(|w| + |p|)$.

C. Méthodes basées sur les automates

Motivation 14 Il est souhaitable d'étendre la recherche de motifs aux **motifs rationnels**, c'est-à-dire rechercher les facteurs d'un texte w qui sont dans $\mathcal{L}(e)$, avec e une expression régulière.

Application 15 `grep` est un utilitaire en ligne de commande qui permet de faire de la recherche de motif rationnels.

Définition 16 **Les expressions régulières** sont définies inductivement comme :

$$e \triangleq \varepsilon \mid a \in \Sigma \mid (e + e) \mid (ee) \mid e^*$$

Théorème 17 Kleene On peut construire un automate fini qui reconnaît $\mathcal{L}(e)$ pour toute expression régulière e.

DEV

Algorithme 18 L'algorithme regexp2dfa (ou Glushkov déterministe) construit l'automate fini déterministe reconnaissant $\mathcal{L}(e)$ pour toute expression régulière e en complexité $\mathcal{O}\left(2^{|e|}\right)$, puis peut l'utiliser pour chercher des motifs en complexité linéaire en la taille du texte.

II. Mesure de similarité

A. Notion de distance [CRO 7.1]

Définition 19 La distance de Hamming est une mesure du nombre de symboles deux-à-deux différents entre deux mots de même longueur.

Exemple 20 La distance de Hamming entre « agregation » et « agergation » est de 2.

Application 21 Les codes correcteurs est la principale utilisation de la distance de Hamming.

Définition 22 La distance de Levenshtein est une *distance d'édition* mesurant le coût minimal (en terme d'opérations élémentaires **substitution**, **insertion** et **suppression**) entre deux chaînes.
Chacune de ces opérations ont un coût associé, notés respectivement $\mathrm{Sub}(a,b)$, $\mathrm{Ins}(a)$ et $\mathrm{Del}(a)$.

Remarque 23 La distance de Levenshtein permet de comparer des chaînes de longueurs différentes.

Exemple 24 Les mots « Dijkstra » et « Djikstre » ont une distance de Levenshtein de 3 pour un coût unitaire sur chacune des opérations.

$$\text{Dijkstra} \xrightarrow{\mathrm{Del}(j)} \text{Dikstra} \xrightarrow{\mathrm{Ins}(j)} \text{Djikstra} \xrightarrow{\mathrm{Sub}(a,e)} \text{Dijkstre}$$

Il reste à montrer qu'une distance de deux est impossible.

Remarque 25 D'autres distances d'édition existent, comme la distance de Damerau-Levenshtein qui supporte également la *permutation* comme opération élémentaires. « Dijkstra » et « Dijkstre » auraient alors une distance de 2.

B. Problème de l'alignement [CRO 7.2]

Définition 26 Le problème de l'alignement optimal d'une distance est le suivant : Étant donné deux mots u et v, quelles opérations élémentaires doivent être réalisées pour « transformer » u en v et vice-versa avec un coût minimal pour cette distance ?

Exemple 27 Un alignement optimal pour la distance de Levenschtein entre les mots « ACGA » et « ATGCTA » est:

$$ACG**A$$
$$ATGCTA$$

On observe deux insertions et une substitution, ce qui correspond à une distance de Levenshtein de 3.

DEV

Algorithme 28 L'algorithme de Needleman–Wunsch utilise la programmation dynamique pour calculer un alignement optimal.

Algorithme 29 L'algorithme d'Hirschberg [GUS 12.1.2] permet de calculer un alignement optimal entre deux mots en combinant programmation dynamique et diviser-pour-régner.

Application 30 L'alignement est un problème important en bioinformatique pour le séquençage ADN, ou pour la correction orthographique automatique.

DEV

Application 31 [CRO 8.2] **La recherche de motif approximative** consiste à trouver des facteurs v d'un mot w ayant une distance d'édition d'au plus k à un motif p.

Algorithme 32 L'algorithme de Sellers , basé sur la programmation dynamique, peut résoudre ce problème en $\mathcal{O}(|w|.|p|)$.

Programme 33 `fzf` est un utilitaire ligne de commande qui permet de faire de la recherche de motif approximative.

Pratique 34 En pratique, on peut parfois se contenter d'une **approximation** de cet alignement optimal. Des outils comme BLAST permettent d'aligner des séquences d'ADN efficacement au moyen d'heuristiques, en sacrifiant la garantie d'optimalité.

III. Compression

Motivation 35 Pour minimiser la taille d'un fichier, on va chercher à tirer parti de motifs répétés pour *encoder* l'information contenue de manière efficace.

Définition 36 **Un codage binaire** est un morphisme $\varphi : \Sigma^* \setminus \{\varepsilon\} \to \{0,1\}^+$. Un codage de **taille fixe** associe des mots de même taille k à chaque symbole. Si ce n'est pas le cas, c'est un codage à **taille variable**.

Définition 37 **Le code** d'un élément $w \in \Sigma^* \setminus \{\varepsilon\}$ est $\varphi(w)$. On dit alors que $\varphi(\Sigma)$ est l'ensemble des codes de φ.

Exemple 38 **Le coadge ASCII** est un codage de taille fixe de 8 bits.

Définition 39 **Le taux de compression** de w pour un codage binaire $\varphi : \{0,1\}^+ \to \{0,1\}^+$ est le rapport de taille $\frac{|\varphi(w)|}{|w|}$. On parle souvent de **taux de compression en moyenne**.

Définition 40 **Un codage φ est dit sans perte** si φ est inversible. Dans ce cas, on note φ^{-1} la **fonction de décodage**. C'est à dire que la donnée décompressée est la même que la donnée de départ.

A. Algorithme de Huffman [TOR 9.5.2.1]

Idée 41 Pour encoder le mot « banane », on vourdrait représenter les lettres « a » et « n » par des codes plus *courts* car elles apparaissent plus souvent.

Définition 42 **φ est un codage préfixe** si aucun code de φ n'est préfixe d'un autre code de φ. Cela permet un décodage sans ambiguïté i.e. φ est inversible.

Définition 43 **Un arbre binaire préfixe** associe à chaque symbole le chemin qui l'atteint depuis la racine.

Algorithme 44 **L'algorithme de Huffman** permet de construire, étant donné une fréquence f_ω d'apparition pour chaque symbole $\omega \in \Sigma$, un arbre préfixe binaire ayant la propriété d'associer aux symboles les plus probables un chemin plus court.

On peut alors retrouver le code Huffman(ω) en suivant le chemin de la racine à ω : si on suit le sous-arbre gauche, on ajoute un 0, sinon on ajoute un 1.

Théorème 45 **Optimalité de l'arbre de Huffman** L'arbre construit par l'algorithme de Huffman minimise la quantité $S = \Sigma f_\omega d_\omega$ où d_ω est la profondeur du symbole ω dans l'arbre, qui détermine directement la longueur de son code, et f_ω est la fréquence d'apparition de ce symbole.

Complexité 46 Soit $|\Sigma|$ le nombre de symboles distincts de notre alphabet, la construction de l'arbre de Huffman est en $\mathcal{O}(|\Sigma| \log(|\Sigma|))$.

B. Algorithme de Lempel-Ziv-Welch (LZW) [TOR 9.5.2.2]

Idée 47 Compresser les facteurs identiques vers une forme compacte.

DEV

Algorithme 48 **Compression** L'algorithme LZW utilise un dictionnaire, qui associe des sous-chaines à des codes. Ce dictionnaire est encodé par un arbre préfixe.

Shéma 49 **Exemple** [TOR p554]

dictionnaire	entrée	résultat
L ↦ 0; A ↦ 1; E ↦ 2; R ↦ 3	LALALALALERE	
LA ↦ 4; AL ↦ 5; LAL ↦ 6; LALA ↦ 7	...	...
ALE ↦ 8; ER ↦ 9; RE ↦ 10	...	0 1 4 6 5 2 3 2

Remarque 50 **Décompression** Le dictionnaire ne nécessite pas d'être transmis, il peut être reconstruit à la volée.

Remarque 51 **L'algorithme LZW procède en une seule passe** , ce qui en fait un algorithme adapté pour les fichiers volumineux ou les flux.

Application 52 **Le format d'archive ZIP** est souvent implémenté via l'algorithme DEFLATE, qui combine une variante de LZW et l'algorithme de Huffman.

Algorithmique du texte. Exemples et Applications.

I. Recherche de motif [CRO 1.3]

1 Motiv
2 Def Un alphabet Σ
3 Ex
4 Def Un mot, texte ou motif
5 Ex

A. Recherche naïve

6 Intuition
7 Algo Approche naïve
8 Complexe

B. Améliorations de la recherche naïve

9 Intuition
10 Algo L'algorithme de Rabin-Karp
11 Complexe
12 Prat
13 Algo L'algorithme de Boyer-Moore

C. Méthodes basées sur les automates

14 Motiv
15 App `grep`
16 Def Les expressions régulières
17 Thm Kleene
18 Algo L'algorithme regexp2dfa (ou Glushkov déterministe)

II. Mesure de similarité

A. Notion de distance [CRO 7.1]

19 Def La distance de Hamming
20 Ex
21 App Les codes correcteurs
22 Def La distance de Levenshtein
23 Rem
24 Ex
25 Rem D'autres distances d'édition

B. Problème de l'alignement [CRO 7.2]

26 Def Le problème de l'alignement optimal
27 Ex
28 Algo L'algorithme de Needleman–Wunsch
29 Algo L'algorithme d'Hirschberg [GUS 12.1.2]
30 App
31 App [CRO 8.2] La recherche de motif approximative
32 Algo L'algorithme de Sellers
33 Programme `fzf`
34 Prat

III. Compression

35 Motiv
36 Def Un codage binaire
37 Def Le code
38 Ex Le coadge ASCII
39 Def Le taux de compression
40 Def Un codage φ est dit sans perte

A. Algorithme de Huffman [TOR 9.5.2.1]

41 Idée
42 Def φ est un codage préfixe
43 Def Un arbre binaire préfixe
44 Algo L'algorithme de Huffman
45 Thm Optimalité de l'arbre de Huffman
46 Complexe

B. Algorithme de Lempel-Ziv-Welch (LZW) [TOR 9.5.2.2]

47 Idée
48 Algo Compression
49 Shéma Exemple [TOR p554]
50 Rem Décompression
51 Rem L'algorithme LZW procède en une seule passe
52 App Le format d'archive ZIP

Motivation

[TOR 9.5] L'algorithmique du texte importante car utilisée dans les éditeurs de texte, les outils de recherche de motif, la bioinformatique. Permet aussi d'introduire des méthodes algorithmiques avancées et des notions de combinatoire.

Ouverture

Ouverture 53 **La compression avec perte** pour un meilleur taux de compression, l'image, son ou vidéo avec les formats MP3 ou de JPEG par exemple.

Remarque

- Plan (stable depuis 2022)
- « Plus longue sous-séquence commune » possible surtout avec un dev Smith-Waterman
- « Analyse Lexical et Syntaxique » possible (orienté compil)

Devs

- Huffman bien,pas sûr que ça se réutilise
- Automate des occurences très bien (beaucoup de recasage)
- Boyer-Moore
- Rabin-Karpe à citer mais pas forcément en dev
- Lempel-Ziv pas mal
- Needleman-Wunsch ou Smith-Waterman
- Hirtschberg si vraiment motivés
- Recherche approximative de motif bon dev, très flexible (possibilité de faire purement algo, preuve de correction, etc)

Idées

- Exemples de programmes UNIX pour résoudre ces problèmes
- Plus de compilation (analyse lexicale etc.)
- Algorithme de Hirschberg's
- Applications classique : compilation, réseau, automate des occurences / Regexp2DFA, git
- Hash, checksum pas au programme, mais pertinent
- Parler de codage préfixe dans la partie compression, expliciter la partie décompression

Bibliographie

[CRO] M. Crochemore & C. Hancart & T. Lecroq, *Algorithmique du texte / Algorithms on strings.*

[GUS] D. Gusfield, *Algorithms for Strings, Trees, and Sequences.*

[TOR] T. Balabonski & S. Conchon & J. Filliâtre & K. Nguyen & L. Sartre, *MP2I MPI, Informatique Cours et exercices corrigés.*

Alois Rautureau - Alexis Hamon