Algorithmique du texte
HLIN608
Licence 3 Informatique

Alban MANCHERON
alban.mancheron@lirmm.fr
http://www.mancheron.infos.st/

Faculté des Sciences & Techniques de l'Université de Montpellier

Année universitaire 2016–2017, 2e semestre
[version 1.6]

Sommaire

I **Présentation (v. 1.6)**

II **Préambule (v. 1.4)**

III ***Dot-plots* et extraction de motifs (v. 1.9)**

I – Présentation (v. 1.7)
Sommaire

I.1/ Licence 3 Informatique

I.2/ Organisation du module

Licence 3 Informatique
Sommaire

I.1/ Licence 3 Informatique

I.2/ Organisation du module

Aspects pragmatiques

Aspects pragmatiques

- outils, logiciels, matériels

Aspects pragmatiques

- outils, logiciels, matériels
- analyse, manipulation, comparaison à $\pm$ large échelle de données bioinformatiques

Aspects pragmatiques

- outils, logiciels, matériels
- analyse, manipulation, comparaison à $\pm$ large échelle de données bioinformatiques
- Génomique, Transcriptomique, Protéomique, ...

Aspects pragmatiques

- outils, logiciels, matériels
- analyse, manipulation, comparaison à $\pm$ large échelle de données bioinformatiques
- Génomique, Transcriptomique, Protéomique, ...

$\vdots$

Aspects pragmatiques

- outils, logiciels, matériels
- analyse, manipulation, comparaison à $\pm$ large échelle de données bioinformatiques
- Génomique, Transcriptomique, Protéomique, ...

$\vdots$

Aspects informatiques

Licence 3 Informatique
Objectifs

Aspects pragmatiques

- outils, logiciels, matériels
- analyse, manipulation, comparaison à $\pm$ large échelle de données bioinformatiques
- Génomique, Transcriptomique, Protéomique, ...

⋮

Aspects informatiques

- structures de données

Licence 3 Informatique
Objectifs

Aspects pragmatiques

- outils, logiciels, matériels
- analyse, manipulation, comparaison à $\pm$ large échelle de données bioinformatiques
- Génomique, Transcriptomique, Protéomique, ...

$\vdots$

Aspects informatiques

- structures de données
- algorithmique

Licence 3 Informatique
Objectifs

Aspects pragmatiques

- outils, logiciels, matériels
- analyse, manipulation, comparaison à $\pm$ large échelle de données bioinformatiques
- Génomique, Transcriptomique, Protéomique, ...

⋮

Aspects informatiques

- structures de données
- algorithmique

⋮

Licence 3 Informatique
Objectifs

Aspects pragmatiques

- outils, logiciels, matériels
- analyse, manipulation, comparaison à $\pm$ large échelle de données bioinformatiques
- Génomique, Transcriptomique, Protéomique, ...

⋮

Aspects informatiques

- structures de données
- algorithmique

⋮

Aspects biologiques

Licence 3 Informatique
Objectifs

Aspects pragmatiques

- outils, logiciels, matériels
- analyse, manipulation, comparaison à $\pm$ large échelle de données bioinformatiques
- Génomique, Transcriptomique, Protéomique, ...

⋮

Aspects informatiques

- structures de données
- algorithmique

⋮

Aspects biologiques

- analyse/comparaison de séquences

Licence 3 Informatique
Objectifs

Aspects pragmatiques

- outils, logiciels, matériels
- analyse, manipulation, comparaison à $\pm$ large échelle de données bioinformatiques
- Génomique, Transcriptomique, Protéomique, ...

⋮

Aspects informatiques

- structures de données
- algorithmique

⋮

Aspects biologiques

- analyse/comparaison de séquences
- technologies de séquençages

Aspects pragmatiques

- outils, logiciels, matériels
- analyse, manipulation, comparaison à ± large échelle de données bioinformatiques
- Génomique, Transcriptomique, Protéomique, ...

⋮

Aspects informatiques

- structures de données
- algorithmique

⋮

Aspects biologiques

- analyse/comparaison de séquences
- technologies de séquençages
- *mapping*

Licence 3 Informatique
Objectifs

Aspects pragmatiques

- outils, logiciels, matériels
- analyse, manipulation, comparaison à $\pm$ large échelle de données bioinformatiques
- Génomique, Transcriptomique, Protéomique, ...

⋮

Aspects informatiques

- structures de données
- algorithmique

⋮

Aspects biologiques

- analyse/comparaison de séquences
- technologies de séquençages
- *mapping*

⋮

Organisation du module
Sommaire

I.1/ Licence 3 Informatique

I.2/ Organisation du module

- Emploi du temps
- Objectifs

Organisation du module
Emploi du temps

Équipe pédagogique

- Alban MANCHERON [13h30 CM/TD/TP]
  ⇒ Introduction, DotPlot simplifié, recherche/extraction/indexation de motifs

- Sylvain DAUDÉ [18h00 CM/TD/TP]
  ⇒ Bords/périodes, Mots de FIBONACCI, Alignement,
  ⇒ Sous-séquences communes, DotPlot.

- Annie CHATEAU (resp. UE) [13h30 CM/TD/TP]
  Assemblage *de novo*, Graphes et théorie de la complexité (problèmes NP)

Organisation du module
Emploi du temps

Équipe pédagogique

- Alban MANCHERON [13h30 CM/TD/TP]
  ⇒ Introduction, DotPlot simplifié, recherche/extraction/indexation de motifs

- Sylvain DAUDÉ [18h00 CM/TD/TP]
  ⇒ Bords/périodes, Mots de FIBONACCI, Alignement,
  ⇒ Sous-séquences communes, DotPlot.

- Annie CHATEAU (resp. UE) [13h30 CM/TD/TP]
  Assemblage *de novo*, Graphes et théorie de la complexité (problèmes NP)

Évaluation (5 ECTS)

- CP : 30% (projet+écrit)
- CT : 70%

Organisation du module
Objectifs

Pour nous [les enseignants]

Organisation du module
Objectifs

Pour nous [les enseignants]

- vous initier à l'algorithmique du texte

Organisation du module
Objectifs

Pour nous [les enseignants]

- vous initier à l'algorithmique du texte
- faire découvrir la bioinformatique

Organisation du module
Objectifs

Pour nous [les enseignants]

- vous initier à l'algorithmique du texte
- faire découvrir la bioinformatique
- faire comprendre la bioinformatique

Organisation du module
Objectifs

Pour nous [les enseignants]

- vous initier à l'algorithmique du texte
- faire découvrir la bioinformatique
- faire comprendre la bioinformatique

Pour vous [les étudiants]

Organisation du module
Objectifs

Pour nous [les enseignants]

- vous initier à l'algorithmique du texte
- faire découvrir la bioinformatique
- faire comprendre la bioinformatique

Pour vous [les étudiants]

- avoir plaisir à faire de l'algorithmique

Organisation du module
Objectifs

Pour nous [les enseignants]

- vous initier à l'algorithmique du texte
- faire découvrir la bioinformatique
- faire comprendre la bioinformatique

Pour vous [les étudiants]

- avoir plaisir à faire de l'algorithmique
- mieux choisir votre orientation

II – Préambule (v. 1.4)
Sommaire

II.1/ Rappels théoriques

II.2/ Rappels de complexité

II.3/ Bonnes pratiques

Rappels théoriques
Sommaire

II.1/ Rappels théoriques

- Algorithmique du texte
- Artillerie mathématique
- Ordres de grandeur

II.2/ Rappels de complexité

II.3/ Bonnes pratiques

Vocabulaire

Alphabet $\longrightarrow \Sigma = \{A, C, G, T\}$

Vocabulaire

Alphabet $\longrightarrow \Sigma = \{A, C, G, T\}$

Préfixe Facteur Suffixe

CAGCTGACTAGCTACGACT

HLIN608 – Algorithmique du texte.

Alban MANCHERON

Rappels théoriques

Algorithmique du texte

Artillerie mathématique

Ordres de grandeur

Rappels de complexité

Bonnes pratiques

Vocabulaire

Alphabet $\longrightarrow \Sigma = \{A, C, G, T\}$

Préfixe, Facteur, Suffixe

CAGCTGACTAGCTACGACT

Notations

- Le mot vide est noté ε. Il est de longueur 0
- La longueur du mot $s = \alpha_1\alpha_2 \cdots \alpha_n$ est égale à $|s| = n$
- Le mot $s[i..j]$ désigne le facteur commençant en i et terminant en j dans s
- On note $s[i]$ le symbole à la position i et $s[i..i] = s[i]$
- Le préfixe de longueur ℓ de s est $s[1..\ell]$
- Le suffixe de longueur ℓ de s est $s[n - \ell + 1..n]$

Vocabulaire

Alphabet $\longrightarrow \Sigma = \{A, C, G, T\}$

Préfixe, Facteur, Suffixe

CAGCTGACTAGCTACGACT

Notations

- Le mot vide est noté ε. Il est de longueur 0
- La longueur du mot $s = \alpha_1\alpha_2 \cdots \alpha_n$ est égale à $|s| = n$
- Le mot $s[i..j]$ désigne le facteur commençant en i et terminant en j dans s
- On note $s[i]$ le symbole à la position i et $s[i..i] = s[i]$
- Le préfixe de longueur ℓ de s est $s[1..\ell]$
- Le suffixe de longueur ℓ de s est $s[n - \ell + 1..n]$

Questions

- Combien de préfixes/suffixes/facteurs pour un mot de longueur n ?
- Combien de sous-séquences pour un mot de longueur n (sachant qu'une sous-séquence s'obtient en enlevant entre 0 et n symboles de la séquence initiale) ?

Notations

- Les sommes de suites de nombres sont notées par le symbole $\sum$
- Les produits de suites de nombres sont notées par le symbole $\prod$

Séries « classiques »

$$\sum_{i=m}^{n} 1 = n - m + 1$$

$$\sum_{i=0}^{n} 1 = n + 1$$

$$\sum_{i=m}^{n} i = \frac{(n-m+1)(m+n)}{2}$$

$$\sum_{i=0}^{n} i = \frac{n(n+1)}{2} = \sum_{i=1}^{n} i$$

$$\sum_{i=0}^{n} i^2 = \frac{n(n+1)(2n+1)}{6}$$

$$\sum_{i=0}^{n} i^3 = \left(\frac{n(n+1)}{2}\right)^2 = \left(\sum_{i=0}^{n} i\right)^2$$

$$\sum_{i=m}^{n} x^i = \frac{x^m - x^{n+1}}{1-x}$$

$$\sum_{i=0}^{n} x^i = \frac{1 - x^{n+1}}{1-x}$$

$$\sum_{i=0}^{n} \binom{n}{i} = 2^n = 1 + \sum_{i=0}^{n-1} 2^i$$

$$\sum_{i=1}^{n} i \binom{n}{i} = n\, 2^{n-1}$$

$$\sum_{i=0}^{n} \binom{i}{k} = \binom{n+1}{k+1}$$

$$(a+b)^n = \sum_{i=0}^{n} \binom{n}{i} a^{n-i} b^i$$

Séries « classiques »

$$\sum_{i=m}^{n} 1 = n - m + 1$$

$$\sum_{i=0}^{n} 1 = n + 1$$

$$\sum_{i=m}^{n} i = \frac{(n-m+1)(m+n)}{2}$$

$$\sum_{i=0}^{n} i = \frac{n(n+1)}{2} = \sum_{i=1}^{n} i$$

$$\sum_{i=0}^{n} i^2 = \frac{n(n+1)(2n+1)}{6}$$

$$\sum_{i=0}^{n} i^3 = \left(\frac{n(n+1)}{2}\right)^2 = \left(\sum_{i=0}^{n} i\right)^2$$

$$\sum_{i=m}^{n} x^i = \frac{x^m - x^{n+1}}{1-x}$$

$$\sum_{i=0}^{n} x^i = \frac{1 - x^{n+1}}{1-x}$$

$$\sum_{i=0}^{n} \binom{n}{i} = 2^n = 1 + \sum_{i=0}^{n-1} 2^i$$

$$\sum_{i=1}^{n} i \binom{n}{i} = n\, 2^{n-1}$$

$$\sum_{i=0}^{n} \binom{i}{k} = \binom{n+1}{k+1}$$

$$(a+b)^n = \sum_{i=0}^{n} \binom{n}{i} a^{n-i} b^i$$

TD : prouver les formules en rouge

Propriétés « classiques »

$$\sum_{n=a}^{b} \alpha\, f(n) = \alpha \sum_{n=a}^{b} f(n) \qquad \sum_{n=a}^{b} (f(n) + g(n)) = \sum_{n=a}^{b} f(n) + \sum_{n=a}^{b} g(n)$$

$$\sum_{n=a}^{b} f(n) = \sum_{n=a+m}^{b+m} f(n-m) \qquad \sum_{n=a}^{b} f(n) = \sum_{n=a}^{c} f(n) + \sum_{n=c+1}^{b} f(n)$$

$$\left(\sum_{n=a}^{b} f(n)\right) \times \left(\sum_{n=c}^{d} g(n)\right) = \sum_{m=a}^{b} \sum_{n=c}^{d} f(m)\, g(n) = \sum_{m=c}^{d} \sum_{n=a}^{b} f(n)\, g(m)$$

$$\sum_{n=a}^{b} \log f(n) = \log\left(\prod_{n=a}^{b} f(n)\right) \qquad \alpha^{\left(\sum_{n=a}^{b} f(n)\right)} = \prod_{n=a}^{b} \alpha^{f(n)}$$

$$\log_b b^n = b^{\log_b n} = n \qquad \log_a x = \frac{\log_b x}{\log_b a}$$

Les notations de LANDAU

- $O\left(g(n)\right) : \exists k > 0, n_0 \; \forall n > n_0 \; |f(n)| \leq k \, |g(n)|$
  g est asymptotiqu^{t} une borne supérieur de f à k près

Les notations de LANDAU

- $O\left(g(n)\right) : \exists k > 0, n_0\ \forall n > n_0\ |f(n)| \leq k\,|g(n)|$
  g est asymptotiqu^{t} une borne supérieur de f à k près
- $\Omega\left(g(n)\right) : \exists k > 0, n_0\ \forall n > n_0\ k\,|g(n)| \leq |f(n)|$
  g est asymptotiqu^{t} une borne inférieur de f à tout k près

Les notations de LANDAU

- $O\left(g(n)\right) : \exists k > 0, n_0\ \forall n > n_0\ |f(n)| \leq k\,|g(n)|$
  g est asymptotiqu^{t} une borne supérieur de f à k près
- $\Omega\left(g(n)\right) : \exists k > 0, n_0\ \forall n > n_0\ k\,|g(n)| \leq |f(n)|$
  g est asymptotiqu^{t} une borne inférieur de f à tout k près
- $\Theta\left(g(n)\right)$:
  $\exists k_1, k_2 > 0, n_0\ \forall n > n_0\ k_1\,|g(n)| < |f(n)| < k_2\,|g(n)|$
  g est asymptotiqu^{t} une borne inf. et sup. de f à k_1 , k_2 près

Les notations de LANDAU

- $O\left(g(n)\right) : \exists k > 0, n_0\ \forall n > n_0\ |f(n)| \leq k\,|g(n)|$
  g est asymptotiqu^{t} une borne supérieur de f à k près
- $\Omega\left(g(n)\right) : \exists k > 0, n_0\ \forall n > n_0\ k\,|g(n)| \leq |f(n)|$
  g est asymptotiqu^{t} une borne inférieur de f à tout k près
- $\Theta\left(g(n)\right)$:
  $\exists k_1, k_2 > 0, n_0\ \forall n > n_0\ k_1\,|g(n)| < |f(n)| < k_2\,|g(n)|$
  g est asymptotiqu^{t} une borne inf. et sup. de f à k_1 , k_2 près
- $o\left(g(n)\right) : \forall \varepsilon > 0\ \exists n_0\ \forall n > n_0\ |f(n)| \leq \varepsilon\,|g(n)|$
  f est asymptotiqu^{t} négligeable devant g

Les notations de LANDAU

- $O\left(g(n)\right) : \exists k > 0, n_0\ \forall n > n_0\ |f(n)| \leq k\,|g(n)|$
  g est asymptotiqu^{t} une borne supérieur de f à k près
- $\Omega\left(g(n)\right) : \exists k > 0, n_0\ \forall n > n_0\ k\,|g(n)| \leq |f(n)|$
  g est asymptotiqu^{t} une borne inférieur de f à tout k près
- $\Theta\left(g(n)\right)$:
  $\exists k_1, k_2 > 0, n_0\ \forall n > n_0\ k_1\,|g(n)| < |f(n)| < k_2\,|g(n)|$
  g est asymptotiqu^{t} une borne inf. et sup. de f à k_1 , k_2 près
- $o\left(g(n)\right) : \forall \varepsilon > 0\ \exists n_0\ \forall n > n_0\ |f(n)| \leq \varepsilon\,|g(n)|$
  f est asymptotiqu^{t} négligeable devant g
- $\omega\left(g(n)\right) : \forall k > 0\ \exists n_0\ \forall n > n_0\ k\,|g(n)| \leq |f(n)|$
  g est asymptotiqu^{t} négligeable devant f

Les notations de LANDAU

- $O\left(g(n)\right) : \exists k > 0, n_0\ \forall n > n_0\ |f(n)| \leq k\,|g(n)|$
  g est asymptotiqu^{t} une borne supérieur de f à k près
- $\Omega\left(g(n)\right) : \exists k > 0, n_0\ \forall n > n_0\ k\,|g(n)| \leq |f(n)|$
  g est asymptotiqu^{t} une borne inférieur de f à tout k près
- $\Theta\left(g(n)\right)$:
  $\exists k_1, k_2 > 0, n_0\ \forall n > n_0\ k_1\,|g(n)| < |f(n)| < k_2\,|g(n)|$
  g est asymptotiqu^{t} une borne inf. et sup. de f à k_1 , k_2 près
- $o\left(g(n)\right) : \forall \varepsilon > 0\ \exists n_0\ \forall n > n_0\ |f(n)| \leq \varepsilon\,|g(n)|$
  f est asymptotiqu^{t} négligeable devant g
- $\omega\left(g(n)\right) : \forall k > 0\ \exists n_0\ \forall n > n_0\ k\,|g(n)| \leq |f(n)|$
  g est asymptotiqu^{t} négligeable devant f
- $\sim g(n) : \forall \varepsilon > 0\ \exists n_0\ \forall n > n_0\ |f(n)/g(n) - 1| < \varepsilon$
  f et g sont asymptotiqu^{t} égales

Les notations de LANDAU

- $h(n) = O\left(g(n)\right)$ et $g(n) = O\left(f(n)\right)$
  $\Rightarrow \; h(n) = O\left(f(n)\right)$

Les notations de LANDAU

- $h(n) = O\left(g(n)\right)$ et $g(n) = O\left(f(n)\right)$
  $\Rightarrow \; h(n) = O\left(f(n)\right)$
- $g_1(n) = O\left(f_1(n)\right)$ et $g_2(n) = O\left(f_2(n)\right)$
  $\Rightarrow \; g_1(n) + g_2(n) = O\left(\max(f_1(n), f_2(n))\right)$

Les notations de LANDAU

- $h(n) = O\left(g(n)\right)$ et $g(n) = O\left(f(n)\right)$
  $\Rightarrow\ h(n) = O\left(f(n)\right)$
- $g_1(n) = O\left(f_1(n)\right)$ et $g_2(n) = O\left(f_2(n)\right)$
  $\Rightarrow\ g_1(n) + g_2(n) = O\left(\max(f_1(n), f_2(n))\right)$
- $g_1(n) = O\left(f_1(n)\right)$ et $g_2(n) = O\left(f_2(n)\right)$
  $\Rightarrow\ g_1(n)\, g_2(n) = O\left(f_1(n) f_2(n)\right)$

Les notations de LANDAU

- $h(n) = O\left(g(n)\right)$ et $g(n) = O\left(f(n)\right)$
  $\Rightarrow\ h(n) = O\left(f(n)\right)$
- $g_1(n) = O\left(f_1(n)\right)$ et $g_2(n) = O\left(f_2(n)\right)$
  $\Rightarrow\ g_1(n) + g_2(n) = O\left(\max(f_1(n), f_2(n))\right)$
- $g_1(n) = O\left(f_1(n)\right)$ et $g_2(n) = O\left(f_2(n)\right)$
  $\Rightarrow\ g_1(n)\, g_2(n) = O\left(f_1(n) f_2(n)\right)$
- $f(n) = \Theta\left(g(n)\right)$
  $\Leftrightarrow\ f(n) = O\left(g(n)\right)$ et $g(n) = O\left(f(n)\right)$

Rappels de complexité
Sommaire

II.1/ Rappels théoriques

II.2/ Rappels de complexité
- Mesure de la complexité
- Ordres de grandeur de la complexité
- Intérêt de la complexité

II.3/ Bonnes pratiques

HLIN608 – *Algorithmique du texte.*

Alban MANCHERON

Rappels théoriques

Rappels de complexité

Mesure de la complexité

Ordres de grandeur de la complexité

Intérêt de la complexité

Bonnes pratiques

?

En temps

En temps

En espace

HLIN608 – Algorithmique du texte.
Alban MANCHERON
Rappels théoriques
Rappels de complexité
Mesure de la complexité
Ordres de grandeur de la complexité
Intérêt de la complexité
Bonnes pratiques

En temps

- nombre d'opérations effectuées lors du déroulement de l'algorithme

En espace

HLIN608 – *Algorithmique du texte.*

Alban MANCHERON

Rappels théoriques

Rappels de complexité

Mesure de la complexité

Ordres de grandeur de la complexité

Intérêt de la complexité

Bonnes pratiques

En temps

- nombre d'opérations effectuées lors du déroulement de l'algorithme

En espace

- mémoire nécessaire pour le stockage des données calculées lors du déroulement de l'algorithme

En temps

- nombre d'opérations effectuées lors du déroulement de l'algorithme

En espace

- mémoire nécessaire pour le stockage des données calculées lors du déroulement de l'algorithme

Dans tous les cas :

En temps

- nombre d'opérations effectuées lors du déroulement de l'algorithme

En espace

- mémoire nécessaire pour le stockage des données calculées lors du déroulement de l'algorithme

Dans tous les cas :

- mesures en fonction de la forme des données en entrée du problème

En temps

- nombre d'opérations effectuées lors du déroulement de l'algorithme

En espace

- mémoire nécessaire pour le stockage des données calculées lors du déroulement de l'algorithme

Dans tous les cas :

- mesures en fonction de la forme des données en entrée du problème
- étude des cas particuliers

En temps

- nombre d'opérations effectuées lors du déroulement de l'algorithme

En espace

- mémoire nécessaire pour le stockage des données calculées lors du déroulement de l'algorithme

Dans tous les cas :

- mesures en fonction de la forme des données en entrée du problème
- étude des cas particuliers
- ordre de grandeur (*e.g.*, logarithmique, linéaire, polynomial, exponentiel).

En temps

- nombre d'opérations effectuées lors du déroulement de l'algorithme

En espace

- mémoire nécessaire pour le stockage des données calculées lors du déroulement de l'algorithme

Dans tous les cas :

- mesures en fonction de la forme des données en entrée du problème
- étude des cas particuliers
- ordre de grandeur (*e.g.*, logarithmique, linéaire, polynomial, exponentiel).

En temps

- nombre d'opérations effectuées lors du déroulement de l'algorithme

En espace

- mémoire nécessaire pour le stockage des données calculées lors du déroulement de l'algorithme

Dans tous les cas :

- mesures en fonction de la forme des données en entrée du problème
- étude des cas particuliers
- ordre de grandeur (*e.g.*, logarithmique, linéaire, polynomial, exponentiel).

En temps

- nombre d'opérations effectuées lors du déroulement de l'algorithme

En espace

- mémoire nécessaire pour le stockage des données calculées lors du déroulement de l'algorithme

Dans tous les cas :

- mesures en fonction de la forme des données en entrée du problème
- étude des cas particuliers
- ordre de grandeur (*e.g.*, logarithmique, linéaire, polynomial, exponentiel).

En temps

- nombre d'opérations effectuées lors du déroulement de l'algorithme

En espace

- mémoire nécessaire pour le stockage des données calculées lors du déroulement de l'algorithme

Dans tous les cas :

- mesures en fonction de la forme des données en entrée du problème
- étude des cas particuliers
- ordre de grandeur (*e.g.*, logarithmique, linéaire, polynomial, exponentiel).

En temps

- nombre d'opérations effectuées lors du déroulement de l'algorithme

En espace

- mémoire nécessaire pour le stockage des données calculées lors du déroulement de l'algorithme

Dans tous les cas :

- mesures en fonction de la forme des données en entrée du problème
- étude des cas particuliers
- ordre de grandeur (*e.g.*, logarithmique, linéaire, polynomial, exponentiel).

En temps

- nombre d'opérations effectuées lors du déroulement de l'algorithme

En espace

- mémoire nécessaire pour le stockage des données calculées lors du déroulement de l'algorithme

Dans tous les cas :

- mesures en fonction de la forme des données en entrée du problème
- étude des cas particuliers
- ordre de grandeur (*e.g.*, logarithmique, linéaire, polynomial, exponentiel).

Rappels de complexité
Ordres de grandeur de la complexité

Rappels de complexité
Intérêt de la complexité

Rappels de complexité
Intérêt de la complexité

Rappels de complexité
Intérêt de la complexité

Rappels de complexité
Intérêt de la complexité

Rappels de complexité
Intérêt de la complexité

Rappels de complexité
Intérêt de la complexité

Rappels de complexité
Intérêt de la complexité

Vocabulaire

Alphabet $\longrightarrow \Sigma = \{A, C, G, T\}$

Préfixe, Facteur, Suffixe

CAGCTGACTAGCTACGACT

Notations

- Le mot vide est noté ε. Il est de longueur 0
- La longueur du mot $s = \alpha_1\alpha_2 \cdots \alpha_n$ est égale à $|s| = n$
- Le mot $s[i..j]$ désigne le facteur commençant en i et terminant en j dans s
- On note $s[i]$ le symbole à la position i et $s[i..i] = s[i]$
- Le préfixe de longueur ℓ de s est $s[1..\ell]$
- Le suffixe de longueur ℓ de s est $s[n - \ell + 1..n]$

Questions

- Combien de préfixes/suffixes/facteurs pour un mot de longueur n ?
- Combien de sous-séquences pour un mot de longueur n (sachant qu'une sous-séquence s'obtient en enlevant entre 0 et n symboles de la séquence initiale) ?

Rappels de complexité
Intérêt de la complexité

Vocabulaire

Alphabet $\longrightarrow \Sigma = \{A, C, G, T\}$

Préfixe, Facteur, Suffixe

CAGCTGACTAGCTACGACT

Notations

- Le mot vide est noté ε. Il est de longueur 0
- La longueur du mot $s = \alpha_1\alpha_2 \cdots \alpha_n$ est égale à $|s| = n$
- Le mot $s[i..j]$ désigne le facteur commençant en i et terminant en j dans s
- On note $s[i]$ le symbole à la position i et $s[i..i] = s[i]$
- Le préfixe de longueur ℓ de s est $s[1..\ell]$
- Le suffixe de longueur ℓ de s est $s[n - \ell + 1..n]$

Questions

- Combien de préfixes/suffixes/facteurs pour un mot de longueur n ?
- Combien de sous-séquences pour un mot de longueur n (sachant qu'une sous-séquence s'obtient en enlevant entre 0 et n symboles de la séquence initiale) ?

II.1/ Rappels théoriques

II.2/ Rappels de complexité

II.3/ Bonnes pratiques

Bonnes pratiques

Réflexion

Bonnes pratiques

Réflexion

- Prendre le temps de l'analyse

Bonnes pratiques

Réflexion

- Prendre le temps de l'analyse
- Réfléchir aux structures de données

Bonnes pratiques

Réflexion

- Prendre le temps de l'analyse
- Réfléchir aux structures de données
- Estimer les paramètres de complexité du problème

Bonnes pratiques

Réflexion

- Prendre le temps de l'analyse
- Réfléchir aux structures de données
- Estimer les paramètres de complexité du problème

Programmation

Bonnes pratiques

Réflexion

- Prendre le temps de l'analyse
- Réfléchir aux structures de données
- Estimer les paramètres de complexité du problème

Programmation

- Coder proprement

Bonnes pratiques

Réflexion

- Prendre le temps de l'analyse
- Réfléchir aux structures de données
- Estimer les paramètres de complexité du problème

Programmation

- Coder proprement
- Choisir un langage approprié

Bonnes pratiques

Réflexion

- Prendre le temps de l'analyse
- Réfléchir aux structures de données
- Estimer les paramètres de complexité du problème

Programmation

- Coder proprement
- Choisir un langage approprié
- Réaliser des tests unitaires

Bonnes pratiques

Réflexion

- Prendre le temps de l'analyse
- Réfléchir aux structures de données
- Estimer les paramètres de complexité du problème

Programmation

- Coder proprement
- Choisir un langage approprié
- Réaliser des tests unitaires
- Appliquer le paradigme « Modèle-Vue-Contrôleur » (MVC)

Bonnes pratiques

Réflexion

- Prendre le temps de l'analyse
- Réfléchir aux structures de données
- Estimer les paramètres de complexité du problème

Programmation

- Coder proprement
- Choisir un langage approprié
- Réaliser des tests unitaires
- Appliquer le paradigme « Modèle-Vue-Contrôleur » (MVC)

Études

Bonnes pratiques

Réflexion

- Prendre le temps de l'analyse
- Réfléchir aux structures de données
- Estimer les paramètres de complexité du problème

Programmation

- Coder proprement
- Choisir un langage approprié
- Réaliser des tests unitaires
- Appliquer le paradigme « Modèle-Vue-Contrôleur » (MVC)

Études

- Poser des questions à bon escient

Bonnes pratiques

Réflexion

- Prendre le temps de l'analyse
- Réfléchir aux structures de données
- Estimer les paramètres de complexité du problème

Programmation

- Coder proprement
- Choisir un langage approprié
- Réaliser des tests unitaires
- Appliquer le paradigme « Modèle-Vue-Contrôleur » (MVC)

Études

- Poser des questions à bon escient
- Être curieux

Bonnes pratiques

Réflexion

- Prendre le temps de l'analyse
- Réfléchir aux structures de données
- Estimer les paramètres de complexité du problème

Programmation

- Coder proprement
- Choisir un langage approprié
- Réaliser des tests unitaires
- Appliquer le paradigme « Modèle-Vue-Contrôleur » (MVC)

Études

- Poser des questions à bon escient
- Être curieux
- Lire les documentations

Bonnes pratiques

Réflexion

- Prendre le temps de l'analyse
- Réfléchir aux structures de données
- Estimer les paramètres de complexité du problème

Programmation

- Coder proprement
- Choisir un langage approprié
- Réaliser des tests unitaires
- Appliquer le paradigme « Modèle-Vue-Contrôleur » (MVC)

Études

- Poser des questions à bon escient
- Être curieux
- Lire les documentations
- Regarder ce que les autres font

III – *Dot-plots* et extraction de motifs (v. 1.9)
Sommaire

III.1/ Alignement visuel de séquences

III.2/ Recherche de motif

III.3/ Extraction de motif

Alignement visuel de séquences
Sommaire

III.1/ Alignement visuel de séquences

- Présentation
- Intérêt de la comparaison de séquences
- Dot plot

III.2/ Recherche de motif

III.3/ Extraction de motif

HLIN608 – Algorithmique du texte.
Alban MANCHERON
Alignement visuel de séquences
Présentation
Intérêt de la comparaison de séquences
Dot plot
Recherche de motif
Extraction de motif

Alignement de séquences

HLIN608 – Algorithmique du texte.
Alban MANCHERON
Alignement visuel de séquences
Présentation
Intérêt de la comparaison de séquences
Dot plot
Recherche de motif
Extraction de motif

Alignement de séquences

- Problème représentatif de la bioinformatique

Alignement de séquences

- Problème représentatif de la bioinformatique
  - ⇒ Un des outils de bioinformatique les plus utilisés

HLIN608 – Algorithmique du texte.
Alban MANCHERON
Alignement visuel de séquences
Présentation
Intérêt de la comparaison de séquences
Dot plot
Recherche de motif
Extraction de motif

Alignement de séquences

- Problème représentatif de la bioinformatique
  - ⇒ Un des outils de bioinformatique les plus utilisés
- Solution graphique et algorithmique

Alignement de séquences

- Problème représentatif de la bioinformatique
  - ⇒ Un des outils de bioinformatique les plus utilisés
- Solution graphique et algorithmique
- Algorithme de base élégant

Alignement de séquences

- Problème représentatif de la bioinformatique
  - ⇒ Un des outils de bioinformatique les plus utilisés
- Solution graphique et algorithmique
- Algorithme de base élégant
- Force et limite de l'informatique en biologie

Alignement de séquences

- Problème représentatif de la bioinformatique
  - ⇒ Un des outils de bioinformatique les plus utilisés
- Solution graphique et algorithmique
- Algorithme de base élégant
- Force et limite de l'informatique en biologie

Problématique

Alignement de séquences

- Problème représentatif de la bioinformatique
  - ⇒ Un des outils de bioinformatique les plus utilisés
- Solution graphique et algorithmique
- Algorithme de base élégant
- Force et limite de l'informatique en biologie

Problématique

- Recherche des similitudes entre plusieurs séquences

Alignement de séquences

- Problème représentatif de la bioinformatique
  - ⇒ Un des outils de bioinformatique les plus utilisés
- Solution graphique et algorithmique
- Algorithme de base élégant
- Force et limite de l'informatique en biologie

Problématique

- Recherche des similitudes entre plusieurs séquences
- Substitution, Insertion/Suppression (*InDel*)

Alignement de séquences

- Problème représentatif de la bioinformatique
  - ⇒ Un des outils de bioinformatique les plus utilisés
- Solution graphique et algorithmique
- Algorithme de base élégant
- Force et limite de l'informatique en biologie

Problématique

- Recherche des similitudes entre plusieurs séquences
- Substitution, Insertion/Suppression (*InDel*)

Exemple (POMME *vs.* GOMME)

```
P O M M E
G O M M E
```

Alignement de séquences

- Problème représentatif de la bioinformatique
  - ⇒ Un des outils de bioinformatique les plus utilisés
- Solution graphique et algorithmique
- Algorithme de base élégant
- Force et limite de l'informatique en biologie

Problématique

- Recherche des similitudes entre plusieurs séquences
- Substitution, Insertion/Suppression (*InDel*)

Exemple (POMME *vs.* GOMME)

```
P O M M E
G O M M E
* O M M E
```

Alignement de séquences

- Problème représentatif de la bioinformatique
  - ⇒ Un des outils de bioinformatique les plus utilisés
- Solution graphique et algorithmique
- Algorithme de base élégant
- Force et limite de l'informatique en biologie

Problématique

- Recherche des similitudes entre plusieurs séquences
- Substitution, Insertion/Suppression (*InDel*)

Exemple (OCTOBRE *vs.* OTARIE)

```
O C T O B R E
O T A R I E
```

Alignement de séquences

- Problème représentatif de la bioinformatique
  - ⇒ Un des outils de bioinformatique les plus utilisés
- Solution graphique et algorithmique
- Algorithme de base élégant
- Force et limite de l'informatique en biologie

Problématique

- Recherche des similitudes entre plusieurs séquences
- Substitution, Insertion/Suppression (*InDel*)

Exemple (OCTOBRE *vs.* OTARIE)

```
O C T O B R E
O T A R I E
O * * * * * -
```

Alignement de séquences

- Problème représentatif de la bioinformatique
  - ⇒ Un des outils de bioinformatique les plus utilisés
- Solution graphique et algorithmique
- Algorithme de base élégant
- Force et limite de l'informatique en biologie

Problématique

- Recherche des similitudes entre plusieurs séquences
- Substitution, Insertion/Suppression (*InDel*)

Exemple (OCTOBRE *vs.* OTARIE)

```
O C T O B R E
O - T A R I E
O - T * * * E
```

Alignement visuel de séquences
Présentation

Alignement de séquences

- Problème représentatif de la bioinformatique
  - ⇒ Un des outils de bioinformatique les plus utilisés
- Solution graphique et algorithmique
- Algorithme de base élégant
- Force et limite de l'informatique en biologie

Problématique

- Recherche des similitudes entre plusieurs séquences
- Substitution, Insertion/Suppression (*InDel*)

Exemple (OCTOBRE *vs.* OTARIE)

```
O C T O B R E
O - T A - R I E
O - T * - R * -
```

Alignement de séquences

- Problème représentatif de la bioinformatique
  - ⇒ Un des outils de bioinformatique les plus utilisés
- Solution graphique et algorithmique
- Algorithme de base élégant
- Force et limite de l'informatique en biologie

Problématique

- Recherche des similitudes entre plusieurs séquences
- Substitution, Insertion/Suppression (*InDel*)

Exemple (OCTOBRE *vs.* OTARIE)

```
O C T O B R - E
O - T A - R I E
O - T * - R - E
```

Biologie moléculaire

HLIN608 – Algorithmique du texte.
Alban MANCHERON
Alignement visuel de séquences
Présentation
Intérêt de la comparaison de séquences
Dot plot
Recherche de motif
Extraction de motif

Biologie moléculaire

- Séquences (ADN, ARN, acides aminés, . . .)

HLIN608 – Algorithmique du texte.
Alban MANCHERON
Alignement visuel de séquences
Présentation
Intérêt de la comparaison de séquences
Dot plot
Recherche de motif
Extraction de motif

Biologie moléculaire

- Séquences (ADN, ARN, acides aminés, . . .)
- Recherche approchée dans une base de données

Biologie moléculaire

- Séquences (ADN, ARN, acides aminés, . . .)
- Recherche approchée dans une base de données
- Prédiction de structure

HLIN608 – *Algorithmique du texte.*

Alban MANCHERON

Alignement visuel de séquences

Présentation

Intérêt de la comparaison de séquences

Dot plot

Recherche de motif

Extraction de motif

Biologie moléculaire

- Séquences (ADN, ARN, acides aminés, . . .)
- Recherche approchée dans une base de données
- Prédiction de structure
- Prédiction de fonction

HLIN608 – Algorithmique du texte.

Alban MANCHERON

Alignement visuel de séquences

Présentation

Intérêt de la comparaison de séquences

Dot plot

Recherche de motif

Extraction de motif

Biologie moléculaire

- Séquences (ADN, ARN, acides aminés, . . .)
- Recherche approchée dans une base de données
- Prédiction de structure
- Prédiction de fonction
- Phylogénie (histoire évolutive des espèces)

HLIN608 – *Algorithmique du texte.*

Alban MANCHERON

Alignement visuel de séquences

Présentation

Intérêt de la comparaison de séquences

Dot plot

Recherche de motif

Extraction de motif

Biologie moléculaire

- Séquences (ADN, ARN, acides aminés, . . .)
- Recherche approchée dans une base de données
- Prédiction de structure
- Prédiction de fonction
- Phylogénie (histoire évolutive des espèces)

Informatique

Biologie moléculaire

- Séquences (ADN, ARN, acides aminés, ...)
- Recherche approchée dans une base de données
- Prédiction de structure
- Prédiction de fonction
- Phylogénie (histoire évolutive des espèces)

Informatique

- Comparaison de fichiers (1 ligne $\sim$ 1 caractère)

Alignement visuel de séquences
Intérêt de la comparaison de séquences

Biologie moléculaire

- Séquences (ADN, ARN, acides aminés, ...)
- Recherche approchée dans une base de données
- Prédiction de structure
- Prédiction de fonction
- Phylogénie (histoire évolutive des espèces)

Informatique

- Comparaison de fichiers (1 ligne $\sim$ 1 caractère)
  - $\Rightarrow$ `diff` sous UNIX/Linux

Alignement visuel de séquences
Intérêt de la comparaison de séquences

Biologie moléculaire

- Séquences (ADN, ARN, acides aminés, ...)
- Recherche approchée dans une base de données
- Prédiction de structure
- Prédiction de fonction
- Phylogénie (histoire évolutive des espèces)

Informatique

- Comparaison de fichiers (1 ligne $\sim$ 1 caractère)
  - $\Rightarrow$ `diff` sous UNIX/Linux
- Recherche approchée dans un dictionnaire (correcteur orthographique, fautes de frappes), ...

Alignement visuel de séquences
Intérêt de la comparaison de séquences

Biologie moléculaire

- Séquences (ADN, ARN, acides aminés, ...)
- Recherche approchée dans une base de données
- Prédiction de structure
- Prédiction de fonction
- Phylogénie (histoire évolutive des espèces)

Informatique

- Comparaison de fichiers (1 ligne $\sim$ 1 caractère)
  - $\Rightarrow$ `diff` sous UNIX/Linux
- Recherche approchée dans un dictionnaire (correcteur orthographique, fautes de frappes), ...

Autres applications

Alignement visuel de séquences
Intérêt de la comparaison de séquences

Biologie moléculaire

- Séquences (ADN, ARN, acides aminés, ...)
- Recherche approchée dans une base de données
- Prédiction de structure
- Prédiction de fonction
- Phylogénie (histoire évolutive des espèces)

Informatique

- Comparaison de fichiers (1 ligne $\sim$ 1 caractère)
  - $\Rightarrow$ `diff` sous UNIX/Linux
- Recherche approchée dans un dictionnaire (correcteur orthographique, fautes de frappes), ...

Autres applications

- Reconnaissance vocale

Alignement visuel de séquences
Intérêt de la comparaison de séquences

Biologie moléculaire

- Séquences (ADN, ARN, acides aminés, ...)
- Recherche approchée dans une base de données
- Prédiction de structure
- Prédiction de fonction
- Phylogénie (histoire évolutive des espèces)

Informatique

- Comparaison de fichiers (1 ligne $\sim$ 1 caractère)
  - $\Rightarrow$ `diff` sous UNIX/Linux
- Recherche approchée dans un dictionnaire (correcteur orthographique, fautes de frappes), ...

Autres applications

- Reconnaissance vocale
- ⋮

Alignement visuel de séquences
Dot plot [Exemple]

- Réponse visuelle au problème

Alignement visuel de séquences
Présentation
Intérêt de la comparaison de séquences
Dot plot
Exemple
Programmation
Filtrage
Recherche de motif
Extraction de motif

Alignement visuel de séquences
Dot plot [Exemple]

- Réponse visuelle au problème
- Largement utilisée

Alignement visuel de séquences
Présentation
Intérêt de la comparaison de séquences
Dot plot
Exemple
Programmation
Filtrage
Recherche de motif
Extraction de motif

Alignement visuel de séquences
Dot plot [Exemple]

- Réponse visuelle au problème
- Largement utilisée

Alignement visuel de séquences
Présentation
Intérêt de la comparaison de séquences
Dot plot
Exemple
Programmation
Filtrage
Recherche de motif
Extraction de motif

Alignement visuel de séquences
Dot plot [Exemple]

- Réponse visuelle au problème
- Largement utilisée

Alignement visuel de séquences
Présentation
Intérêt de la comparaison de séquences
Dot plot
Exemple
Programmation
Filtrage
Recherche de motif
Extraction de motif

Alignement visuel de séquences
Dot plot [Exemple]

- Réponse visuelle au problème
- Largement utilisée

Alignement visuel de séquences
Dot plot [Exemple]

- Réponse visuelle au problème
- Largement utilisée

Alignement visuel de séquences
Dot plot [Exemple]

- Réponse visuelle au problème
- Largement utilisée

- Importance de la représentation graphique pour percevoir l'information

  ⇒ « Nettoyage » de l'affichage

Alignement visuel de séquences
Présentation
Intérêt de la comparaison de séquences
Dot plot
Exemple
Programmation
Filtrage
Recherche de motif
Extraction de motif

DotPlot simple

```
int** dotPlot(const string &s1, const string &s2) {
  size_t l1 = s1.length(), l2 = s2.length();
  int** res = new int* [l1];
  for (size_t i = 0; i < l1; i++) {
    res[i] = new int [l2];
    for (size_t j = 0; j < l2; j++) {
      res[i][j] = (s1[i] == s2[j]);
    }
  }
  return res;
}
```

Alignement visuel de séquences
Dot plot [Filtrage 1/3]

Alignement visuel de séquences
Présentation
Intérêt de la comparaison de séquences
Dot plot
Exemple
Programmation
Filtrage
Recherche de motif
Extraction de motif

- L'idée de base est de ne marquer que les « points » qui sont dans un motif potentiellement d'intérêt.

Alignement visuel de séquences
Dot plot [Filtrage 1/3]

Alignement visuel de séquences
Présentation
Intérêt de la comparaison de séquences
Dot plot
Exemple
Programmation
Filtrage
Recherche de motif
Extraction de motif

- L'idée de base est de ne marquer que les « points » qui sont dans un motif potentiellement d'intérêt.
  - $\Rightarrow$ $res[i][j] \leftarrow Vrai$ si et seulement si $s_1[i-k..i+k] = s_2[j-k..j+k]$ (pour $k > 0$)

Alignement visuel de séquences
Dot plot [Filtrage 1/3]

Alignement visuel de séquences
Présentation
Intérêt de la comparaison de séquences
Dot plot
Exemple
Programmation
Filtrage
Recherche de motif
Extraction de motif

- L'idée de base est de ne marquer que les « points » qui sont dans un motif potentiellement d'intérêt.
  - ⇒ $res[i][j] \leftarrow Vrai$ si et seulement si $s_1[i-k..i+k] = s_2[j-k..j+k]$ (pour $k > 0$)
  - ⇒ Filtrage moins brutal : seuil de similarité entre motifs

Alignement visuel de séquences
Dot plot [Filtrage 1/3]

Alignement visuel de séquences
Présentation
Intérêt de la comparaison de séquences
Dot plot
Exemple
Programmation
Filtrage
Recherche de motif
Extraction de motif

- L'idée de base est de ne marquer que les « points » qui sont dans un motif potentiellement d'intérêt.
  - ⇒ $res[i][j] \leftarrow Vrai$ si et seulement si $s_1[i-k..i+k] = s_2[j-k..j+k]$ (pour $k > 0$)
  - ⇒ Filtrage moins brutal : seuil de similarité entre motifs
- Attention aux limites des tableaux

Alignement visuel de séquences
Dot plot [Filtrage 1/3]

Alignement visuel de séquences
Présentation
Intérêt de la comparaison de séquences
Dot plot
Exemple
Programmation
Filtrage
Recherche de motif
Extraction de motif

- L'idée de base est de ne marquer que les « points » qui sont dans un motif potentiellement d'intérêt.
  - ⇒ $res[i][j] \leftarrow Vrai$ si et seulement si $s_1[i-k..i+k] = s_2[j-k..j+k]$ (pour $k > 0$)
  - ⇒ Filtrage moins brutal : seuil de similarité entre motifs
- Attention aux limites des tableaux
- Motifs complémentaires/inversés

Dot plot (50 × 80) sans filtrage

Dot plot (50 × 80) avec filtrage [k=1]

Dot plot (50 × 80) avec filtrage [k=1]

Dot plot (50 × 80) avec filtrage [k=2]

Dot plot (50×80) avec filtrage [k=2]

- Complexité ?

Dot plot (50 × 80) avec filtrage [k=2]

- Complexité ?
  - ⇒ Dans le cas d'une identité totale on peut améliorer cette complexité
  - ⇒ Le remplissage d'une ligne revient à faire une recherche de motif exact dans un mot

Recherche de motif
Sommaire

III.1/ Alignement visuel de séquences

III.2/ Recherche de motif
- Algorithme naïf
- Illustration
- Algorithme KMP
- Calcul du bord maximal

III.3/ Extraction de motif

Recherche exacte

```
1  #include <iostream>
2  #include <string>
3  #include <libgen.h>
4
5  using namespace std;
6
7  int main(int argc, char** argv) {
8    string txt = "ACACTCCACT", mot = "ACT";
9    size_t l1 = txt.length(), l2 = mot.length();
10
11   for (size_t i=0; i < l1; i++) {
12     for (size_t j = 0; i + j < l1 && j < l2 && txt[i + j] == mot[j]; j++) {
13       if (j == l2 - 1) {
14         cout << "Found at location " << i << endl;
15       }
16     }
17   }
18   return 0;
19 }
```

Recherche de motif
Algorithme naïf [Programmation]

Complexité ?

Recherche exacte

```
1  #include <iostream>
2  #include <string>
3  #include <libgen.h>
4
5  using namespace std;
6
7  int main(int argc, char** argv) {
8    string txt = "ACACTCCACT", mot = "ACT";
9    size_t l1 = txt.length(), l2 = mot.length();
10
11   for (size_t i=0; i < l1; i++) {
12     for (size_t j = 0; i + j < l1 && j < l2 && txt[i + j] == mot[j]; j++) {
13       if (j == l2 - 1) {
14         cout << "Found at location " << i << endl;
15       }
16     }
17   }
18   return 0;
19 }
```

Recherche de motif
Algorithme naïf [Programmation]

Complexité ?

⇒ Pire de cas :

⇒ Meilleur de cas :

⇒ Cas moyen :

```
1  #include <iostream>
2  #include <string>
3  #include <libgen.h>
4
5  using namespace std;
6
7  int main(int argc, char** argv) {
8    string txt = "ACACTCCACT", mot = "ACT";
9    size_t l1 = txt.length(), l2 = mot.length();
10
11   for (size_t i=0; i < l1; i++) {
12     for (size_t j = 0; i + j < l1 && j < l2 && txt[i + j] == mot[j]; j++) {
13       if (j == l2 - 1) {
14         cout << "Found at location " << i << endl;
15       }
16     }
17   }
18   return 0;
19 }
```

Recherche de motif
Algorithme naïf [Programmation]

Complexité ?

⇒ Pire de cas : $O(n\,k)$

⇒ Meilleur de cas :

⇒ Cas moyen :

```
1  #include <iostream>
2  #include <string>
3  #include <libgen.h>
4
5  using namespace std;
6
7  int main(int argc, char** argv) {
8    string txt = "ACACTCCACT", mot = "ACT";
9    size_t l1 = txt.length(), l2 = mot.length();
10
11   for (size_t i=0; i < l1; i++) {
12     for (size_t j = 0; i + j < l1 && j < l2 && txt[i + j] == mot[j]; j++) {
13       if (j == l2 - 1) {
14         cout << "Found at location " << i << endl;
15       }
16     }
17   }
18   return 0;
19 }
```

Recherche de motif
Algorithme naïf [Programmation]

Complexité ?

⇒ Pire de cas : $O(n\,k)$

- Recherche de `AAAAAAAC` dans le mot `AAAAAAAAAAAAAAAAAAAAAAAAAAAAAA`

⇒ Meilleur de cas :

⇒ Cas moyen :

```
1  #include <iostream>
2  #include <string>
3  #include <libgen.h>
4
5  using namespace std;
6
7  int main(int argc, char** argv) {
8    string txt = "ACACTCCACT", mot = "ACT";
9    size_t l1 = txt.length(), l2 = mot.length();
10
11   for (size_t i=0; i < l1; i++) {
12     for (size_t j = 0; i + j < l1 && j < l2 && txt[i + j] == mot[j]; j++) {
13       if (j == l2 - 1) {
14         cout << "Found at location " << i << endl;
15       }
16     }
17   }
18   return 0;
19 }
```

Recherche de motif
Algorithme naïf [Programmation]

Complexité ?

⇒ Pire de cas : $O(n\,k)$

- Recherche de `AAAAAAAC` dans le mot `AAAAAAAAAAAAAAAAAAAAAAAAAAAAAA`

⇒ Meilleur de cas : $O(n)$

⇒ Cas moyen :

```
1  #include <iostream>
2  #include <string>
3  #include <libgen.h>
4
5  using namespace std;
6
7  int main(int argc, char** argv) {
8    string txt = "ACACTCCACT", mot = "ACT";
9    size_t l1 = txt.length(), l2 = mot.length();
10
11   for (size_t i=0; i < l1; i++) {
12     for (size_t j = 0; i + j < l1 && j < l2 && txt[i + j] == mot[j]; j++) {
13       if (j == l2 - 1) {
14         cout << "Found at location " << i << endl;
15       }
16     }
17   }
18   return 0;
19 }
```

Recherche de motif
Algorithme naïf [Programmation]

Complexité ?

⇒ Pire de cas : $O(n\,k)$

- Recherche de `AAAAAAAC` dans le mot `AAAAAAAAAAAAAAAAAAAAAAAAAAAAAA`

⇒ Meilleur de cas : $O(n)$

- Recherche de `CCCCCCCC` dans le mot `AAAAAAAAAAAAAAAAAAAAAAAAAAAAAA`

⇒ Cas moyen :

```
1  #include <iostream>
2  #include <string>
3  #include <libgen.h>
4
5  using namespace std;
6
7  int main(int argc, char** argv) {
8    string txt = "ACACTCCACT", mot = "ACT";
9    size_t l1 = txt.length(), l2 = mot.length();
10
11   for (size_t i=0; i < l1; i++) {
12     for (size_t j = 0; i + j < l1 && j < l2 && txt[i + j] == mot[j]; j++) {
13       if (j == l2 - 1) {
14         cout << "Found at location " << i << endl;
15       }
16     }
17   }
18   return 0;
19 }
```

Recherche de motif
Algorithme naïf [Programmation]

Complexité ?

⇒ Pire de cas : $O(n\,k)$

- Recherche de `AAAAAAAC` dans le mot `AAAAAAAAAAAAAAAAAAAAAAAAAAAAAA`

⇒ Meilleur de cas : $O(n)$

- Recherche de `CCCCCCCC` dans le mot `AAAAAAAAAAAAAAAAAAAAAAAAAAAAAA`

⇒ Cas moyen : dépend du nombre de match exacts

```
1  #include <iostream>
2  #include <string>
3  #include <libgen.h>
4
5  using namespace std;
6
7  int main(int argc, char** argv) {
8    string txt = "ACACTCCACT", mot = "ACT";
9    size_t l1 = txt.length(), l2 = mot.length();
10
11   for (size_t i=0; i < l1; i++) {
12     for (size_t j = 0; i + j < l1 && j < l2 && txt[i + j] == mot[j]; j++) {
13       if (j == l2 - 1) {
14         cout << "Found at location " << i << endl;
15       }
16     }
17   }
18   return 0;
19 }
```

Recherche du mot `COCO` dans la séquence `COCORICO`

Recherche du mot `COCO` dans la séquence `COCORICO`

C O C O R I C O

Alignement visuel de séquences
Recherche de motif
Algorithme naïf
Illustration
Algorithme KMP
Calcul du bord maximal
Extraction de motif

Recherche du mot COCO dans la séquence COCORICO

C O C O R I C O

$i \leftarrow 0$

Recherche du mot `COCO` dans la séquence `COCORICO`

C O C O R I C O

$i \leftarrow 0$ C

Recherche du mot `COCO` dans la séquence `COCORICO`

C O C O R I C O

$i \leftarrow 0$ C O

Recherche du mot COCO dans la séquence COCORICO

C O C O R I C O

$i \leftarrow 0$ C O C

Recherche du mot COCO dans la séquence COCORICO

	C	O	C	O	R	I	C	O
$i \leftarrow 0$	C	O	C	O				

Recherche du mot COCO dans la séquence COCORICO

	C	O	C	O	R	I	C	O
$i \leftarrow 0$	C	O	C	O				
$i \leftarrow 1$								

Recherche du mot COCO dans la séquence COCORICO

	C	O	C	O	R	I	C	O
$i \leftarrow 0$	C	O	C	O				
$i \leftarrow 1$		C						

Recherche du mot COCO dans la séquence COCORICO

	C	O	C	O	R	I	C	O
$i \leftarrow 0$	C	O	C	O				
$i \leftarrow 1$		C						
$i \leftarrow 2$								

Recherche du mot `COCO` dans la séquence `COCORICO`

	C	O	C	O	R	I	C	O
$i \leftarrow 0$	C	O	C	O				
$i \leftarrow 1$		C						
$i \leftarrow 2$			C					

Recherche du mot COCO dans la séquence COCORICO

	C	O	C	O	R	I	C	O
$i \leftarrow 0$	C	O	C	O				
$i \leftarrow 1$		C						
$i \leftarrow 2$			C	O				

Recherche du mot COCO dans la séquence COCORICO

	C	O	C	O	R	I	C	O
$i \leftarrow 0$	C	O	C	O				
$i \leftarrow 1$		C						
$i \leftarrow 2$			C	O	C			

Recherche du mot COCO dans la séquence COCORICO

	C	O	C	O	R	I	C	O
$i \leftarrow 0$	C	O	C	O				
$i \leftarrow 1$		C						
$i \leftarrow 2$			C	O	C			
$i \leftarrow 3$								

Recherche du mot COCO dans la séquence COCORICO

	C	O	C	O	R	I	C	O
$i \leftarrow 0$	C	O	C	O				
$i \leftarrow 1$		C						
$i \leftarrow 2$			C	O	C			
$i \leftarrow 3$				C				

Recherche du mot COCO dans la séquence COCORICO

	C	O	C	O	R	I	C	O
$i \leftarrow 0$	C	O	C	O				
$i \leftarrow 1$		C						
$i \leftarrow 2$			C	O	C			
$i \leftarrow 3$				C				
$i \leftarrow 4$								

Recherche du mot COCO dans la séquence COCORICO

	C	O	C	O	R	I	C	O
$i \leftarrow 0$	C	O	C	O				
$i \leftarrow 1$		C						
$i \leftarrow 2$			C	O	C			
$i \leftarrow 3$				C				
$i \leftarrow 4$					C			

Recherche du mot COCO dans la séquence COCORICO

	C	O	C	O	R	I	C	O
$i \leftarrow 0$	C	O	C	O				
$i \leftarrow 1$		C						
$i \leftarrow 2$			C	O	C			
$i \leftarrow 3$				C				
$i \leftarrow 4$					C			
$i \leftarrow 5$								

Recherche du mot COCO dans la séquence COCORICO

	C	O	C	O	R	I	C	O
$i \leftarrow 0$	C	O	C	O				
$i \leftarrow 1$		C						
$i \leftarrow 2$			C	O	C			
$i \leftarrow 3$				C				
$i \leftarrow 4$					C			
$i \leftarrow 5$						C		

Recherche du mot COCO dans la séquence COCORICO

	C	O	C	O	R	I	C	O
$i \leftarrow 0$	C	O	C	O				
$i \leftarrow 1$		C						
$i \leftarrow 2$			C	O	C			
$i \leftarrow 3$				C				
$i \leftarrow 4$					C			
$i \leftarrow 5$						C		
$i \leftarrow 6$								

Recherche du mot `COCO` dans la séquence `COCORICO`

	C	O	C	O	R	I	C	O
$i \leftarrow 0$	C	O	C	O				
$i \leftarrow 1$		C						
$i \leftarrow 2$			C	O	C			
$i \leftarrow 3$				C				
$i \leftarrow 4$					C			
$i \leftarrow 5$						C		
$i \leftarrow 6$							C	

Recherche du mot COCO dans la séquence COCORICO

	C	O	C	O	R	I	C	O
$i \leftarrow 0$	C	O	C	O				
$i \leftarrow 1$		C						
$i \leftarrow 2$			C	O	C			
$i \leftarrow 3$				C				
$i \leftarrow 4$					C			
$i \leftarrow 5$						C		
$i \leftarrow 6$							C	O

Recherche du mot COCO dans la séquence COCORICO

	C	O	C	O	R	I	C	O
$i \leftarrow 0$	C	O	C	O				
$i \leftarrow 1$		C						
$i \leftarrow 2$			C	O	C			
$i \leftarrow 3$				C				
$i \leftarrow 4$					C			
$i \leftarrow 5$						C		
$i \leftarrow 6$							C	O
$i \leftarrow 7$								

Recherche du mot COCO dans la séquence COCORICO

	C	O	C	O	R	I	C	O
$i \leftarrow 0$	C	O	C	O				
$i \leftarrow 1$		C						
$i \leftarrow 2$			C	O	C			
$i \leftarrow 3$				C				
$i \leftarrow 4$					C			
$i \leftarrow 5$						C		
$i \leftarrow 6$							C	O
$i \leftarrow 7$								C

Principe Éviter de refaire des comparaisons superflues

Recherche de motif
Algorithme KMP

Principe Éviter de refaire des comparaisons superflues

- ⇒ Sauter les itérations ou aucune correspondance n'est possible
- ⇒ Une correspondance à l'itération suivante chevauchera partiellement la correspondance courante

HLIN608 – *Algorithmique du texte.*

Alban MANCHERON

Alignement visuel de séquences

Recherche de motif

Algorithme naïf

Illustration

Algorithme KMP

Calcul du bord maximal

Extraction de motif

Principe Éviter de refaire des comparaisons superflues

- ⇒ Sauter les itérations ou aucune correspondance n'est possible
- ⇒ Une correspondance à l'itération suivante chevauchera partiellement la correspondance courante

KNUTH-MORRIS-PRATT

Principe Éviter de refaire des comparaisons superflues

- ⇒ Sauter les itérations ou aucune correspondance n'est possible
- ⇒ Une correspondance à l'itération suivante chevauchera partiellement la correspondance courante

KNUTH-MORRIS-PRATT

	C	O	C	O	R	I	C	O
$i \leftarrow 0$	C	O	C	O				
$i \leftarrow 1$		C						
$i \leftarrow 2$			C	O	C			
$i \leftarrow 3$				C				
$i \leftarrow 4$					C			
$i \leftarrow 5$						C		
$i \leftarrow 6$							C	O
$i \leftarrow 7$								C

Principe Éviter de refaire des comparaisons superflues

- ⇒ Sauter les itérations ou aucune correspondance n'est possible
- ⇒ Une correspondance à l'itération suivante chevauchera partiellement la correspondance courante

KNUTH-MORRIS-PRATT

	C	O	C	O	R	I	C	O
$i \leftarrow 0$	C	O	C	O				
~~$i \leftarrow 1$~~		~~C~~						
$i \leftarrow 2$			C	O	C			
$i \leftarrow 3$				C				
$i \leftarrow 4$					C			
$i \leftarrow 5$						C		
$i \leftarrow 6$							C	O
$i \leftarrow 7$								C

Principe Éviter de refaire des comparaisons superflues

- ⇒ Sauter les itérations ou aucune correspondance n'est possible
- ⇒ Une correspondance à l'itération suivante chevauchera partiellement la correspondance courante

KNUTH-MORRIS-PRATT

	C	O	C	O	R	I	C	O
$i \leftarrow 0$	C	O	C	O				
~~$i \leftarrow 1$~~		~~C~~						
$i \leftarrow 2$			C	O	C			
~~$i \leftarrow 3$~~				~~C~~				
$i \leftarrow 4$					C			
$i \leftarrow 5$						C		
$i \leftarrow 6$							C	O
$i \leftarrow 7$								C

Principe Éviter de refaire des comparaisons superflues

- ⇒ Sauter les itérations ou aucune correspondance n'est possible
- ⇒ Une correspondance à l'itération suivante chevauchera partiellement la correspondance courante

KNUTH-MORRIS-PRATT

	C	O	C	O	R	I	C	O
$i \leftarrow 0$	C	O	C	O				
~~$i \leftarrow 1$~~		~~C~~						
$i \leftarrow 2$			C	O	C			
~~$i \leftarrow 3$~~				~~C~~				
~~$i \leftarrow 4$~~					~~C~~			
$i \leftarrow 5$						C		
$i \leftarrow 6$							C	O
$i \leftarrow 7$								C

Principe Éviter de refaire des comparaisons superflues

- ⇒ Sauter les itérations ou aucune correspondance n'est possible
- ⇒ Une correspondance à l'itération suivante chevauchera partiellement la correspondance courante

KNUTH-MORRIS-PRATT

	C	O	C	O	R	I	C	O
$i \leftarrow 0$	C	O	C	O				
~~$i \leftarrow 1$~~		~~C~~						
$i \leftarrow 2$			C	O	C			
~~$i \leftarrow 3$~~				~~C~~				
~~$i \leftarrow 4$~~					~~C~~			
$i \leftarrow 5$						C		
$i \leftarrow 6$							C	O
~~$i \leftarrow 7$~~								~~C~~

Bord maximal [*overlap*]

Le chevauchement de deux mots x et y est la taille du plus long suffixe propre de x qui est également préfixe de y.

KNUTH-MORRIS-PRATT

	C	O	C	O	R	I	C	O
$i \leftarrow 0$	C	O	C	O				
~~$i \leftarrow 1$~~		~~C~~						
$i \leftarrow 2$			C	O	C			
~~$i \leftarrow 3$~~				~~C~~				
~~$i \leftarrow 4$~~					~~C~~			
$i \leftarrow 5$						C		
$i \leftarrow 6$							C	O
~~$i \leftarrow 7$~~								~~C~~

Bord maximal [*overlap*]

Le chevauchement de deux mots x et y est la taille du plus long suffixe propre de x qui est également préfixe de y.

- Si le préfixe $m[1..\ell]$ correspond à $s[i..i+\ell-1]$ et que $m[\ell+1] \neq s[i+\ell]$, où aura lieu (au mieux) la prochaine correspondance dans $s[i..|s|]$?

KNUTH-MORRIS-PRATT

	C	O	C	O	R	I	C	O
$i \leftarrow 0$	C	O	C	O				
~~$i \leftarrow 1$~~		~~C~~						
$i \leftarrow 2$			C	O	C			
~~$i \leftarrow 3$~~				~~C~~				
~~$i \leftarrow 4$~~					~~C~~			
$i \leftarrow 5$						C		
$i \leftarrow 6$							C	O
~~$i \leftarrow 7$~~								~~C~~

Le chevauchement maximal (overlap)

Le chevauchement de deux mots x et y est la taille du plus long suffixe propre de x qui est également préfixe de y.

- Si le préfixe $m[1..\ell]$ correspond à $s[i..i+\ell-1]$ et que $m[\ell+1] \neq s[i+\ell]$, où aura lieu (au mieux) la prochaine correspondance dans $s[i..|s|]$?

KNUTH-MORRIS-PRATT

	C	O	C	O	R	I	C	O
$i \leftarrow 0$	C	O	C	O				
~~$i \leftarrow 1$~~		~~C~~						
$i \leftarrow 2$			C	O	C			
~~$i \leftarrow 3$~~				~~C~~				
~~$i \leftarrow 4$~~					~~C~~			
$i \leftarrow 5$						C		
$i \leftarrow 6$							C	O
~~$i \leftarrow 7$~~								~~C~~

Le chevauchement maximal

Le chevauchement de deux mots x et y est la taille du plus long suffixe propre de x qui est également préfixe de y.

- Si le préfixe $m[1..\ell]$ correspond à $s[i..i+\ell-1]$ et que $m[\ell+1] \neq s[i+\ell]$, où aura lieu (au mieux) la prochaine correspondance dans $s[i..|s|]$?

KNUTH-MORRIS-PRATT

	C	O	C	O	R	I	C	O
$i \leftarrow 0$	C	O	C	O				
~~$i \leftarrow 1$~~		~~C~~						
$i \leftarrow 2$			C	O	C			
~~$i \leftarrow 3$~~				~~C~~				
~~$i \leftarrow 4$~~					~~C~~			
$i \leftarrow 5$						C		
$i \leftarrow 6$							C	O
~~$i \leftarrow 7$~~								~~C~~

- Si le préfixe $m[1..\ell]$ correspond à $s[i..i+\ell-1]$ et que $m[\ell+1] \neq s[i+\ell]$, où aura lieu (au mieux) la prochaine correspondance dans $s[i..|s|]$?

KNUTH-MORRIS-PRATT

	C	O	C	O	R	I	C	O
$i \leftarrow 0$	C	O	C	O				
~~$i \leftarrow 1$~~		~~C~~						
$i \leftarrow 2$			C	O	C			
~~$i \leftarrow 3$~~				~~C~~				
~~$i \leftarrow 4$~~					~~C~~			
$i \leftarrow 5$						C		
$i \leftarrow 6$							C	O
~~$i \leftarrow 7$~~								~~C~~

- Si le préfixe $m[1..\ell]$ correspond à $s[i..i+\ell-1]$ et que $m[\ell+1] \neq s[i+\ell]$, où aura lieu (au mieux) la prochaine correspondance dans $s[i..|s|]$?
  - $\Rightarrow$ à la position $i + \ell - overlap(m[1..\ell], m[1..\ell])$

KNUTH-MORRIS-PRATT

	C	O	C	O	R	I	C	O
$i \leftarrow 0$	C	O	C	O				
~~$i \leftarrow 1$~~	~~C~~							
$i \leftarrow 2$			C	O	C			
~~$i \leftarrow 3$~~				~~C~~				
~~$i \leftarrow 4$~~					~~C~~			
$i \leftarrow 5$						C		
$i \leftarrow 6$							C	O
~~$i \leftarrow 7$~~								~~C~~

- Calcul dynamique

HLIN608 – *Algorithmique du texte.*

Alban MANCHERON

Alignement visuel de séquences

Recherche de motif

Algorithme naïf

Illustration

Algorithme KMP

Calcul du bord maximal

Extraction de motif

- Calcul dynamique

Calcul dynamique du chevauchement

```
int Overlap(const string &seq) {
  bool ok = false;
  size_t lg, l = lg = seq.length();
  while (!ok && l--) {
    ok = true;
    for (size_t i = 0; ok && (i < l); i++) {
      ok = (seq[i] == seq[lg-l+i]);
    }
  }
  return l;
}
```

- Calcul dynamique

Calcul dynamique du chevauchement

```
int Overlap(const string &seq) {
  bool ok = false;
  size_t lg, l = lg = seq.length();
  while (!ok && l--) {
    ok = true;
    for (size_t i = 0; ok && (i < l); i++) {
      ok = (seq[i] == seq[lg-l+i]);
    }
  }
  return l;
}
```

- Calcul dynamique

Calcul dynamique du chevauchement

```
int Overlap(const string &seq) {
  bool ok = false;
  size_t lg, l = lg = seq.length();
  while (!ok && l--) {
    ok = true;
    for (size_t i = 0; ok && (i < l); i++) {
      ok = (seq[i] == seq[lg-l+i]);
    }
  }
  return l;
}
```

- Calcul dynamique

Calcul dynamique du chevauchement

```
1  int Overlap(const string &seq) {
2    bool ok = false;
3    size_t lg, l = lg = seq.length();
4    while (!ok && l--) {
5      ok = true;
6      for (size_t i = 0; ok && (i < l); i++) {
7        ok = (seq[i] == seq[lg-l+i]);
8      }
9    }
10   return l;
11 }
```

- Calcul dynamique

Calcul dynamique du chevauchement

```
int Overlap(const string &seq) {
  bool ok = false;
  size_t lg, l = lg = seq.length();
  while (!ok && l--) {
    ok = true;
    for (size_t i = 0; ok && (i < l); i++) {
      ok = (seq[i] == seq[lg-l+i]);
    }
  }
  return l;
}
```

- Calcul dynamique

Calcul dynamique du chevauchement

```
int Overlap(const string &seq) {
  bool ok = false;
  size_t lg, l = lg = seq.length();
  while (!ok && l--) {
    ok = true;
    for (size_t i = 0; ok && (i < l); i++) {
      ok = (seq[i] == seq[lg-l+i]);
    }
  }
  return l;
}
```

⇒ redondance ! ! !

⇒ Complexité temporelle de la recherche en $O\left(n\,m^2\right)$

- Pré-traitement simple

Calcul des chevauchements de tous les préfixes

```
int* Overlap(const string &seq) {
  size_t lg = seq.length();
  size_t* overlap = new int[lg + 1];
  overlap[0] = (size_t) -1;
  for (size_t i = 1; i < lg; i++) {
    overlap[i] = Overlap(seq.substr(0, i));
  }
  return overlap;
}
```

- Pré-traitement simple

Calcul des chevauchements de tous les préfixes

```
int* Overlap(const string &seq) {
  size_t lg = seq.length();
  size_t* overlap = new int[lg + 1];
  overlap[0] = (size_t) -1;
  for (size_t i = 1; i < lg; i++) {
    overlap[i] = Overlap(seq.substr(0, i));
  }
  return overlap;
}
```

$\Rightarrow$ Complexité temporelle en $O\left(n + m^3\right)$

- Pré-traitement « malin »

Calcul efficace des chevauchements de tous les préfixes

```
int* Overlap(const string &seq) {
  size_t lg = seq.length();
  size_t* overlap = new int[lg + 1];
  overlap[0] = (size_t) -1;
  for (size_t i = 0; i < lg; i++) {
    overlap[i + 1] = overlap[i] + 1;
    while (overlap[i + 1] > 0 &&
           seq[i] != seq[overlap[i + 1] - 1]) {
      overlap[i + 1] = overlap[overlap[i + 1] - 1] + 1;
    }
  }
  return overlap;
}
```

- Pré-traitement « malin »

Calcul efficace des chevauchements de tous les préfixes

```
int* Overlap(const string &seq) {
  size_t lg = seq.length();
  size_t* overlap = new int[lg + 1];
  overlap[0] = (size_t) -1;
  for (size_t i = 0; i < lg; i++) {
    overlap[i + 1] = overlap[i] + 1;
    while (overlap[i + 1] > 0 &&
           seq[i] != seq[overlap[i + 1] - 1]) {
      overlap[i + 1] = overlap[overlap[i + 1] - 1] + 1;
    }
  }
  return overlap;
}
```

$\Rightarrow$ Complexité temporelle en $O(n+m)$

- Pré-traitement « malin »

Calcul efficace des chevauchements de tous les préfixes

```
int* Overlap(const string &seq) {
  size_t lg = seq.length();
  size_t* overlap = new int[lg + 1];
  overlap[0] = (size_t) -1;
  for (size_t i = 0; i < lg; i++) {
    overlap[i + 1] = overlap[i] + 1;
    while (overlap[i + 1] > 0 &&
           seq[i] != seq[overlap[i + 1] - 1]) {
      overlap[i + 1] = overlap[overlap[i + 1] - 1] + 1;
    }
  }
  return overlap;
}
```

i	0	1	2	3	4
seq	C	O	C	O	\0
$overlap$					

- Pré-traitement « malin »

Calcul efficace des chevauchements de tous les préfixes

```
1  int* Overlap(const string &seq) {
2    size_t lg = seq.length();
3    size_t* overlap = new int[lg + 1];
4    overlap[0] = (size_t) -1;
5    for (size_t i = 0; i < lg; i++) {
6      overlap[i + 1] = overlap[i] + 1;
7      while (overlap[i + 1] > 0 &&
8             seq[i] != seq[overlap[i + 1] - 1]) {
9        overlap[i + 1] = overlap[overlap[i + 1] - 1] + 1;
10     }
11   }
12   return overlap;
13 }
```

i	0	1	2	3	4
seq	C	O	C	O	\0
$overlap$	-1				

ligne 4

- Pré-traitement « malin »

Calcul efficace des chevauchements de tous les préfixes

```
1  int* Overlap(const string &seq) {
2    size_t lg = seq.length();
3    size_t* overlap = new int[lg + 1];
4    overlap[0] = (size_t) -1;
5    for (size_t i = 0; i < lg; i++) {
6      overlap[i + 1] = overlap[i] + 1;
7      while (overlap[i + 1] > 0 &&
8             seq[i] != seq[overlap[i + 1] - 1]) {
9        overlap[i + 1] = overlap[overlap[i + 1] - 1] + 1;
10     }
11   }
12   return overlap;
13 }
```

i	0	1	2	3	4
seq	C	O	C	O	\0
$overlap$	-1				

ligne 5

- Pré-traitement « malin »

Calcul efficace des chevauchements de tous les préfixes

```
1  int* Overlap(const string &seq) {
2    size_t lg = seq.length();
3    size_t* overlap = new int[lg + 1];
4    overlap[0] = (size_t) -1;
5    for (size_t i = 0; i < lg; i++) {
6      overlap[i + 1] = overlap[i] + 1;
7      while (overlap[i + 1] > 0 &&
8             seq[i] != seq[overlap[i + 1] - 1]) {
9        overlap[i + 1] = overlap[overlap[i + 1] - 1] + 1;
10     }
11   }
12   return overlap;
13 }
```

i	0	1	2	3	4
seq	C	O	C	O	\0
$overlap$	-1	0			

ligne 6

- Pré-traitement « malin »

Calcul efficace des chevauchements de tous les préfixes

```
1  int* Overlap(const string &seq) {
2    size_t lg = seq.length();
3    size_t* overlap = new int[lg + 1];
4    overlap[0] = (size_t) -1;
5    for (size_t i = 0; i < lg; i++) {
6      overlap[i + 1] = overlap[i] + 1;
7      while (overlap[i + 1] > 0 &&
8             seq[i] != seq[overlap[i + 1] - 1]) {
9        overlap[i + 1] = overlap[overlap[i + 1] - 1] + 1;
10     }
11   }
12   return overlap;
13 }
```

i	0	1	2	3	4
seq	C	O	C	O	\0
$overlap$	-1	$\boxed{0}$			

$\boxed{\text{ligne 7}}$

- Pré-traitement « malin »

Calcul efficace des chevauchements de tous les préfixes

```
1  int* Overlap(const string &seq) {
2    size_t lg = seq.length();
3    size_t* overlap = new int[lg + 1];
4    overlap[0] = (size_t) -1;
5    for (size_t i = 0; i < lg; i++) {
6      overlap[i + 1] = overlap[i] + 1;
7      while (overlap[i + 1] > 0 &&
8             seq[i] != seq[overlap[i + 1] - 1]) {
9        overlap[i + 1] = overlap[overlap[i + 1] - 1] + 1;
10     }
11   }
12   return overlap;
13 }
```

i	0	1	2	3	4
seq	C	O	C	O	\0
$overlap$	-1	0			

ligne 5

- Pré-traitement « malin »

Calcul efficace des chevauchements de tous les préfixes

```
1  int* Overlap(const string &seq) {
2    size_t lg = seq.length();
3    size_t* overlap = new int[lg + 1];
4    overlap[0] = (size_t) -1;
5    for (size_t i = 0; i < lg; i++) {
6      overlap[i + 1] = overlap[i] + 1;
7      while (overlap[i + 1] > 0 &&
8             seq[i] != seq[overlap[i + 1] - 1]) {
9        overlap[i + 1] = overlap[overlap[i + 1] - 1] + 1;
10     }
11   }
12   return overlap;
13 }
```

i	0	1	2	3	4
seq	C	O	C	O	\0
$overlap$	-1	0	1		

ligne 6

- Pré-traitement « malin »

Calcul efficace des chevauchements de tous les préfixes

```
1  int* Overlap(const string &seq) {
2    size_t lg = seq.length();
3    size_t* overlap = new int[lg + 1];
4    overlap[0] = (size_t) -1;
5    for (size_t i = 0; i < lg; i++) {
6      overlap[i + 1] = overlap[i] + 1;
7      while (overlap[i + 1] > 0 &&
8             seq[i] != seq[overlap[i + 1] - 1]) {
9        overlap[i + 1] = overlap[overlap[i + 1] - 1] + 1;
10     }
11   }
12   return overlap;
13 }
```

i	0	1	2	3	4
seq	C	O	C	O	\0
$overlap$	-1	0	$\boxed{1}$		

$\boxed{\text{ligne 7}}$

- Pré-traitement « malin »

Calcul efficace des chevauchements de tous les préfixes

```
1  int* Overlap(const string &seq) {
2    size_t lg = seq.length();
3    size_t* overlap = new int[lg + 1];
4    overlap[0] = (size_t) -1;
5    for (size_t i = 0; i < lg; i++) {
6      overlap[i + 1] = overlap[i] + 1;
7      while (overlap[i + 1] > 0 &&
8             seq[i] != seq[overlap[i + 1] - 1]) {
9        overlap[i + 1] = overlap[overlap[i + 1] - 1] + 1;
10     }
11   }
12   return overlap;
13 }
```

i	0	1	2	3	4	
seq	C	O	C	O	\0	ligne 8
$overlap$	-1	0	1			

- Pré-traitement « malin »

Calcul efficace des chevauchements de tous les préfixes

```
1  int* Overlap(const string &seq) {
2    size_t lg = seq.length();
3    size_t* overlap = new int[lg + 1];
4    overlap[0] = (size_t) -1;
5    for (size_t i = 0; i < lg; i++) {
6      overlap[i + 1] = overlap[i] + 1;
7      while (overlap[i + 1] > 0 &&
8             seq[i] != seq[overlap[i + 1] - 1]) {
9        overlap[i + 1] = overlap[overlap[i + 1] - 1] + 1;
10     }
11   }
12   return overlap;
13 }
```

i	0	1	2	3	4
seq	C	O	C	O	\0
$overlap$	-1	0	0		

ligne 9

- Pré-traitement « malin »

Calcul efficace des chevauchements de tous les préfixes

```
1  int* Overlap(const string &seq) {
2    size_t lg = seq.length();
3    size_t* overlap = new int[lg + 1];
4    overlap[0] = (size_t) -1;
5    for (size_t i = 0; i < lg; i++) {
6      overlap[i + 1] = overlap[i] + 1;
7      while (overlap[i + 1] > 0 &&
8             seq[i] != seq[overlap[i + 1] - 1]) {
9        overlap[i + 1] = overlap[overlap[i + 1] - 1] + 1;
10     }
11   }
12   return overlap;
13 }
```

i	0	1	2	3	4
seq	C	O	C	O	\0
$overlap$	-1	0	$\boxed{0}$		

ligne 7

- Pré-traitement « malin »

Calcul efficace des chevauchements de tous les préfixes

```
1  int* Overlap(const string &seq) {
2    size_t lg = seq.length();
3    size_t* overlap = new int[lg + 1];
4    overlap[0] = (size_t) -1;
5    for (size_t i = 0; i < lg; i++) {
6      overlap[i + 1] = overlap[i] + 1;
7      while (overlap[i + 1] > 0 &&
8             seq[i] != seq[overlap[i + 1] - 1]) {
9        overlap[i + 1] = overlap[overlap[i + 1] - 1] + 1;
10     }
11   }
12   return overlap;
13 }
```

i	0	1	2	3	4
seq	C	O	C	O	\0
$overlap$	-1	0	0		

ligne 5

- Pré-traitement « malin »

Calcul efficace des chevauchements de tous les préfixes

```
1  int* Overlap(const string &seq) {
2    size_t lg = seq.length();
3    size_t* overlap = new int[lg + 1];
4    overlap[0] = (size_t) -1;
5    for (size_t i = 0; i < lg; i++) {
6      overlap[i + 1] = overlap[i] + 1;
7      while (overlap[i + 1] > 0 &&
8             seq[i] != seq[overlap[i + 1] - 1]) {
9        overlap[i + 1] = overlap[overlap[i + 1] - 1] + 1;
10     }
11   }
12   return overlap;
13 }
```

i	0	1	2	3	4
seq	C	O	C	O	\0
$overlap$	-1	0	0	1	

ligne 6

- Pré-traitement « malin »

Calcul efficace des chevauchements de tous les préfixes

```
1  int* Overlap(const string &seq) {
2    size_t lg = seq.length();
3    size_t* overlap = new int[lg + 1];
4    overlap[0] = (size_t) -1;
5    for (size_t i = 0; i < lg; i++) {
6      overlap[i + 1] = overlap[i] + 1;
7      while (overlap[i + 1] > 0 &&
8             seq[i] != seq[overlap[i + 1] - 1]) {
9        overlap[i + 1] = overlap[overlap[i + 1] - 1] + 1;
10     }
11   }
12   return overlap;
13 }
```

i	0	1	2	3	4
seq	C	O	C	O	\0
$overlap$	-1	0	0	$\boxed{1}$	

$\boxed{\text{ligne 7}}$

- Pré-traitement « malin »

Calcul efficace des chevauchements de tous les préfixes

```
1  int* Overlap(const string &seq) {
2    size_t lg = seq.length();
3    size_t* overlap = new int[lg + 1];
4    overlap[0] = (size_t) -1;
5    for (size_t i = 0; i < lg; i++) {
6      overlap[i + 1] = overlap[i] + 1;
7      while (overlap[i + 1] > 0 &&
8             seq[i] != seq[overlap[i + 1] - 1]) {
9        overlap[i + 1] = overlap[overlap[i + 1] - 1] + 1;
10     }
11   }
12   return overlap;
13 }
```

i	0	1	2	3	4
seq	C	O	C	O	\0
$overlap$	-1	0	0	1	

ligne 8

- Pré-traitement « malin »

Calcul efficace des chevauchements de tous les préfixes

```
1  int* Overlap(const string &seq) {
2    size_t lg = seq.length();
3    size_t* overlap = new int[lg + 1];
4    overlap[0] = (size_t) -1;
5    for (size_t i = 0; i < lg; i++) {
6      overlap[i + 1] = overlap[i] + 1;
7      while (overlap[i + 1] > 0 &&
8             seq[i] != seq[overlap[i + 1] - 1]) {
9        overlap[i + 1] = overlap[overlap[i + 1] - 1] + 1;
10     }
11   }
12   return overlap;
13 }
```

i	0	1	2	3	4
seq	C	O	C	O	\0
$overlap$	-1	0	0	1	

ligne 5

- Pré-traitement « malin »

Calcul efficace des chevauchements de tous les préfixes

```
1  int* Overlap(const string &seq) {
2    size_t lg = seq.length();
3    size_t* overlap = new int[lg + 1];
4    overlap[0] = (size_t) -1;
5    for (size_t i = 0; i < lg; i++) {
6      overlap[i + 1] = overlap[i] + 1;
7      while (overlap[i + 1] > 0 &&
8             seq[i] != seq[overlap[i + 1] - 1]) {
9        overlap[i + 1] = overlap[overlap[i + 1] - 1] + 1;
10     }
11   }
12   return overlap;
13 }
```

i	0	1	2	3	4
seq	C	O	C	O	\0
$overlap$	-1	0	0	$\boxed{1}$	$\boxed{2}$

$\boxed{\text{ligne 6}}$

- Pré-traitement « malin »

Calcul efficace des chevauchements de tous les préfixes

```
1  int* Overlap(const string &seq) {
2    size_t lg = seq.length();
3    size_t* overlap = new int[lg + 1];
4    overlap[0] = (size_t) -1;
5    for (size_t i = 0; i < lg; i++) {
6      overlap[i + 1] = overlap[i] + 1;
7      while (overlap[i + 1] > 0 &&
8             seq[i] != seq[overlap[i + 1] - 1]) {
9        overlap[i + 1] = overlap[overlap[i + 1] - 1] + 1;
10     }
11   }
12   return overlap;
13 }
```

i	0	1	2	3	4
seq	C	O	C	O	\0
$overlap$	-1	0	0	1	$\boxed{2}$

ligne 7

- Pré-traitement « malin »

Calcul efficace des chevauchements de tous les préfixes

```
1  int* Overlap(const string &seq) {
2    size_t lg = seq.length();
3    size_t* overlap = new int[lg + 1];
4    overlap[0] = (size_t) -1;
5    for (size_t i = 0; i < lg; i++) {
6      overlap[i + 1] = overlap[i] + 1;
7      while (overlap[i + 1] > 0 &&
8             seq[i] != seq[overlap[i + 1] - 1]) {
9        overlap[i + 1] = overlap[overlap[i + 1] - 1] + 1;
10     }
11   }
12   return overlap;
13 }
```

i	0	1	2	3	4
seq	C	O	C	O	\0
$overlap$	-1	0	0	1	2

ligne 8

- Pré-traitement « malin »

Calcul efficace des chevauchements de tous les préfixes

```
1  int* Overlap(const string &seq) {
2    size_t lg = seq.length();
3    size_t* overlap = new int[lg + 1];
4    overlap[0] = (size_t) -1;
5    for (size_t i = 0; i < lg; i++) {
6      overlap[i + 1] = overlap[i] + 1;
7      while (overlap[i + 1] > 0 &&
8             seq[i] != seq[overlap[i + 1] - 1]) {
9        overlap[i + 1] = overlap[overlap[i + 1] - 1] + 1;
10     }
11   }
12   return overlap;
13 }
```

i	0	1	2	3	4
seq	C	O	C	O	\0
$overlap$	-1	0	0	1	2

ligne 12

- Pré-traitement « malin »

Calcul efficace des chevauchements de tous les préfixes

```
1  int* Overlap(const string &seq) {
2    size_t lg = seq.length();
3    size_t* overlap = new int[lg + 1];
4    overlap[0] = (size_t) -1;
5    for (size_t i = 0; i < lg; i++) {
6      overlap[i + 1] = overlap[i] + 1;
7      while (overlap[i + 1] > 0 &&
8             seq[i] != seq[overlap[i + 1] - 1]) {
9        overlap[i + 1] = overlap[overlap[i + 1] - 1] + 1;
10     }
11   }
12   return overlap;
13 }
```

$Overlap(seq) = \ell$

seq

i	0	1	2	3	4
seq	C	O	C	O	\0
overlap	-1	0	0	1	2

ligne 12

- Pré-traitement « malin »

Calcul efficace des chevauchements de tous les préfixes

```
1  int* Overlap(const string &seq) {
2    size_t lg = seq.length();
3    size_t* overlap = new int[lg + 1];
4    overlap[0] = (size_t) -1;
5    for (size_t i = 0; i < lg; i++) {
6      overlap[i + 1] = overlap[i] + 1;
7      while (overlap[i + 1] > 0 &&
8             seq[i] != seq[overlap[i + 1] - 1]) {
9        overlap[i + 1] = overlap[overlap[i + 1] - 1] + 1;
10     }
11   }
12   return overlap;
13 }
```

$Overlap(seq.\alpha)$?

i	0	1	2	3	4
seq	C	O	C	O	\0
$overlap$	-1	0	0	1	2

ligne 12

- Pré-traitement « malin »

Calcul efficace des chevauchements de tous les préfixes

```
1  int* Overlap(const string &seq) {
2    size_t lg = seq.length();
3    size_t* overlap = new int[lg + 1];
4    overlap[0] = (size_t) -1;
5    for (size_t i = 0; i < lg; i++) {
6      overlap[i + 1] = overlap[i] + 1;
7      while (overlap[i + 1] > 0 &&
8             seq[i] != seq[overlap[i + 1] - 1]) {
9        overlap[i + 1] = overlap[overlap[i + 1] - 1] + 1;
10     }
11   }
12   return overlap;
13 }
```

$Overlap(seq.\alpha) > \ell + 1?$

seq ?

i	0	1	2	3	4
seq	C	O	C	O	\0
overlap	-1	0	0	1	2

ligne 12

- Pré-traitement « malin »

Calcul efficace des chevauchements de tous les préfixes

```
1  int* Overlap(const string &seq) {
2    size_t lg = seq.length();
3    size_t* overlap = new int[lg + 1];
4    overlap[0] = (size_t) -1;
5    for (size_t i = 0; i < lg; i++) {
6      overlap[i + 1] = overlap[i] + 1;
7      while (overlap[i + 1] > 0 &&
8             seq[i] != seq[overlap[i + 1] - 1]) {
9        overlap[i + 1] = overlap[overlap[i + 1] - 1] + 1;
10     }
11   }
12   return overlap;
13 }
```

$Overlap(seq.\alpha) > \ell + 1?$

i	0	1	2	3	4
seq	C	O	C	O	\0
$overlap$	-1	0	0	1	2

ligne 12

- Pré-traitement « malin »

Calcul efficace des chevauchements de tous les préfixes

```
1  int* Overlap(const string &seq) {
2    size_t lg = seq.length();
3    size_t* overlap = new int[lg + 1];
4    overlap[0] = (size_t) -1;
5    for (size_t i = 0; i < lg; i++) {
6      overlap[i + 1] = overlap[i] + 1;
7      while (overlap[i + 1] > 0 &&
8             seq[i] != seq[overlap[i + 1] - 1]) {
9        overlap[i + 1] = overlap[overlap[i + 1] - 1] + 1;
10     }
11   }
12   return overlap;
13 }
```

$Overlap(seq.\alpha) > \ell + 1?$

seq ?

i	0	1	2	3	4
seq	C	O	C	O	\0
overlap	-1	0	0	1	2

ligne 12

- Pré-traitement « malin »

Calcul efficace des chevauchements de tous les préfixes

```
1  int* Overlap(const string &seq) {
2    size_t lg = seq.length();
3    size_t* overlap = new int[lg + 1];
4    overlap[0] = (size_t) -1;
5    for (size_t i = 0; i < lg; i++) {
6      overlap[i + 1] = overlap[i] + 1;
7      while (overlap[i + 1] > 0 &&
8             seq[i] != seq[overlap[i + 1] - 1]) {
9        overlap[i + 1] = overlap[overlap[i + 1] - 1] + 1;
10     }
11   }
12   return overlap;
13 }
```

i	0	1	2	3	4
seq	C	O	C	O	\0
$overlap$	-1	0	0	1	2

ligne 12

- Pré-traitement « malin »

Calcul efficace des chevauchements de tous les préfixes

```
1  int* Overlap(const string &seq) {
2    size_t lg = seq.length();
3    size_t* overlap = new int[lg + 1];
4    overlap[0] = (size_t) -1;
5    for (size_t i = 0; i < lg; i++) {
6      overlap[i + 1] = overlap[i] + 1;
7      while (overlap[i + 1] > 0 &&
8             seq[i] != seq[overlap[i + 1] - 1]) {
9        overlap[i + 1] = overlap[overlap[i + 1] - 1] + 1;
10     }
11   }
12   return overlap;
13 }
```

i	0	1	2	3	4
seq	C	O	C	O	\0
$overlap$	-1	0	0	1	2

ligne 12

- Pré-traitement « malin »

Calcul efficace des chevauchements de tous les préfixes

```
1  int* Overlap(const string &seq) {
2    size_t lg = seq.length();
3    size_t* overlap = new int[lg + 1];
4    overlap[0] = (size_t) -1;
5    for (size_t i = 0; i < lg; i++) {
6      overlap[i + 1] = overlap[i] + 1;
7      while (overlap[i + 1] > 0 &&
8             seq[i] != seq[overlap[i + 1] - 1]) {
9        overlap[i + 1] = overlap[overlap[i + 1] - 1] + 1;
10     }
11   }
12   return overlap;
13 }
```

$Overlap(seq.\alpha) > \ell + 1?$

seq ?

i	0	1	2	3	4
seq	C	O	C	O	\0
overlap	-1	0	0	1	2

ligne 12

- Pré-traitement « malin »

Calcul efficace des chevauchements de tous les préfixes

```
1  int* Overlap(const string &seq) {
2    size_t lg = seq.length();
3    size_t* overlap = new int[lg+1];
4    overlap[0] = (size_t) -1;
5    for (size_t i = 0; i < lg; i++) {
6      overlap[i+1] = overlap[i] + 1;
7      while (overlap[i+1] > 0 &&
8             seq[i] != seq[overlap[i+1] - 1]) {
9        overlap[i+1] = overlap[overlap[i+1] - 1] + 1;
10     }
11   }
12   return overlap;
13 }
```

i	0	1	2	3	4
seq	C	O	C	O	\0
$overlap$	-1	0	0	1	2

ligne 12

- Pré-traitement « malin »

Calcul efficace des chevauchements de tous les préfixes

```
1  int* Overlap(const string &seq) {
2    size_t lg = seq.length();
3    size_t* overlap = new int[lg + 1];
4    overlap[0] = (size_t) -1;
5    for (size_t i = 0; i < lg; i++) {
6      overlap[i + 1] = overlap[i] + 1;
7      while (overlap[i + 1] > 0 &&
8             seq[i] != seq[overlap[i + 1] - 1]) {
9        overlap[i + 1] = overlap[overlap[i + 1] - 1] + 1;
10     }
11   }
12   return overlap;
13 }
```

i	0	1	2	3	4
seq	C	O	C	O	\0
$overlap$	-1	0	0	1	2

ligne 12

- Pré-traitement « malin »

Calcul efficace des chevauchements de tous les préfixes

```
1  int* Overlap(const string &seq) {
2    size_t lg = seq.length();
3    size_t* overlap = new[lg + 1];
4    overlap[0] = (size_t) -1;
5    for (size_t i = 0; i < lg; i++) {
6      overlap[i + 1] = overlap[i] + 1;
7      while (overlap[i + 1] > 0 &&
8             seq[i] != seq[overlap[i + 1] - 1]) {
9        overlap[i + 1] = overlap[overlap[i + 1] - 1] + 1;
10     }
11   }
12   return overlap;
13 }
```

$Overlap(seq.\alpha) > \ell + 1?$

seq

Impossible

i	0	1	2	3	4
seq	C	O	C	O	\0
overlap	-1	0	0	1	2

ligne 12

- Pré-traitement « malin »

Calcul efficace des chevauchements de tous les préfixes

```
1  int* Overlap(const string &seq) {
2    size_t lg = seq.length();
3    size_t* overlap = new int[lg + 1];
4    overlap[0] = (size_t) -1;
5    for (size_t i = 0; i < lg; i++) {
6      overlap[i + 1] = overlap[i] + 1;
7      while (overlap[i + 1] > 0 &&
8             seq[i] != seq[overlap[i + 1] - 1]) {
9        overlap[i + 1] = overlap[overlap[i + 1] - 1] + 1;
10     }
11   }
12   return overlap;
13 }
```

$Overlap(seq.\alpha) = \ell + 1?$

seq ?

i	0	1	2	3	4
seq	C	O	C	O	\0
$overlap$	-1	0	0	1	2

ligne 12

- Pré-traitement « malin »

Calcul efficace des chevauchements de tous les préfixes

```
1  int* Overlap(const string &seq) {
2    size_t lg = seq.length();
3    size_t* overlap = new int[lg + 1];
4    overlap[0] = (size_t) -1;
5    for (size_t i = 0; i < lg; i++) {
6      overlap[i + 1] = overlap[i] + 1;
7      while (overlap[i + 1] > 0 &&
8             seq[i] != seq[overlap[i + 1] - 1]) {
9        overlap[i + 1] = overlap[overlap[i + 1] - 1] + 1;
10     }
11   }
12   return overlap;
13 }
```

$Overlap(seq.\alpha) = \ell + 1?$

seq **Possible**

i	0	1	2	3	4
seq	C	O	C	O	\0
$overlap$	-1	0	0	1	2

ligne 12

- Pré-traitement « malin »

Calcul efficace des chevauchements de tous les préfixes

```
1  int* Overlap(const string &seq) {
2    size_t lg = seq.length();
3    size_t* overlap = new int[lg + 1];
4    overlap[0] = (size_t) -1;
5    for (size_t i = 0; i < lg; i++) {
6      overlap[i + 1] = overlap[i] + 1;
7      while (overlap[i + 1] > 0 &&
8             seq[i] != seq[overlap[i + 1] - 1]) {
9        overlap[i + 1] = overlap[overlap[i + 1] - 1] + 1;
10     }
11   }
12   return overlap;
13 }
```

Overlap(seq.α)?

seq

i	0	1	2	3	4
seq	C	O	C	O	\0
overlap	-1	0	0	1	2

ligne 12

- Pré-traitement « malin »

Calcul efficace des chevauchements de tous les préfixes

```
1  int* Overlap(const string &seq) {
2    size_t lg = seq.length();
3    size_t* overlap = new int[lg + 1];
4    overlap[0] = (size_t) -1;
5    for (size_t i = 0; i < lg; i++) {
6      overlap[i + 1] = overlap[i] + 1;
7      while (overlap[i + 1] > 0 &&
8             seq[i] != seq[overlap[i + 1] - 1]) {
9        overlap[i + 1] = overlap[overlap[i + 1] - 1] + 1;
10     }
11   }
12   return overlap;
13 }
```

i	0	1	2	3	4
seq	C	O	C	O	\0
$overlap$	-1	0	0	1	2

ligne 12

- Pré-traitement « malin »

Calcul efficace des chevauchements de tous les préfixes

```
1  int* Overlap(const string &seq) {
2    size_t lg = seq.length();
3    size_t* overlap = new int[lg + 1];
4    overlap[0] = (size_t) -1;
5    for (size_t i = 0; i < lg; i++) {
6      overlap[i + 1] = overlap[i] + 1;
7      while (overlap[i + 1] > 0 &&
8             seq[i] != seq[overlap[i + 1] - 1]) {
9        overlap[i + 1] = overlap[overlap[i + 1] - 1] + 1;
10     }
11   }
12   return overlap;
13 }
```

Overlap(seq.α)?

seq

i	0	1	2	3	4
seq	C	O	C	O	\0
overlap	-1	0	0	1	2

ligne 12

- Pré-traitement « malin »

Calcul efficace des chevauchements de tous les préfixes

```
1  int* Overlap(const string &seq) {
2    size_t lg = seq.length();
3    size_t* overlap = new int[lg + 1];
4    overlap[0] = (size_t) -1;
5    for (size_t i = 0; i < lg; i++) {
6      overlap[i + 1] = overlap[i] + 1;
7      while (overlap[i + 1] > 0 &&
8             seq[i] != seq[overlap[i + 1] - 1]) {
9        overlap[i + 1] = overlap[overlap[i + 1] - 1] + 1;
10     }
11   }
12   return overlap;
13 }
```

i	0	1	2	3	4
seq	C	O	C	O	\0
$overlap$	-1	0	0	1	2

ligne 12

- Pré-traitement « malin »

Calcul efficace des chevauchements de tous les préfixes

```
1  int* Overlap(const string &seq) {
2    size_t lg = seq.length();
3    size_t* overlap = new int[lg + 1];
4    overlap[0] = (size_t) -1;
5    for (size_t i = 0; i < lg; i++) {
6      overlap[i + 1] = overlap[i] + 1;
7      while (overlap[i + 1] > 0 &&
8             seq[i] != seq[overlap[i + 1] - 1]) {
9        overlap[i + 1] = overlap[overlap[i + 1] - 1] + 1;
10     }
11   }
12   return overlap;
13 }
```

i	0	1	2	3	4
seq	C	O	C	O	\0
$overlap$	-1	0	0	1	2

ligne 12

- Pré-traitement « malin »

Calcul efficace des chevauchements de tous les préfixes

```
1  int* Overlap(const string &seq) {
2    size_t lg = seq.length();
3    size_t* overlap = new int[lg + 1];
4    overlap[0] = (size_t) -1;
5    for (size_t i = 0; i < lg; i++) {
6      overlap[i + 1] = overlap[i] + 1;
7      while (overlap[i + 1] > 0 &&
8             seq[i] != seq[overlap[i + 1] - 1]) {
9        overlap[i + 1] = overlap[overlap[i + 1] - 1] + 1;
10     }
11   }
12   return overlap;
13 }
```

Overlap(seq.α)?

seq

i	0	1	2	3	4
seq	C	O	C	O	\0
overlap	-1	0	0	1	2

ligne 12

- Pré-traitement « malin »

Calcul efficace des chevauchements de tous les préfixes

```
1  int* Overlap(const string &seq) {
2    size_t lg = seq.length();
3    size_t* overlap = new int[lg + 1];
4    overlap[0] = (size_t) -1;
5    for (size_t i = 0; i < lg; i++) {
6      overlap[i + 1] = overlap[i] + 1;
7      while (overlap[i + 1] > 0 &&
8             seq[i] != seq[overlap[i + 1] - 1]) {
9        overlap[i + 1] = overlap[overlap[i + 1] - 1] + 1;
10     }
11   }
12   return overlap;
13 }
```

Overlap(seq.α)?

seq

i	0	1	2	3	4
seq	C	O	C	O	\0
$overlap$	-1	0	0	1	2

ligne 12

- Pré-traitement « malin »

Calcul efficace des chevauchements de tous les préfixes

```
1  int* Overlap(const string &seq) {
2    size_t lg = seq.length();
3    size_t* overlap = new int[lg+1];
4    overlap[0] = (size_t) -1;
5    for (size_t i = 0; i < lg; i++) {
6      overlap[i+1] = overlap[i] + 1;
7      while (overlap[i+1] > 0 &&
8             seq[i] != seq[overlap[i+1] - 1]) {
9        overlap[i+1] = overlap[overlap[i+1] - 1] + 1;
10     }
11   }
12   return overlap;
13 }
```

$$Overlap(seq.\alpha) = Overlap^x(seq) + 1$$

i	0	1	2	3	4
seq	C	O	C	O	\0
$overlap$	-1	0	0	1	2

ligne 12

Extraction de motif
Sommaire

III.1/ Alignement visuel de séquences

III.2/ Recherche de motif

III.3/ Extraction de motif
- Contexte
- Quelques programmes [presque] naïfs
- Algorithme KMR
- Structures d'index

Extraction de motif
Contexte

- Recherche *versus* extraction de motif

Extraction de motif
Contexte

- Recherche *versus* extraction de motif

  Recherche motif connu *a priori*

Extraction de motif
Contexte

- Recherche *versus* extraction de motif

  Recherche motif connu *a priori*
  Extraction motif inconnu *a priori*

Extraction de motif
Contexte

- Recherche *versus* extraction de motif

  Recherche motif connu *a priori*
  Extraction motif inconnu *a priori*

$\Rightarrow$ Espace des possible très différent ! ! !

Extraction de motif
Contexte

- Recherche *versus* extraction de motif

  Recherche motif connu *a priori*
  Extraction motif inconnu *a priori*

$\Rightarrow$ Espace des possible très différent ! ! !

$\Rightarrow$ Incidence de la notion de similarité

Extraction de motif
Contexte

- Recherche *versus* extraction de motif

  Recherche motif connu *a priori*
  Extraction motif inconnu *a priori*

$\Rightarrow$ Espace des possible très différent ! ! !

$\Rightarrow$ Incidence de la notion de similarité

Problème 1 Extraction des motifs répétés de longueur k

Extraction de motif
Contexte

- Recherche *versus* extraction de motif

  Recherche motif connu *a priori*
  Extraction motif inconnu *a priori*

$\Rightarrow$ Espace des possible très différent ! ! !

$\Rightarrow$ Incidence de la notion de similarité

Problème 1 Extraction des motifs répétés de longueur k

Problème 2 Extraction du(des) plus long(s) motif(s) répété(s)

Extraction exacte

```
1  bool Extract(const string &seq, unsigned int k) {
2    unsigned int i, j, l, lg = seq.length();
3    bool result = false;
4    for (i = 0; i < lg - k; i++) {
5      string motif = seq.substr(i, k);
6      int *overlap = Overlap(motif);
7      string msg = "";
8      j = i + 1;
9      while (j < lg - k) {
10       bool ok = true;
11       for (l = 0; ok && (l < k); l++) {
12         ok = (seq[j + l] == motif[l]);
13       }
14       if (ok) {
15         if (msg.empty()) {
16           msg = string("Le motif '") + motif + "' apparait aux positions " + i;
17         }
18         msg.append(", ").append(j);
19         l--;
20       }
21       j += l - overlap[l + 1] + 1;
22     }
23     delete [] overlap;
24     if (!msg.empty()) {
25       ostream << msg << "." << endl;
26       result = true;
27     }
28   }
29   return result;
30 }
```

Extraction exacte

```
1  bool Extract(const string &seq, unsigned int k) {
2    unsigned int i, j, l, lg = seq.length();
3    bool result = false;
4    for (i = 0; i < lg - k; i++) {
5      string motif = seq.substr(i, k);
6      int *overlap = Overlap(motif);
7      string msg = "";
8      j = i + 1;
9      while (j < lg - k) {
10       bool ok = true;
11       for (l = 0; ok && (l < k); l++) {
12         ok = (seq[j + l] == motif[l]);
13       }
14       if (ok) {
15         if (msg.empty()) {
16           msg = string("Le motif '") + motif + "' apparait aux positions " + i;
17         }
18         msg.append(", ").append(j);
19         l--;
20       }
21       j += l - overlap[l + 1] + 1;
22     }
23     delete [] overlap;
24     if (!msg.empty()) {
25       ostream << msg << "." << endl;
26       result = true;
27     }
28   }
29   return result;
30 }
```

Complexité en $O\left(n^2 - k^2\right)$

Extraction exacte

```
1 int Extract(const string &seq) {
2   unsigned int i, j, k, l;
3   for (k = seq.length() - 1; k > 0; k--) {
4     if (Extract(seq, k)) {
5       return k;
6     }
7   }
8   return 0;
9 }
```

HLIN608 – Algorithmique du texte.

Alban MANCHERON

Alignement visuel de séquences

Recherche de motif

Extraction de motif

Contexte

Quelques programmes [presque] naïfs

Algorithme KMR

Structures d'index

Extraction exacte

```
int Extract(const string &seq) {
  unsigned int i, j, k, l;
  for (k = seq.length() - 1; k > 0; k--) {
    if (Extract(seq, k)) {
      return k;
    }
  }
  return 0;
}
```

$$\text{Complexité en} \begin{cases} \Omega(n) \\ O(n^3) \end{cases}$$

Extraction de motif
Algorithme KMR [1/3]

⇒ Karp, Miller & Rosenberg, 1972

$\Rightarrow$ KARP, MILLER & ROSENBERG, 1972

Définition (Relation d'équivalence)

Deux positions i et j dans une chaîne de caractères s de longueur n sont k-équivalentes, noté $i\ E_k\ j$, si et seulement si les deux facteurs de longueur k débutant aux positions i et j dans s sont identiques ($s[i..i+k-1] = s[j..j+k-1]$).

$\Rightarrow$ KARP, MILLER & ROSENBERG, 1972

Définition (Relation d'équivalence)

Deux positions i et j dans une chaîne de caractères s de longueur n sont k-équivalentes, noté $i\ E_k\ j$, si et seulement si les deux facteurs de longueur k débutant aux positions i et j dans s sont identiques ($s[i..i+k-1] = s[j..j+k-1]$).

Lemme

Soient a, b, i et j tels que $1 \leq i \leq n$, $1 \leq j \leq n$, $a \geq b > 0$ et $\max(i,j) + a + b - 1 \leq n$, alors
$i\ E_{a+b}\ j \Leftrightarrow (i\ E_a\ j \wedge i+b\ E_a\ j+b)$, *avec* $b \leq a$.

Extraction de motif
Algorithme KMR [1/3]

⇒ KARP, MILLER & ROSENBERG, 1972

Définition (Relation d'équivalence)

Deux positions i et j dans une chaîne de caractères s de longueur n sont k-équivalentes, noté $i\ E_k\ j$, si et seulement si les deux facteurs de longueur k débutant aux positions i et j dans s sont identiques ($s[i..i+k-1] = s[j..j+k-1]$).

Lemme

Soient a, b, i et j tels que $1 \leq i \leq n$, $1 \leq j \leq n$, $a \geq b > 0$ et $\max(i,j) + a + b - 1 \leq n$, alors
$i\ E_{a+b}\ j \Leftrightarrow (i\ E_a\ j \wedge i+b\ E_a\ j+b)$, *avec* $b \leq a$.

s

$\Rightarrow$ KARP, MILLER & ROSENBERG, 1972

Définition (Relation d'équivalence)

Deux positions i et j dans une chaîne de caractères s de longueur n sont k-équivalentes, noté $i\ E_k\ j$, si et seulement si les deux facteurs de longueur k débutant aux positions i et j dans s sont identiques ($s[i..i+k-1] = s[j..j+k-1]$).

Lemme

Soient a, b, i et j tels que $1 \leq i \leq n$, $1 \leq j \leq n$, $a \geq b > 0$ et $\max(i,j) + a + b - 1 \leq n$, alors
$i\ E_{a+b}\ j \Leftrightarrow (i\ E_a\ j \wedge i+b\ E_a\ j+b)$, *avec* $b \leq a$.

⇒ KARP, MILLER & ROSENBERG, 1972

Définition (Relation d'équivalence)

Deux positions i et j dans une chaîne de caractères s de longueur n sont k-équivalentes, noté $i\ E_k\ j$, si et seulement si les deux facteurs de longueur k débutant aux positions i et j dans s sont identiques ($s[i..i+k-1] = s[j..j+k-1]$).

Lemme

Soient a, b, i et j tels que $1 \leq i \leq n$, $1 \leq j \leq n$, $a \geq b > 0$ et $\max(i,j) + a + b - 1 \leq n$, alors
$i\ E_{a+b}\ j \Leftrightarrow (i\ E_a\ j \wedge i+b\ E_a\ j+b)$, *avec* $b \leq a$.

$\Rightarrow$ KARP, MILLER & ROSENBERG, 1972

Définition (Relation d'équivalence)

Deux positions i et j dans une chaîne de caractères s de longueur n sont k-équivalentes, noté $i\ E_k\ j$, si et seulement si les deux facteurs de longueur k débutant aux positions i et j dans s sont identiques ($s[i..i+k-1] = s[j..j+k-1]$).

Lemme

Soient a, b, i et j tels que $1 \leq i \leq n$, $1 \leq j \leq n$, $a \geq b > 0$ et $\max(i,j) + a + b - 1 \leq n$, alors
$i\ E_{a+b}\ j \Leftrightarrow (i\ E_a\ j \wedge i+b\ E_a\ j+b)$, *avec* $b \leq a$.

Extraction de motif
Algorithme KMR [2/3]

Algorithme KMR $\Rightarrow$ construction des relations E_k à partir de E_1

HLIN608 – *Algorithmique du texte.*

Alban MANCHERON

Alignement visuel de séquences

Recherche de motif

Extraction de motif

Contexte

Quelques programmes [presque] naïfs

Algorithme KMR

Structures d'index

Algorithme KMR $\Rightarrow$ construction des relations E_k à partir de E_1

1. Principe de fonctionnement

Extraction de motif
Algorithme KMR [2/3]

Algorithme KMR $\Rightarrow$ construction des relations E_k à partir de E_1

1. Principe de fonctionnement
   1. La relation E_k est représentée par le vecteur v_k de taille $n - k + 1$

Extraction de motif
Algorithme KMR [2/3]

Algorithme KMR $\Rightarrow$ construction des relations E_k à partir de E_1

1. Principe de fonctionnement
   1. La relation E_k est représentée par le vecteur v_k de taille $n - k + 1$
   2. Chaque relation E_k détermine e_k classes, numérotées de 1 a e_k.

Extraction de motif
Algorithme KMR [2/3]

Algorithme KMR $\Rightarrow$ construction des relations E_k à partir de E_1

1. Principe de fonctionnement
   1. La relation E_k est représentée par le vecteur v_k de taille $n - k + 1$
   2. Chaque relation E_k détermine e_k classes, numérotées de 1 a e_k.
   3. E_{a+b} est construit à partir de E_a.

Extraction de motif
Algorithme KMR [2/3]

Algorithme KMR $\Rightarrow$ construction des relations E_k à partir de E_1

1. Principe de fonctionnement
   1. La relation E_k est représentée par le vecteur v_k de taille $n - k + 1$
   2. Chaque relation E_k détermine e_k classes, numérotées de 1 a e_k.
   3. E_{a+b} est construit à partir de E_a.
   4. Utilisation de deux ensembles $\mathcal{P}$ et $\mathcal{Q}$ de e_a piles chacun.

Extraction de motif
Algorithme KMR [2/3]

Algorithme KMR $\Rightarrow$ construction des relations E_k à partir de E_1

1. Principe de fonctionnement
   1. La relation E_k est représentée par le vecteur v_k de taille $n - k + 1$
   2. Chaque relation E_k détermine e_k classes, numérotées de 1 a e_k.
   3. E_{a+b} est construit à partir de E_a.
   4. Utilisation de deux ensembles $\mathcal{P}$ et $\mathcal{Q}$ de e_a piles chacun.
2. Construction

Extraction de motif
Algorithme KMR [2/3]

Algorithme KMR $\Rightarrow$ construction des relations E_k à partir de E_1

1. Principe de fonctionnement
   1. La relation E_k est représentée par le vecteur v_k de taille $n - k + 1$
   2. Chaque relation E_k détermine e_k classes, numérotées de 1 a e_k.
   3. E_{a+b} est construit à partir de E_a.
   4. Utilisation de deux ensembles $\mathcal{P}$ et $\mathcal{Q}$ de e_a piles chacun.
2. Construction
   1. Chaque position i de la séquence s est placé dans la pile de l'ensemble $\mathcal{P}$ qui correspond à la classe de E_a du mot $s[i..i + a - 1]$ (*i.e.*, $\mathcal{P}_{v_a[i]}$)

Algorithme KMR $\Rightarrow$ construction des relations E_k à partir de E_1

1. Principe de fonctionnement
   1. La relation E_k est représentée par le vecteur v_k de taille $n - k + 1$
   2. Chaque relation E_k détermine e_k classes, numérotées de 1 a e_k.
   3. E_{a+b} est construit à partir de E_a.
   4. Utilisation de deux ensembles $\mathcal{P}$ et $\mathcal{Q}$ de e_a piles chacun.
2. Construction
   1. Chaque position i de la séquence s est placé dans la pile de l'ensemble $\mathcal{P}$ qui correspond à la classe de E_a du mot $s[i..i + a - 1]$ (*i.e.*, $\mathcal{P}_{v_a[i]}$)
   2. Les piles de $\mathcal{P}$ sont dépilées successivement. Chaque position i est placée dans la pile $\mathcal{Q}_{v_a[i+b]}$.

Algorithme KMR $\Rightarrow$ construction des relations E_k à partir de E_1

1. Principe de fonctionnement
   1. La relation E_k est représentée par le vecteur v_k de taille $n - k + 1$
   2. Chaque relation E_k détermine e_k classes, numérotées de 1 a e_k.
   3. E_{a+b} est construit à partir de E_a.
   4. Utilisation de deux ensembles $\mathcal{P}$ et $\mathcal{Q}$ de e_a piles chacun.
2. Construction
   1. Chaque position i de la séquence s est placé dans la pile de l'ensemble $\mathcal{P}$ qui correspond à la classe de E_a du mot $s[i..i + a - 1]$ (*i.e.*, $\mathcal{P}_{v_a[i]}$)
   2. Les piles de $\mathcal{P}$ sont dépilées successivement. Chaque position i est placée dans la pile $\mathcal{Q}_{v_a[i+b]}$.
   3. Les piles de $\mathcal{Q}$ sont dépilées successivement. La classe $v_a[i]$ de chaque position i déjà retirée est comparée à la classe précédente. Si les deux classes sont différentes, les positions proviennent de deux piles de $\mathcal{P}$ différentes et le nombre de classes de E_{a+b} est incrémenté.

Extraction de motif
Algorithme KMR [3/3]

i	1	2	3	4	5	6	7	8	9
s	R	O	U	D	O	U	D	O	U

Extraction de motif
Algorithme KMR [3/3]

i	1	2	3	4	5	6	7	8	9
s	R	O	U	D	O	U	D	O	U

$e_1 \Rightarrow ?$

Extraction de motif
Algorithme KMR [3/3]

i	1	2	3	4	5	6	7	8	9
s	R	O	U	D	O	U	D	O	U
v_1	1	2	3	4	2	3	4	2	3

$e_1 \Rightarrow 4$

classe 1 $s[1] = \texttt{R}$

classe 2 $s[2] = s[5] = s[8] = \texttt{O}$

classe 3 $s[3] = s[6] = s[9] = \texttt{U}$

classe 4 $s[4] = s[7] = \texttt{D}$

Extraction de motif
Algorithme KMR [3/3]

i	1	2	3	4	5	6	7	8	9
s	R	O	U	D	O	U	D	O	U
v_1	1	2	3	4	2	3	4	2	3

$e_2 \Rightarrow ?$

$a = 1, b = 1$

classe 1 $s[1] = \texttt{R}$

classe 2 $s[2] = s[5] = s[8] = \texttt{O}$

classe 3 $s[3] = s[6] = s[9] = \texttt{U}$

classe 4 $s[4] = s[7] = \texttt{D}$

Extraction de motif
Algorithme KMR [3/3]

i	1	2	3	4	5	6	7	8	9
s	R	O	U	D	O	U	D	O	U
v_1	1	2	3	4	2	3	4	2	3

$e_2 \Rightarrow ?$

$a = 1, b = 1$

classe 1 $s[1] = \texttt{R}$

classe 2 $s[2] = s[5] = s[8] = \texttt{O}$

classe 3 $s[3] = s[6] = s[9] = \texttt{U}$

classe 4 $s[4] = s[7] = \texttt{D}$

Extraction de motif
Algorithme KMR [3/3]

i	1	2	3	4	5	6	7	8	9
s	R	O	U	D	O	U	D	O	U
v_1	1	2	3	4	2	3	4	2	3

$e_2 \Rightarrow ?$

$a = 1, b = 1$

classe 1 $s[1] = \texttt{R}$

classe 2 $s[2] = s[5] = s[8] = \texttt{O}$

classe 3 $s[3] = s[6] = s[9] = \texttt{U}$

classe 4 $s[4] = s[7] = \texttt{D}$

Extraction de motif
Algorithme KMR [3/3]

i	1	2	3	4	5	6	7	8	9
s	R	O	U	D	O	U	D	O	U
v_1	1	2	3	4	2	3	4	2	3

$e_2 \Rightarrow ?$

$a = 1, b = 1$

classe 1 $s[1] = \texttt{R}$

classe 2 $s[2] = s[5] = s[8] = \texttt{O}$

classe 3 $s[3] = s[6] = s[9] = \texttt{U}$

classe 4 $s[4] = s[7] = \texttt{D}$

Extraction de motif
Algorithme KMR [3/3]

i	1	2	3	4	5	6	7	8	9
s	R	O	U	D	O	U	D	O	U
v_1	1	2	3	4	2	3	4	2	3

$e_2 \Rightarrow ?$

$a = 1, b = 1$

classe 1 $s[1] = \texttt{R}$

classe 2 $s[2] = s[5] = s[8] = \texttt{O}$

classe 3 $s[3] = s[6] = s[9] = \texttt{U}$

classe 4 $s[4] = s[7] = \texttt{D}$

Extraction de motif
Algorithme KMR [3/3]

i	1	2	3	4	5	6	7	8	9
s	R	O	U	D	O	U	D	O	U
v_1	1	2	3	4	2	3	4	2	3

$e_2 \Rightarrow ?$

$a = 1, b = 1$

classe 1 $s[1] = \texttt{R}$

classe 2 $s[2] = s[5] = s[8] = \texttt{O}$

classe 3 $s[3] = s[6] = s[9] = \texttt{U}$

classe 4 $s[4] = s[7] = \texttt{D}$

Extraction de motif

Algorithme KMR [3/3]

i	1	2	3	4	5	6	7	8	9
s	R	O	U	D	O	U	D	O	U
v_1	1	2	3	4	2	3	4	2	3

$e_2 \Rightarrow ?$

$a = 1, b = 1$

classe 1 $s[1] = \texttt{R}$

classe 2 $s[2] = s[5] = s[8] = \texttt{O}$

classe 3 $s[3] = s[6] = s[9] = \texttt{U}$

classe 4 $s[4] = s[7] = \texttt{D}$

Extraction de motif

Algorithme KMR [3/3]

i	1	2	3	4	5	6	7	8	9
s	R	O	U	D	O	U	D	O	U
v_1	1	2	3	4	2	3	4	2	3

$\mathcal{P}_1$	$\mathcal{P}_2$	$\mathcal{P}_3$	$\mathcal{P}_4$	$\mathcal{Q}_1$	$\mathcal{Q}_2$	$\mathcal{Q}_3$	$\mathcal{Q}_4$
	8						
	5	6	7				
1	2	3	4				

$e_2 \Rightarrow ?$

$a = 1, b = 1$

classe 1 $s[1] = \texttt{R}$

classe 2 $s[2] = s[5] = s[8] = \texttt{O}$

classe 3 $s[3] = s[6] = s[9] = \texttt{U}$

classe 4 $s[4] = s[7] = \texttt{D}$

Extraction de motif
Algorithme KMR [3/3]

i	1	2	3	4	5	6	7	8	9
s	R	O	U	D	O	U	D	O	U
v_1	1	2	3	4	2	3	4	2	3

$\mathcal{P}_1$	$\mathcal{P}_2$	$\mathcal{P}_3$	$\mathcal{P}_4$	$\mathcal{Q}_1$	$\mathcal{Q}_2$	$\mathcal{Q}_3$	$\mathcal{Q}_4$
	8	9					
	5	6	7				
1	2	3	4				

$e_2 \Rightarrow ?$

$a = 1, b = 1$

classe 1 $s[1] = \texttt{R}$

classe 2 $s[2] = s[5] = s[8] = \texttt{O}$

classe 3 $s[3] = s[6] = s[9] = \texttt{U}$

classe 4 $s[4] = s[7] = \texttt{D}$

Extraction de motif

Algorithme KMR [3/3]

i	1	2	3	4	5	6	7	8	9
s	R	O	U	D	O	U	D	O	U
v_1	1	2	3	4	2	3	4	2	3

$\mathcal{P}_1$	$\mathcal{P}_2$	$\mathcal{P}_3$	$\mathcal{P}_4$	$\mathcal{Q}_1$	$\mathcal{Q}_2$	$\mathcal{Q}_3$	$\mathcal{Q}_4$
	8	9					
	5	6	7				
~~1~~	2	3	4		1		

$e_2 \Rightarrow ?$

$a = 1, b = 1$

classe 1 $s[1] = \texttt{R}$

classe 2 $s[2] = s[5] = s[8] = \texttt{O}$

classe 3 $s[3] = s[6] = s[9] = \texttt{U}$

classe 4 $s[4] = s[7] = \texttt{D}$

Extraction de motif
Algorithme KMR [3/3]

i	1	2	3	4	5	6	7	8	9
s	R	O	U	D	O	U	D	O	U
v_1	1	2	3	4	2	3	4	2	3

$\mathcal{P}_1$	$\mathcal{P}_2$	$\mathcal{P}_3$	$\mathcal{P}_4$
	$\cancel{8}$	9	
	5	6	7
	2	3	4

$\mathcal{Q}_1$	$\mathcal{Q}_2$	$\mathcal{Q}_3$	$\mathcal{Q}_4$
		8	
	1		

$e_2 \Rightarrow ?$

$a = 1, b = 1$

classe 1 $s[1] = \texttt{R}$

classe 2 $s[2] = s[5] = s[8] = \texttt{O}$

classe 3 $s[3] = s[6] = s[9] = \texttt{U}$

classe 4 $s[4] = s[7] = \texttt{D}$

Extraction de motif
Algorithme KMR [3/3]

i	1	2	3	4	5	6	7	8	9
s	R	O	U	D	O	U	D	O	U
v_1	1	2	3	4	2	3	4	2	3

$\mathcal{P}_1$	$\mathcal{P}_2$	$\mathcal{P}_3$	$\mathcal{P}_4$
		9	
	~~5~~	6	7
	2	3	4

$\mathcal{Q}_1$	$\mathcal{Q}_2$	$\mathcal{Q}_3$	$\mathcal{Q}_4$
		5	
		8	
	1		

$e_2 \Rightarrow ?$

$a = 1, b = 1$

classe 1 $s[1] = \texttt{R}$

classe 2 $s[2] = s[5] = s[8] = \texttt{O}$

classe 3 $s[3] = s[6] = s[9] = \texttt{U}$

classe 4 $s[4] = s[7] = \texttt{D}$

Extraction de motif

Algorithme KMR [3/3]

i	1	2	3	4	5	6	7	8	9
s	R	O	U	D	O	U	D	O	U
v_1	1	2	3	4	2	3	4	2	3

$e_2 \Rightarrow ?$

$a = 1, b = 1$

classe 1 $s[1] = \texttt{R}$

classe 2 $s[2] = s[5] = s[8] = \texttt{O}$

classe 3 $s[3] = s[6] = s[9] = \texttt{U}$

classe 4 $s[4] = s[7] = \texttt{D}$

Extraction de motif

Algorithme KMR [3/3]

i	1	2	3	4	5	6	7	8	9
s	R	O	U	D	O	U	D	O	U
v_1	1	2	3	4	2	3	4	2	3

$\mathcal{P}_1$	$\mathcal{P}_2$	$\mathcal{P}_3$	$\mathcal{P}_4$
		$\cancel{9}$	
		6	7
		3	4

$\mathcal{Q}_1$	$\mathcal{Q}_2$	$\mathcal{Q}_3$	$\mathcal{Q}_4$
		2	
		5	
		8	
	1		

$e_2 \Rightarrow ?$

$a = 1, b = 1$

classe 1 $s[1] = \texttt{R}$

classe 2 $s[2] = s[5] = s[8] = \texttt{O}$

classe 3 $s[3] = s[6] = s[9] = \texttt{U}$

classe 4 $s[4] = s[7] = \texttt{D}$

Extraction de motif

Algorithme KMR [3/3]

i	1	2	3	4	5	6	7	8	9
s	R	O	U	D	O	U	D	O	U
v_1	1	2	3	4	2	3	4	2	3

$e_2 \Rightarrow ?$

$a = 1, b = 1$

classe 1 $s[1] = \texttt{R}$

classe 2 $s[2] = s[5] = s[8] = \texttt{O}$

classe 3 $s[3] = s[6] = s[9] = \texttt{U}$

classe 4 $s[4] = s[7] = \texttt{D}$

Extraction de motif

Algorithme KMR [3/3]

i	1	2	3	4	5	6	7	8	9
s	R	O	U	D	O	U	D	O	U
v_1	1	2	3	4	2	3	4	2	3

$\mathcal{P}_1$	$\mathcal{P}_2$	$\mathcal{P}_3$	$\mathcal{P}_4$
			7
		$\cancel{3}$	4

$\mathcal{Q}_1$	$\mathcal{Q}_2$	$\mathcal{Q}_3$	$\mathcal{Q}_4$
			3
			6
		2	
		5	
		8	
	1		

$e_2 \Rightarrow ?$

$a = 1, b = 1$

classe 1 $s[1] = \texttt{R}$

classe 2 $s[2] = s[5] = s[8] = \texttt{O}$

classe 3 $s[3] = s[6] = s[9] = \texttt{U}$

classe 4 $s[4] = s[7] = \texttt{D}$

Extraction de motif

Algorithme KMR [3/3]

i	1	2	3	4	5	6	7	8	9
s	R	O	U	D	O	U	D	O	U
v_1	1	2	3	4	2	3	4	2	3

$\mathcal{P}_1$	$\mathcal{P}_2$	$\mathcal{P}_3$	$\mathcal{P}_4$	$\mathcal{Q}_1$	$\mathcal{Q}_2$	$\mathcal{Q}_3$	$\mathcal{Q}_4$
					7		
							3 6
						2 5 8	
			~~7~~ 4		1		

$e_2 \Rightarrow ?$

$a = 1, b = 1$

classe 1 $s[1] = \texttt{R}$

classe 2 $s[2] = s[5] = s[8] = \texttt{O}$

classe 3 $s[3] = s[6] = s[9] = \texttt{U}$

classe 4 $s[4] = s[7] = \texttt{D}$

Extraction de motif
Algorithme KMR [3/3]

i	1	2	3	4	5	6	7	8	9
s	R	O	U	D	O	U	D	O	U
v_1	1	2	3	4	2	3	4	2	3

$\mathcal{P}_1$	$\mathcal{P}_2$	$\mathcal{P}_3$	$\mathcal{P}_4$
			$\cancel{4}$

$\mathcal{Q}_1$	$\mathcal{Q}_2$	$\mathcal{Q}_3$	$\mathcal{Q}_4$
	4 7		
			3 6
		2 5 8	
	1		

$e_2 \Rightarrow ?$

$a = 1, b = 1$

classe 1 $s[1] = \texttt{R}$

classe 2 $s[2] = s[5] = s[8] = \texttt{O}$

classe 3 $s[3] = s[6] = s[9] = \texttt{U}$

classe 4 $s[4] = s[7] = \texttt{D}$

Extraction de motif

Algorithme KMR [3/3]

i	1	2	3	4	5	6	7	8	9
s	R	O	U	D	O	U	D	O	U
v_1	1	2	3	4	2	3	4	2	3

	$\mathcal{P}_1$	$\mathcal{P}_2$	$\mathcal{P}_3$	$\mathcal{P}_4$
$\mathcal{Q}_1$				
$\mathcal{Q}_2$	4, 7			
$\mathcal{Q}_3$				3, 6
$\mathcal{Q}_4$		2, 5, 8		
	1			

$e_2 \Rightarrow ?$

$a = 1, b = 1$

classe 1 $s[4..5] = s[7..8] = \texttt{DO}$

classe 2 $s[3..4] = s[6..7] = \texttt{UD}$

classe 3
$s[2..3] = s[5..6] = s[8..9] = \texttt{OU}$

classe ignorée $s[1..2] = \texttt{RO}$

Extraction de motif

Algorithme KMR [3/3]

i	1	2	3	4	5	6	7	8	9
s	R	O	U	D	O	U	D	O	U
v_1	-	3	2	1	3	2	1	3	-

$e_2 \Rightarrow 3$

classe 1 $s[4..5] = s[7..8] = \texttt{DO}$

classe 2 $s[3..4] = s[6..7] = \texttt{UD}$

classe 3
$s[2..3] = s[5..6] = s[8..9] = \texttt{OU}$

classe ignorée $s[1..2] = \texttt{RO}$

Extraction de motif

Algorithme KMR [3/3]

i	1	2	3	4	5	6	7	8	9
s	R	O	U	D	O	U	D	O	U
v_2	-	3	2	1	3	2	1	3	-

$e_4 \Rightarrow ?$

$a = 2, b = 2$

classe 1 $s[4..5] = s[7..8] = \texttt{DO}$

classe 2 $s[3..4] = s[6..7] = \texttt{UD}$

classe 3
$s[2..3] = s[5..6] = s[8..9] = \texttt{OU}$

classe ignorée $s[1..2] = \texttt{RO}$

Extraction de motif

Algorithme KMR [3/3]

i	1	2	3	4	5	6	7	8	9
s	R	O	U	D	O	U	D	O	U
v_2	-	3	2	1	3	2	1	3	-

$e_4 \Rightarrow$?

$a = 2, b = 2$

classe 1 $s[4..5] = s[7..8] = \texttt{DO}$

classe 2 $s[3..4] = s[6..7] = \texttt{UD}$

classe 3
$s[2..3] = s[5..6] = s[8..9] = \texttt{OU}$

classe ignorée $s[1..2] = \texttt{RO}$

Extraction de motif

Algorithme KMR [3/3]

i	1	2	3	4	5	6	7	8	9
s	R	O	U	D	O	U	D	O	U
v_2	-	3	2	1	3	2	1	3	-

$e_4 \Rightarrow ?$

$a = 2, b = 2$

classe 1 $s[4..5] = s[7..8] = \texttt{DO}$

classe 2 $s[3..4] = s[6..7] = \texttt{UD}$

classe 3
$s[2..3] = s[5..6] = s[8..9] = \texttt{OU}$

classe ignorée $s[1..2] = \texttt{RO}$

Extraction de motif

Algorithme KMR [3/3]

i	1	2	3	4	5	6	7	8	9
s	R	O	U	D	O	U	D	O	U
v_2	-	3	2	1	3	2	1	3	-

$e_4 \Rightarrow ?$

$a = 2, b = 2$

classe 1 $s[4..5] = s[7..8] = \texttt{DO}$

classe 2 $s[3..4] = s[6..7] = \texttt{UD}$

classe 3
$s[2..3] = s[5..6] = s[8..9] = \texttt{OU}$

classe ignorée $s[1..2] = \texttt{RO}$

Extraction de motif

Algorithme KMR [3/3]

i	1	2	3	4	5	6	7	8	9
s	R	O	U	D	O	U	D	O	U
v_2	-	3	2	1	3	2	1	3	-

$e_4 \Rightarrow ?$

$a = 2, b = 2$

classe 1 $s[4..5] = s[7..8] = \texttt{DO}$

classe 2 $s[3..4] = s[6..7] = \texttt{UD}$

classe 3
$s[2..3] = s[5..6] = s[8..9] = \texttt{OU}$

classe ignorée $s[1..2] = \texttt{RO}$

Extraction de motif

Algorithme KMR [3/3]

i	1	2	3	4	5	6	7	8	9
s	R	O	U	D	O	U	D	O	U
v_2	-	3	2	1	3	2	1	3	-

$\mathcal{P}_1$	$\mathcal{P}_2$	$\mathcal{P}_3$	$\mathcal{Q}_1$	$\mathcal{Q}_2$	$\mathcal{Q}_3$
	6	5			
4	3	2			

$e_4 \Rightarrow ?$

$a = 2, b = 2$

classe 1 $s[4..5] = s[7..8] = \texttt{DO}$

classe 2 $s[3..4] = s[6..7] = \texttt{UD}$

classe 3
$s[2..3] = s[5..6] = s[8..9] = \texttt{OU}$

classe ignorée $s[1..2] = \texttt{RO}$

Extraction de motif

Algorithme KMR [3/3]

i	1	2	3	4	5	6	7	8	9
s	R	O	U	D	O	U	D	O	U
v_2	-	3	2	1	3	2	1	3	-

$\mathcal{P}_1$	$\mathcal{P}_2$	$\mathcal{P}_3$	$\mathcal{Q}_1$	$\mathcal{Q}_2$	$\mathcal{Q}_3$
7	6	5			
4	3	2			

$e_4 \Rightarrow ?$

$a = 2, b = 2$

classe 1 $s[4..5] = s[7..8] = \texttt{DO}$

classe 2 $s[3..4] = s[6..7] = \texttt{UD}$

classe 3
$s[2..3] = s[5..6] = s[8..9] = \texttt{OU}$

classe ignorée $s[1..2] = \texttt{RO}$

Extraction de motif

Algorithme KMR [3/3]

i	1	2	3	4	5	6	7	8	9
s	R	O	U	D	O	U	D	O	U
v_2	-	3	2	1	3	2	1	3	-

$\mathcal{P}_1$	$\mathcal{P}_2$	$\mathcal{P}_3$
		8
7	6	5
4	3	2

$\mathcal{Q}_1$	$\mathcal{Q}_2$	$\mathcal{Q}_3$

$e_4 \Rightarrow ?$

$a = 2, b = 2$

classe 1 $s[4..5] = s[7..8] = \texttt{DO}$

classe 2 $s[3..4] = s[6..7] = \texttt{UD}$

classe 3
$s[2..3] = s[5..6] = s[8..9] = \texttt{OU}$

classe ignorée $s[1..2] = \texttt{RO}$

Extraction de motif

Algorithme KMR [3/3]

i	1	2	3	4	5	6	7	8	9
s	R	O	U	D	O	U	D	O	U
v_2	-	3	2	1	3	2	1	3	-

$\mathcal{P}_1$	$\mathcal{P}_2$	$\mathcal{P}_3$	$\mathcal{Q}_1$	$\mathcal{Q}_2$	$\mathcal{Q}_3$
		8			
7	6	5			
4	3	2			

$e_4 \Rightarrow ?$

$a = 2, b = 2$

classe 1 $s[4..5] = s[7..8] = \texttt{DO}$

classe 2 $s[3..4] = s[6..7] = \texttt{UD}$

classe 3
$s[2..3] = s[5..6] = s[8..9] = \texttt{OU}$

classe ignorée $s[1..2] = \texttt{RO}$

Extraction de motif

Algorithme KMR [3/3]

i	1	2	3	4	5	6	7	8	9
s	R	O	U	D	O	U	D	O	U
v_2	-	3	2	1	3	2	1	3	-

$\mathcal{P}_1$	$\mathcal{P}_2$	$\mathcal{P}_3$	$\mathcal{Q}_1$	$\mathcal{Q}_2$	$\mathcal{Q}_3$
		8			
~~7~~	6	5			
4	3	2			

$e_4 \Rightarrow ?$

$a = 2, b = 2$

classe 1 $s[4..5] = s[7..8] = \texttt{DO}$

classe 2 $s[3..4] = s[6..7] = \texttt{UD}$

classe 3
$s[2..3] = s[5..6] = s[8..9] = \texttt{OU}$

classe ignorée $s[1..2] = \texttt{RO}$

Extraction de motif

Algorithme KMR [3/3]

i	1	2	3	4	5	6	7	8	9
s	R	O	U	D	O	U	D	O	U
v_2	-	3	2	1	3	2	1	3	-

$\mathcal{P}_1$	$\mathcal{P}_2$	$\mathcal{P}_3$
		8
	6	5
$\cancel{4}$	3	2

$\mathcal{Q}_1$	$\mathcal{Q}_2$	$\mathcal{Q}_3$
	4	

$e_4 \Rightarrow ?$

$a = 2, b = 2$

classe 1 $s[4..5] = s[7..8] = \texttt{DO}$

classe 2 $s[3..4] = s[6..7] = \texttt{UD}$

classe 3
$s[2..3] = s[5..6] = s[8..9] = \texttt{OU}$

classe ignorée $s[1..2] = \texttt{RO}$

Extraction de motif

Algorithme KMR [3/3]

i	1	2	3	4	5	6	7	8	9
s	R	O	U	D	O	U	D	O	U
v_2	-	3	2	1	3	2	1	3	-

$\mathcal{P}_1$	$\mathcal{P}_2$	$\mathcal{P}_3$
		8
	~~6~~	5
	3	2

$\mathcal{Q}_1$	$\mathcal{Q}_2$	$\mathcal{Q}_3$
		6
	4	

$e_4 \Rightarrow ?$

$a = 2, b = 2$

classe 1 $s[4..5] = s[7..8] = \texttt{DO}$

classe 2 $s[3..4] = s[6..7] = \texttt{UD}$

classe 3
$s[2..3] = s[5..6] = s[8..9] = \texttt{OU}$

classe ignorée $s[1..2] = \texttt{RO}$

Extraction de motif

Algorithme KMR [3/3]

i	1	2	3	4	5	6	7	8	9
s	R	O	U	D	O	U	D	O	U
v_2	-	3	2	1	3	2	1	3	-

$\mathcal{P}_1$	$\mathcal{P}_2$	$\mathcal{P}_3$
		8
		5
	$\cancel{3}$	2

$\mathcal{Q}_1$	$\mathcal{Q}_2$	$\mathcal{Q}_3$
		3
		6
	4	

$e_4 \Rightarrow ?$

$a = 2, b = 2$

classe 1 $s[4..5] = s[7..8] = \texttt{DO}$

classe 2 $s[3..4] = s[6..7] = \texttt{UD}$

classe 3
$s[2..3] = s[5..6] = s[8..9] = \texttt{OU}$

classe ignorée $s[1..2] = \texttt{RO}$

Extraction de motif

Algorithme KMR [3/3]

i	1	2	3	4	5	6	7	8	9
s	R	O	U	D	O	U	D	O	U
v_2	-	3	2	1	3	2	1	3	-

$\mathcal{P}_1$	$\mathcal{P}_2$	$\mathcal{P}_3$
		~~8~~
		5
		2

$\mathcal{Q}_1$	$\mathcal{Q}_2$	$\mathcal{Q}_3$
		3
		6
	4	

$e_4 \Rightarrow ?$

$a = 2, b = 2$

classe 1 $s[4..5] = s[7..8] = \texttt{DO}$

classe 2 $s[3..4] = s[6..7] = \texttt{UD}$

classe 3
$s[2..3] = s[5..6] = s[8..9] = \texttt{OU}$

classe ignorée $s[1..2] = \texttt{RO}$

Extraction de motif

Algorithme KMR [3/3]

i	1	2	3	4	5	6	7	8	9
s	R	O	U	D	O	U	D	O	U
v_2	-	3	2	1	3	2	1	3	-

$e_4 \Rightarrow ?$

$a = 2, b = 2$

classe 1 $s[4..5] = s[7..8] = \texttt{DO}$

classe 2 $s[3..4] = s[6..7] = \texttt{UD}$

classe 3
$s[2..3] = s[5..6] = s[8..9] = \texttt{OU}$

classe ignorée $s[1..2] = \texttt{RO}$

Extraction de motif

Algorithme KMR [3/3]

i	1	2	3	4	5	6	7	8	9
s	R	O	U	D	O	U	D	O	U
v_2	-	3	2	1	3	2	1	3	-

$\mathcal{P}_1$	$\mathcal{P}_2$	$\mathcal{P}_3$
		$\cancel{2}$

$\mathcal{Q}_1$	$\mathcal{Q}_2$	$\mathcal{Q}_3$
2, 5		
		3, 6
	4	

$e_4 \Rightarrow ?$

$a = 2, b = 2$

classe 1 $s[4..5] = s[7..8] = \texttt{DO}$

classe 2 $s[3..4] = s[6..7] = \texttt{UD}$

classe 3
$s[2..3] = s[5..6] = s[8..9] = \texttt{OU}$

classe ignorée $s[1..2] = \texttt{RO}$

Extraction de motif

Algorithme KMR [3/3]

i	1	2	3	4	5	6	7	8	9
s	R	O	U	D	O	U	D	O	U
v_2	-	3	2	1	3	2	1	3	-

$\mathcal{P}_1$	$\mathcal{P}_2$	$\mathcal{P}_3$

$\mathcal{Q}_1$	$\mathcal{Q}_2$	$\mathcal{Q}_3$
2, 5		
		3, 6
	4	

$e_4 \Rightarrow ?$

$a = 2, b = 2$

classe 1 $s[2..5] = s[5..8] = \texttt{OUDO}$

classe 2 $s[3..6] = s[6..9] = \texttt{UDOU}$

classe ignorée $s[4..7] = \texttt{DOUD}$

Extraction de motif
Algorithme KMR [3/3]

i	1	2	3	4	5	6	7	8	9
s	R	O	U	D	O	U	D	O	U
v_4	-	1	2	-	1	2	-	-	-

$e_4 \Rightarrow 2$

classe 1 $s[2..5] = s[5..8] = \texttt{OUDO}$

classe 2 $s[3..6] = s[6..9] = \texttt{UDOU}$

classe ignorée $s[4..7] = \texttt{DOUD}$

Extraction de motif

Algorithme KMR [3/3]

i	1	2	3	4	5	6	7	8	9
s	R	O	U	D	O	U	D	O	U
v_4	-	1	2	-	1	2	-	-	-

$e_8 \Rightarrow ?$

$a = 4, b = 4$

classe 1 $s[2..5] = s[5..8] = \texttt{OUDO}$

classe 2 $s[3..6] = s[6..9] = \texttt{UDOU}$

classe ignorée $s[4..7] = \texttt{DOUD}$

Extraction de motif

Algorithme KMR [3/3]

i	1	2	3	4	5	6	7	8	9
s	R	O	U	D	O	U	D	O	U
v_4	-	1	2	-	1	2	-	-	-

$e_8 \Rightarrow ?$

$a = 4, b = 4$

classe 1 $s[2..5] = s[5..8] = \texttt{OUDO}$

classe 2 $s[3..6] = s[6..9] = \texttt{UDOU}$

classe ignorée $s[4..7] = \texttt{DOUD}$

Extraction de motif

Algorithme KMR [3/3]

i	1	2	3	4	5	6	7	8	9
s	R	O	U	D	O	U	D	O	U
v_4	-	1	2	-	1	2	-	-	-

$e_8 \Rightarrow ?$

$a = 4, b = 4$

classe 1 $s[2..5] = s[5..8] =$ OUDO

classe 2 $s[3..6] = s[6..9] =$ UDOU

classe ignorée $s[4..7] =$ DOUD

Extraction de motif

Algorithme KMR [3/3]

i	1	2	3	4	5	6	7	8	9
s	R	O	U	D	O	U	D	O	U
v_4	-	1	2	-	1	2	-	-	-

$e_8 \Rightarrow ?$

$a = 4, b = 4$

classe 1 $s[2..5] = s[5..8] = \texttt{OUDO}$

classe 2 $s[3..6] = s[6..9] = \texttt{UDOU}$

classe ignorée $s[4..7] = \texttt{DOUD}$

Extraction de motif

Algorithme KMR [3/3]

i	1	2	3	4	5	6	7	8	9
s	R	O	U	D	O	U	D	O	U
v_4	-	1	2	-	1	2	-	-	-

$e_8 \Rightarrow ?$

$a = 4, b = 4$

classe 1 $s[2..5] = s[5..8] = \texttt{OUDO}$

classe 2 $s[3..6] = s[6..9] = \texttt{UDOU}$

classe ignorée $s[4..7] = \texttt{DOUD}$

Extraction de motif

Algorithme KMR [3/3]

i	1	2	3	4	5	6	7	8	9
s	R	O	U	D	O	U	D	O	U
v_4	-	1	2	-	1	2	-	-	-

	$\mathcal{P}_1$	$\mathcal{P}_2$
	5	6
	2	3

	$\mathcal{Q}_1$	$\mathcal{Q}_2$

$e_8 \Rightarrow ?$

$a = 4, b = 4$

classe 1 $s[2..5] = s[5..8] = \texttt{OUDO}$

classe 2 $s[3..6] = s[6..9] = \texttt{UDOU}$

classe ignorée $s[4..7] = \texttt{DOUD}$

Extraction de motif

Algorithme KMR [3/3]

i	1	2	3	4	5	6	7	8	9
s	R	O	U	D	O	U	D	O	U
v_4	-	1	2	-	1	2	-	-	-

$e_8 \Rightarrow ?$

$a = 4, b = 4$

classe 1 $s[2..5] = s[5..8] = \texttt{OUDO}$

classe 2 $s[3..6] = s[6..9] = \texttt{UDOU}$

classe ignorée $s[4..7] = \texttt{DOUD}$

Extraction de motif

Algorithme KMR [3/3]

i	1	2	3	4	5	6	7	8	9
s	R	O	U	D	O	U	D	O	U
v_4	-	1	2	-	1	2	-	-	-

$\mathcal{P}_1$	$\mathcal{P}_2$
5	6
2	3

$\mathcal{Q}_1$	$\mathcal{Q}_2$

$e_8 \Rightarrow ?$

$a = 4, b = 4$

classe 1 $s[2..5] = s[5..8] = \texttt{OUDO}$

classe 2 $s[3..6] = s[6..9] = \texttt{UDOU}$

classe ignorée $s[4..7] = \texttt{DOUD}$

Extraction de motif

Algorithme KMR [3/3]

i	1	2	3	4	5	6	7	8	9
s	R	O	U	D	O	U	D	O	U
v_4	-	1	2	-	1	2	-	-	-

5	6		
2	3		
$\mathcal{P}_1$	$\mathcal{P}_2$	$\mathcal{Q}_1$	$\mathcal{Q}_2$

$e_8 \Rightarrow ?$

$a = 4, b = 4$

classe 1 $s[2..5] = s[5..8] = \texttt{OUDO}$

classe 2 $s[3..6] = s[6..9] = \texttt{UDOU}$

classe ignorée $s[4..7] = \texttt{DOUD}$

Extraction de motif

Algorithme KMR [3/3]

i	1	2	3	4	5	6	7	8	9
s	R	O	U	D	O	U	D	O	U
v_4	-	1	2	-	1	2	-	-	-

$\mathcal{P}_1$	$\mathcal{P}_2$
~~5~~	6
2	3

$\mathcal{Q}_1$	$\mathcal{Q}_2$

$e_8 \Rightarrow ?$

$a = 4, b = 4$

classe 1 $s[2..5] = s[5..8] = \texttt{OUDO}$

classe 2 $s[3..6] = s[6..9] = \texttt{UDOU}$

classe ignorée $s[4..7] = \texttt{DOUD}$

Extraction de motif

Algorithme KMR [3/3]

i	1	2	3	4	5	6	7	8	9
s	R	O	U	D	O	U	D	O	U
v_4	-	1	2	-	1	2	-	-	-

$e_8 \Rightarrow ?$

$a = 4, b = 4$

classe 1 $s[2..5] = s[5..8] = \texttt{OUDO}$

classe 2 $s[3..6] = s[6..9] = \texttt{UDOU}$

classe ignorée $s[4..7] = \texttt{DOUD}$

Extraction de motif

Algorithme KMR [3/3]

i	1	2	3	4	5	6	7	8	9
s	R	O	U	D	O	U	D	O	U
v_4	-	1	2	-	1	2	-	-	-

$e_8 \Rightarrow ?$

$a = 4, b = 4$

classe 1 $s[2..5] = s[5..8] = \texttt{OUDO}$

classe 2 $s[3..6] = s[6..9] = \texttt{UDOU}$

classe ignorée $s[4..7] = \texttt{DOUD}$

Extraction de motif

Algorithme KMR [3/3]

i	1	2	3	4	5	6	7	8	9
s	R	O	U	D	O	U	D	O	U
v_4	-	1	2	-	1	2	-	-	-

$e_8 \Rightarrow ?$

$a = 4, b = 4$

classe 1 $s[2..5] = s[5..8] =$ OUDO

classe 2 $s[3..6] = s[6..9] =$ UDOU

classe ignorée $s[4..7] =$ DOUD

Extraction de motif
Algorithme KMR [3/3]

i	1	2	3	4	5	6	7	8	9
s	R	O	U	D	O	U	D	O	U
v_4	-	1	2	-	1	2	-	-	-

$e_8 \Rightarrow ?$

$a = 4, b = 4$

classe ignorée $s[2..9] = \texttt{OUDOUDOU}$

Extraction de motif
Algorithme KMR [3/3]

i	1	2	3	4	5	6	7	8	9
s	R	O	U	D	O	U	D	O	U
v_8	-	-	-	-	-	-	-	-	-

$e_8 \Rightarrow 0$

classe ignorée $s[2..9] = \texttt{OUDOUDOU}$

Extraction de motif

Algorithme KMR [3/3]

i	1	2	3	4	5	6	7	8	9
s	R	O	U	D	O	U	D	O	U
v_4	-	1	2	-	1	2	-	-	-

5	6		
2	3		
$\mathcal{P}_1$	$\mathcal{P}_2$	$\mathcal{Q}_1$	$\mathcal{Q}_2$

$e_6 \Rightarrow ?$

$a = 4, b = 2$

classe 1 $s[2..5] = s[5..8] = \texttt{OUDO}$

classe 2 $s[3..6] = s[6..9] = \texttt{UDOU}$

classe ignorée $s[4..7] = \texttt{DOUD}$

Extraction de motif

Algorithme KMR [3/3]

i	1	2	3	4	5	6	7	8	9
s	R	O	U	D	O	U	D	O	U
v_4	-	1	2	-	1	2	-	-	-

$\mathcal{P}_1$	$\mathcal{P}_2$	$\mathcal{Q}_1$	$\mathcal{Q}_2$
~~5~~	6		
2	3		

$e_6 \Rightarrow ?$

$a = 4, b = 2$

classe 1 $s[2..5] = s[5..8] =$ OUDO

classe 2 $s[3..6] = s[6..9] =$ UDOU

classe ignorée $s[4..7] =$ DOUD

Extraction de motif

Algorithme KMR [3/3]

i	1	2	3	4	5	6	7	8	9
s	R	O	U	D	O	U	D	O	U
v_4	-	1	2	-	1	2	-	-	-

$e_6 \Rightarrow ?$

$a = 4, b = 2$

classe 1 $s[2..5] = s[5..8] = \texttt{OUDO}$

classe 2 $s[3..6] = s[6..9] = \texttt{UDOU}$

classe ignorée $s[4..7] = \texttt{DOUD}$

Extraction de motif

Algorithme KMR [3/3]

i	1	2	3	4	5	6	7	8	9
s	R	O	U	D	O	U	D	O	U
v_4	-	1	2	-	1	2	-	-	-

$e_6 \Rightarrow ?$

$a = 4, b = 2$

classe 1 $s[2..5] = s[5..8] =$ OUDO

classe 2 $s[3..6] = s[6..9] =$ UDOU

classe ignorée $s[4..7] =$ DOUD

Extraction de motif

Algorithme KMR [3/3]

i	1	2	3	4	5	6	7	8	9
s	R	O	U	D	O	U	D	O	U
v_4	-	1	2	-	1	2	-	-	-

$e_6 \Rightarrow ?$

$a = 4, b = 2$

classe 1 $s[2..5] = s[5..8] = \texttt{OUDO}$

classe 2 $s[3..6] = s[6..9] = \texttt{UDOU}$

classe ignorée $s[4..7] = \texttt{DOUD}$

Extraction de motif
Algorithme KMR [3/3]

i	1	2	3	4	5	6	7	8	9
s	R	O	U	D	O	U	D	O	U
v_6	-	-	-	-	-	-	-	-	-

$e_6 \Rightarrow 0$

classe ignorée $s[3..8] = \texttt{UDOUDO}$

Extraction de motif

Algorithme KMR [3/3]

i	1	2	3	4	5	6	7	8	9
s	R	O	U	D	O	U	D	O	U
v_4	-	1	2	-	1	2	-	-	-

$e_5 \Rightarrow ?$

$a = 4, b = 1$

classe 1 $s[2..5] = s[5..8] = \texttt{OUDO}$

classe 2 $s[3..6] = s[6..9] = \texttt{UDOU}$

classe ignorée $s[4..7] = \texttt{DOUD}$

Extraction de motif

Algorithme KMR [3/3]

i	1	2	3	4	5	6	7	8	9
s	R	O	U	D	O	U	D	O	U
v_4	-	1	2	-	1	2	-	-	-

$\mathcal{P}_1$	$\mathcal{P}_2$
~~5~~	6
2	3

$\mathcal{Q}_1$	$\mathcal{Q}_2$
	5

$e_5 \Rightarrow ?$

$a = 4, b = 1$

classe 1 $s[2..5] = s[5..8] = \texttt{OUDO}$

classe 2 $s[3..6] = s[6..9] = \texttt{UDOU}$

classe ignorée $s[4..7] = \texttt{DOUD}$

Extraction de motif
Algorithme KMR [3/3]

i	1	2	3	4	5	6	7	8	9
s	R	O	U	D	O	U	D	O	U
v_4	-	1	2	-	1	2	-	-	-

$e_5 \Rightarrow ?$

$a = 4, b = 1$

classe 1 $s[2..5] = s[5..8] =$ OUDO

classe 2 $s[3..6] = s[6..9] =$ UDOU

classe ignorée $s[4..7] =$ DOUD

Extraction de motif

Algorithme KMR [3/3]

i	1	2	3	4	5	6	7	8	9
s	R	O	U	D	O	U	D	O	U
v_4	-	1	2	-	1	2	-	-	-

$e_5 \Rightarrow$?

$a = 4, b = 1$

classe 1 $s[2..5] = s[5..8] =$ OUDO

classe 2 $s[3..6] = s[6..9] =$ UDOU

classe ignorée $s[4..7] =$ DOUD

Extraction de motif

Algorithme KMR [3/3]

i	1	2	3	4	5	6	7	8	9
s	R	O	U	D	O	U	D	O	U
v_4	-	1	2	-	1	2	-	-	-

$e_5 \Rightarrow ?$

$a = 4, b = 1$

classe 1 $s[2..5] = s[5..8] = \texttt{OUDO}$

classe 2 $s[3..6] = s[6..9] = \texttt{UDOU}$

classe ignorée $s[4..7] = \texttt{DOUD}$

Extraction de motif
Algorithme KMR [3/3]

i	1	2	3	4	5	6	7	8	9
s	R	O	U	D	O	U	D	O	U
v_4	-	1	2	-	1	2	-	-	-

$e_5 \Rightarrow ?$

$a = 4, b = 1$

classe 1 $s[2..5] = s[5..8] =$ OUDO

classe 2 $s[3..6] = s[6..9] =$ UDOU

classe ignorée $s[4..7] =$ DOUD

Extraction de motif
Algorithme KMR [3/3]

i	1	2	3	4	5	6	7	8	9
s	R	O	U	D	O	U	D	O	U
v_5	-	1	-	-	1	-	-	-	-

$e_5 \Rightarrow 1$

classe 1 $s[2..5] = s[5..9] = \texttt{OUDOU}$

Extraction de motif
Algorithme KMR [3/3]

i	1	2	3	4	5	6	7	8	9
s	R	O	U	D	O	U	D	O	U

$e_5 \Rightarrow 1$

classe 1 $s[2..5] = s[5..9] = \texttt{OUDOU}$

That' All, Folks ! ! !

HLIN608 – *Algorithmique du texte.*

Alban MANCHERON

Alignement visuel de séquences

Recherche de motif

Extraction de motif

Contexte

Quelques programmes [presque] naïfs

Algorithme KMR

Structures d'index

Hachage

Arbres

Automates

Tables

$\Rightarrow$ **Pré-traitement des données en vue de les exploiter plus efficacement**

Extraction de motif
Structures d'index [Hachage]

⇒ **Pré-traitement des données en vue de les exploiter plus efficacement**

⇒ Un facteur est un préfixe d'un suffixe vecteurs

⇒ **Pré-traitement des données en vue de les exploiter plus efficacement**

⇒ Un facteur est un préfixe d'un suffixe vecteurs

Une solution simple

Indexer les suffixes d'une séquence dans une table de hachage.

$\Rightarrow$ **Pré-traitement des données en vue de les exploiter plus efficacement**

$\Rightarrow$ Un facteur est un préfixe d'un suffixe vecteurs

Une solution simple

Indexer les suffixes d'une séquence dans une table de hachage.

Problèmes liés à la complexité

Espace et temps de construction en $O\left(n^2\right)$.

Définition (Arbre [enraciné])

$\equiv$ graphe non orienté connexe et sans cycle.

- Un **nœud** est un sommet de l'arbre
- La **racine** est un nœud particulier de l'arbre qui lui confère une orientation
  - $\Rightarrow$ La **racine** est le seul nœud sans prédécesseur
- Une **feuille** est un nœud de l'arbre sans successeurs
- Un nœud est dit **interne** s'il n'est ni la racine ni une feuille
- La **profondeur** d'un nœud est la longueur du chemin (unique) de la racine vers ce nœud.
- Le **niveau** d'un nœud est sa profondeur plus 1.
- La **hauteur** est égale à la plus grande profondeur de l'arbre
- La **largeur** est égale au plus grand nombre de nœud au même niveau de l'arbre

Définition (Arbre [enraciné])

≡ graphe non orienté connexe et sans cycle.

- Un **nœud** est un sommet de l'arbre
- La **racine** est un nœud particulier de l'arbre qui lui confère une orientation
  - ⇒ La **racine** est le seul nœud sans prédécesseur
- Une **feuille** est un nœud de l'arbre sans successeurs
- Un nœud est dit **interne** s'il n'est pas la racine ni une feuille
- La **profondeur** d'un nœud est la longueur du chemin (unique) de la racine vers ce nœud.
- Le **niveau** d'un nœud est sa profondeur plus 1.
- La **hauteur** est égale à la plus grande profondeur de l'arbre
- La **largeur** est égale au plus grand nombre de nœud au même niveau de l'arbre

Définition (Arbre [enraciné])

≡ graphe non orienté connexe et sans cycle.

- Un **nœud** est un sommet de l'arbre
- La **racine** est un nœud particulier de l'arbre qui lui confère une orientation
  - ⇒ La **racine** est le seul nœud sans prédécesseur
- Une **feuille** est un nœud de l'arbre sans successeurs
- Un nœud est dit **interne** s'il n'est ni la racine ni une feuille
- La **profondeur** d'un nœud est la longueur du chemin (unique) de la racine vers ce nœud.
- Le **niveau** d'un nœud est sa profondeur plus 1.
- La **hauteur** est égale à la plus grande profondeur de l'arbre
- La **largeur** est égale au plus grand nombre de nœud au même niveau de l'arbre

Définition (Arbre [enraciné])

≡ graphe non orienté connexe et sans cycle.

- Un **nœud** est un sommet de l'arbre
- La **racine** est un nœud particulier de l'arbre qui lui confère une orientation
  - ⇒ La **racine** est le seul nœud sans prédécesseur
- Une **feuille** est un nœud de l'arbre sans successeurs
- Un nœud est dit **interne** s'il n'est ni la racine ni une feuille
- La **profondeur** d'un nœud est la longueur du chemin (unique) de la racine vers ce nœud.
- Le **niveau** d'un nœud est sa profondeur plus 1.
- La **hauteur** est égale à la plus grande profondeur de l'arbre
- La **largeur** est égale au plus grand nombre de nœud au même niveau de l'arbre

Définition (Arbre [enraciné])

≡ graphe non orienté connexe et sans cycle.

- Un **nœud** est un sommet de l'arbre
- La **racine** est un nœud particulier de l'arbre qui lui confère une orientation
  - ⇒ La **racine** est le seul nœud sans prédécesseur
- Une **feuille** est un nœud de l'arbre sans successeurs
- Un nœud est dit **interne** s'il n'est ni la racine ni une feuille
- La **profondeur** d'un nœud est la longueur du chemin (unique) de la racine vers ce nœud.
- Le **niveau** d'un nœud est sa profondeur plus 1.
- La **hauteur** est égale à la plus grande profondeur de l'arbre
- La **largeur** est égale au plus grand nombre de nœud au même niveau de l'arbre

Définition (Arbre [enraciné])

≡ graphe non orienté connexe et sans cycle.

- Un **nœud** est un sommet de l'arbre
- La **racine** est un nœud particulier de l'arbre qui lui confère une orientation
  - ⇒ La **racine** est le seul nœud sans prédécesseur
- Une **feuille** est un nœud de l'arbre sans successeurs
- Un nœud est dit **interne** s'il n'est ni la racine ni une feuille
- La **profondeur** d'un nœud est la longueur du chemin (unique) de la racine vers ce nœud.
- Le **niveau** d'un nœud est sa profondeur plus 1.
- La **hauteur** est égale à la plus grande profondeur de l'arbre
- La **largeur** est égale au plus grand nombre de nœud au même niveau de l'arbre

HLIN608 – *Algorithmique du texte.*

Alban MANCHERON

Alignement visuel de séquences

Recherche de motif

Extraction de motif

Contexte

Quelques programmes [presque] naïfs

Algorithme KMR

Structures d'index

Hachage

Arbres

Automates

Tables

Définition (*Trie*)

≡ Arbre enraciné construit à partir d'un ensemble de mots (structure proposée en 1960 par Edward FREDKIN).

- Chaque chemin de la racine à une feuille de l'arbre étiquette un des mots de l'ensemble
  - Tous les mots de l'ensemble sont représentés
  - Aucun mot supplémentaire n'y est présent

⇒ Tous les mots de l'ensemble représenté qui partagent le même préfixe sont situés dans le sous-arbre issu du nœud associé à ce préfixe

- Le terme *Trie* provient du mot anglais *retrieval*

HLIN608 – *Algorithmique du texte.*

Alban MANCHERON

Alignement visuel de séquences

Recherche de motif

Extraction de motif

Contexte

Quelques programmes [presque] naïfs

Algorithme KMR

Structures d'index

Hachage

Arbres

Automates

Tables

Définition (*Trie*)

≡ Arbre enraciné construit à partir d'un ensemble de mots (structure proposée en 1960 par Edward FREDKIN).

- Chaque chemin de la racine à une feuille de l'arbre étiquette un des mots de l'ensemble
  - Tous les mots de l'ensemble sont représentés
  - Aucun mot supplémentaire n'y est présent

⇒ Tous les mots de l'ensemble représenté qui partagent le même préfixe sont situés dans le sous-arbre issu du nœud associé à ce préfixe

- Le terme *Trie* provient du mot anglais *retrieval*

Ajout d'un symbole marquant la fin du mot ⇒ cas où un mot et son préfixe sont dans l'ensemble à indexer…

Définition (*Trie*)

- ≡ Arbre enraciné construit à partir d'un ensemble de mots (structure proposée en 1960 par Edward FREDKIN)
- Chaque chemin de la racine à une feuille de l'arbre étiquette un des mots de l'ensemble
  - Tous les mots de l'ensemble sont représentés
  - Aucun mot supplémentaire n'y est présent
- ⇒ Tous les mots de l'ensemble représenté qui partagent le même préfixe sont situés dans le sous-arbre issu du nœud associé à ce préfixe
- Le terme *Trie* provient du mot anglais *retrieval*

Trie de l'ensemble:
{DOUX, DO, ROUX, ROUDOUDOU}

Ajout d'un symbole marquant la fin du mot ⇒ cas où un mot et son préfixe sont dans l'ensemble à indexer…

Définition (*Trie*)

≡ Arbre enraciné construit à partir d'un ensemble de mots (structure proposée en 1960 par Edward FREDKIN)

Trie de l'ensemble:
{DOUX, DO, ROUX, ROUDOUDOU}

- Chaque chemin de la racine à une feuille de l'arbre étiquette un des mots de l'ensemble
  - Tous les mots de l'ensemble sont représentés
  - Aucun mot supplémentaire n'y est présent

⇒ Tous les mots de l'ensemble représenté qui partagent le même préfixe sont situés dans le sous-arbre issu du nœud associé à ce préfixe

- Le terme *Trie* provient du mot anglais *retrieval*

Ajout d'un symbole marquant la fin du mot ⇒ cas où un mot et son préfixe sont dans l'ensemble à indexer…

Définition (*Trie*)

≡ Arbre enraciné construit à partir d'un ensemble de mots (structure proposée en 1960 par Edward FREDKIN)

- Chaque chemin de la racine à une feuille de l'arbre étiquette un des mots de l'ensemble
  - Tous les mots de l'ensemble sont représentés
  - Aucun mot supplémentaire n'y est présent

⇒ Tous les mots de l'ensemble représenté qui partagent le même préfixe sont situés dans le sous-arbre issu du nœud associé à ce préfixe

- Le terme *Trie* provient du mot anglais *retrieval*

Trie de l'ensemble:
{DOUX, DO, ROUX, ROUDOUDOU}

Ajout d'un symbole marquant la fin du mot ⇒ cas où un mot et son préfixe sont dans l'ensemble à indexer…

Définition (*Trie*)

≡ Arbre enraciné construit à partir d'un ensemble de mots (structure proposée en 1960 par Edward FREDKIN)

- Chaque chemin de la racine à une feuille de l'arbre étiquette un des mots de l'ensemble
  - Tous les mots de l'ensemble sont représentés
  - Aucun mot supplémentaire n'y est présent

⇒ Tous les mots de l'ensemble représenté qui partagent le même préfixe sont situés dans le sous-arbre issu du nœud associé à ce préfixe

- Le terme *Trie* provient du mot anglais *retrieval*

Trie de l'ensemble:
{DOUX, DO, ROUX, ROUDOUDOU}

Ajout d'un symbole marquant la fin du mot ⇒ cas où un mot et son préfixe sont dans l'ensemble à indexer…

Définition (*Trie*)

Trie de l'ensemble:
{DOUX, DO, ROUX, ROUDOUDOU}

≡ Arbre enraciné construit à partir d'un ensemble de mots (structure proposée en 1960 par Edward FREDKIN)

- Chaque chemin de la racine à une feuille de l'arbre étiquette un des mots de l'ensemble
  - Tous les mots de l'ensemble sont représentés
  - Aucun mot supplémentaire n'y est présent

⇒ Tous les mots de l'ensemble représenté qui partagent le même préfixe sont situés dans le sous-arbre issu du nœud associé à ce préfixe

- Le terme *Trie* provient du mot anglais *retrieval*

Ajout d'un symbole marquant la fin du mot ⇒ cas où un mot et son préfixe sont dans l'ensemble à indexer…

Trie d'un ensemble de n mots

- Nombre de feuilles ?
- Hauteur ?

HLIN608 – *Algorithmique du texte.*

Alban MANCHERON

Alignement visuel de séquences

Recherche de motif

Extraction de motif

Contexte

Quelques programmes [presque] naïfs

Algorithme KMR

Structures d'index

Hachage

Arbres

Automates

Tables

Extraction de motif
Structures d'index [Arbres]

Trie d'un ensemble de n mots

- Nombre de feuilles ?
  - $\Rightarrow$ n feuilles
- Hauteur ?

Alignement visuel de séquences

Recherche de motif

Extraction de motif

Contexte

Quelques programmes [presque] naïfs

Algorithme KMR

Structures d'index

Hachage

Arbres

Automates

Tables

Extraction de motif
Structures d'index [Arbres]

Trie d'un ensemble de n mots

- Nombre de feuilles ?
  - $\Rightarrow$ n feuilles
- Hauteur ?
  - $\Rightarrow$ longueur du plus grand mot de l'ensemble

Alignement visuel de séquences

Recherche de motif

Extraction de motif

Contexte

Quelques programmes [presque] naïfs

Algorithme KMR

Structures d'index

Hachage

Arbres

Automates

Tables

HLIN608 – *Algorithmique du texte.*

Alban MANCHERON

Alignement visuel de séquences

Recherche de motif

Extraction de motif

Contexte

Quelques programmes [presque] naïfs

Algorithme KMR

Structures d'index

Hachage

Arbres

Automates

Tables

Définition (*Patricia Trie*)

$\equiv$ version compacte d'un *Trie* (structure proposée en 1968 par Donald MORRISON).

- seuls la racine, les nœuds internes ayant au moins deux fils, ainsi que les feuilles sont conservés

= *Practical Algorithm To Retrieve Information Coded In Alphanumeric Trie*

Définition (*Patricia Trie*)

= version compacte d'un *Trie* (structure proposée en 1968 par Donald MORRISON)

- seuls la racine, les nœuds internes ayant au moins deux fils, ainsi que les feuilles sont conservés

= *Practical Algorithm To Retrieve Information Coded In Alphanumeric Trie*

Patricia Trie de l'ensemble:
{DOUX, DO, ROUX, ROUDOUDOU}

•

Définition (*Patricia Trie*)

= version compacte d'un *Trie* (structure proposée en 1968 par Donald MORRISON)

- seuls la racine, les nœuds internes ayant au moins deux fils, ainsi que les feuilles sont conservés

= *Practical Algorithm To Retrieve Information Coded In Alphanumeric Trie*

Patricia Trie de l'ensemble:
{DOUX, DO, ROUX, ROUDOUDOU}

DOUX$

Définition (*Patricia Trie*)

= version compacte d'un *Trie* (structure proposée en 1968 par Donald MORRISON)

- seuls la racine, les nœuds internes ayant au moins deux fils, ainsi que les feuilles sont conservés

= *Practical Algorithm To Retrieve Information Coded In Alphanumeric Trie*

Patricia Trie de l'ensemble:
{DOUX, DO, ROUX, ROUDOUDOU}

Définition (*Patricia Trie*)

= version compacte d'un *Trie* (structure proposée en 1968 par Donald MORRISON)

- seuls la racine, les nœuds internes ayant au moins deux fils, ainsi que les feuilles sont conservés

= *Practical Algorithm To Retrieve Information Coded In Alphanumeric Trie*

Patricia Trie de l'ensemble:
{DOUX, DO, ROUX, ROUDOUDOU}

Définition (*Patricia Trie*)

= version compacte d'un *Trie* (structure proposée en 1968 par Donald MORRISON)

- seuls la racine, les nœuds internes ayant au moins deux fils, ainsi que les feuilles sont conservés

= *Practical Algorithm To Retrieve Information Coded In Alphanumeric Trie*

Patricia Trie de l'ensemble:
{DOUX, DO, ROUX, ROUDOUDOU}

HLIN608 – *Algorithmique du texte.*

Alban MANCHERON

Alignement visuel de séquences

Recherche de motif

Extraction de motif

Contexte

Quelques programmes [presque] naïfs

Algorithme KMR

Structures d'index

Hachage

Arbres

Automates

Tables

Définition (Arbre [compact] des suffixes)

≡ *[Patricia] Trie* des suffixes d'un mot

- construction linéaire (temps/espace)
  - ⇒ Peter WEINER en 1973
- construction linéaire *on-line*
  - ⇒ Esko UKKONEN en 1995

Définition (Arbre [compact] des suffixes)

≡ *[Patricia] Trie* des suffixes d'un mot

- construction linéaire *off-line*
  - ⇒ Peter WEINER en 1973
- construction linéaire *on-line*
  - ⇒ Esko UKKONEN en 1995

Arbre [compact] des suffixes du mot
ROUDOUDOU

Définition (Arbre [compact] des suffixes)

≡ *[Patricia] Trie* des suffixes d'un mot

- construction linéaire *off-line*
  - ⇒ Peter WEINER en 1973
- construction linéaire *on-line*
  - ⇒ Esko UKKONEN en 1995

Arbre [compact] des suffixes du mot ROUDOUDOU

Définition (Arbre [compact] des suffixes)

≡ *[Patricia] Trie* des suffixes d'un mot

- construction linéaire *off-line*
  - ⇒ Peter WEINER en 1973
- construction linéaire *on-line*
  - ⇒ Esko UKKONEN en 1995

Arbre [compact] des suffixes du mot ROUDOUDOU

Définition (Arbre [compact] des suffixes)

≡ *[Patricia] Trie* des suffixes d'un mot

- construction linéaire *off-line*
  - ⇒ Peter WEINER en 1973
- construction linéaire *on-line*
  - ⇒ Esko UKKONEN en 1995

Arbre [compact] des suffixes du mot ROUDOUDOU

Définition (Arbre [compact] des suffixes)

≡ *[Patricia] Trie* des suffixes d'un mot

- construction linéaire *off-line*
  - ⇒ Peter WEINER en 1973
- construction linéaire *on-line*
  - ⇒ Esko UKKONEN en 1995

Arbre [compact] des suffixes du mot ROUDOUDOU

Définition (Arbre [compact] des suffixes)

≡ *[Patricia] Trie* des suffixes d'un mot

- construction linéaire *offline*
  - ⇒ Peter WEINER en 1973
- construction linéaire *online*
  - ⇒ Esko UKKONEN en 1995

Arbre [compact] des suffixes du mot ROUDOUDOU

Définition (Arbre [compact] des suffixes)

≡ *[Patricia] Trie* des suffixes d'un mot

- construction linéaire *off-line*
  - ⇒ Peter WEINER en 1973
- construction linéaire *on-line*
  - ⇒ Esko UKKONEN en 1995

Arbre [compact] des suffixes du mot ROUDOUDOU

Définition (Arbre [compact] des suffixes)

≡ *[Patricia] Trie* des suffixes d'un mot

- construction linéaire *off-line*
  - ⇒ Peter WEINER en 1973
- construction linéaire *on-line*
  - ⇒ Esko UKKONEN en 1995

Arbre [compact] des suffixes du mot ROUDOUDOU

Définition (Arbre [compact] des suffixes)

≡ *[Patricia] Trie* des suffixes d'un mot

- construction linéaire *off-line*
  - ⇒ Peter WEINER en 1973
- construction linéaire *on-line*
  - ⇒ Esko UKKONEN en 1995

Arbre [compact] des suffixes du mot ROUDOUDOU

Définition (Arbre [compact] des suffixes)

≡ *[Patricia] Trie* des suffixes d'un mot

- construction linéaire *off-line*
  - ⇒ Peter WEINER en 1973
- construction linéaire *on-line*
  - ⇒ Esko UKKONEN en 1995

Arbre [compact] des suffixes du mot ROUDOUDOU

Définition (Arbre [compact] des suffixes)

≡ *[Patricia] Trie* des suffixes d'un mot

- construction linéaire *off-line*
  - ⇒ Peter WEINER en 1973
- construction linéaire *on-line*
  - ⇒ Esko UKKONEN en 1995

Arbre [compact] des suffixes du mot ROUDOUDOU

Définition (Automate à états finis)

$\equiv$ quintuplet $(\mathcal{Q}, \mathcal{I}, \mathcal{F}, \Sigma, \delta)$

HLIN608 – *Algorithmique du texte.*

Alban MANCHERON

Alignement visuel de séquences

Recherche de motif

Extraction de motif

Contexte

Quelques programmes [presque] naïfs

Algorithme KMR

Structures d'index

Hachage

Arbres

Automates

Tables

Définition (Automate à états finis)

$\equiv$ quintuplet $(\mathcal{Q}, \mathcal{I}, \mathcal{F}, \Sigma, \delta)$

- $\mathcal{Q}$: ensemble fini d'états

HLIN608 – *Algorithmique du texte.*

Alban MANCHERON

Alignement visuel de séquences

Recherche de motif

Extraction de motif

Contexte

Quelques programmes [presque] naïfs

Algorithme KMR

Structures d'index

Hachage

Arbres

Automates

Tables

HLIN608 – *Algorithmique du texte.*

Alban MANCHERON

Alignement visuel de séquences

Recherche de motif

Extraction de motif

Contexte

Quelques programmes [presque] naïfs

Algorithme KMR

Structures d'index

Hachage

Arbres

Automates

Tables

Définition (Automate à états finis)

$\equiv$ quintuplet $(\mathcal{Q}, \mathcal{I}, \mathcal{F}, \Sigma, \delta)$

- $\mathcal{Q}$: ensemble fini d'états
- $\mathcal{I}(\subseteq Q)$: ensemble des états initiaux

Définition (Automate à états finis)

$\equiv$ quintuplet $(\mathcal{Q}, \mathcal{I}, \mathcal{F}, \Sigma, \delta)$

- $\mathcal{Q}$: ensemble fini d'états
- $\mathcal{I}(\subseteq Q)$: ensemble des états initiaux
- $\mathcal{F}(\subseteq Q)$: ensemble des états finaux

HLIN608 – *Algorithmique du texte.*

Alban MANCHERON

Alignement visuel de séquences

Recherche de motif

Extraction de motif

Contexte

Quelques programmes [presque] naïfs

Algorithme KMR

Structures d'index

Hachage

Arbres

Automates

Tables

Définition (Automate à états finis)

$\equiv$ quintuplet $(\mathcal{Q}, \mathcal{I}, \mathcal{F}, \Sigma, \delta)$

- $\mathcal{Q}$: ensemble fini d'états
- $\mathcal{I}(\subseteq Q)$: ensemble des états initiaux
- $\mathcal{F}(\subseteq Q)$: ensemble des états finaux
- Σ : alphabet

Alignement visuel de séquences

Recherche de motif

Extraction de motif

Contexte

Quelques programmes [presque] naïfs

Algorithme KMR

Structures d'index

Hachage

Arbres

Automates

Tables

Définition (Automate à états finis)

$\equiv$ quintuplet $(\mathcal{Q}, \mathcal{I}, \mathcal{F}, \Sigma, \delta)$

- $\mathcal{Q}$: ensemble fini d'états
- $\mathcal{I}(\subseteq Q)$: ensemble des états initiaux
- $\mathcal{F}(\subseteq Q)$: ensemble des états finaux
- Σ : alphabet
- $\delta \in \mathcal{P}(\mathcal{Q} \times \Sigma \times \mathcal{Q})$: relation de transition

HLIN608 – *Algorithmique du texte.*

Alban MANCHERON

Alignement visuel de séquences

Recherche de motif

Extraction de motif

Contexte

Quelques programmes [presque] naïfs

Algorithme KMR

Structures d'index

Hachage

Arbres

Automates

Tables

Définition (Automate à états finis)

$\equiv$ quintuplet $(\mathcal{Q}, \mathcal{I}, \mathcal{F}, \Sigma, \delta)$

- $\mathcal{Q}$: ensemble fini d'états
- $\mathcal{I}(\subseteq Q)$: ensemble des états initiaux
- $\mathcal{F}(\subseteq Q)$: ensemble des états finaux
- Σ : alphabet
- $\delta \in \mathcal{P}(\mathcal{Q} \times \Sigma \times \mathcal{Q})$: relation de transition

Propriétés sympathiques (ou pas !)

$\Rightarrow$ Déterminisme

Alignement visuel de séquences

Recherche de motif

Extraction de motif

Contexte

Quelques programmes [presque] naïfs

Algorithme KMR

Structures d'index

Hachage

Arbres

Automates

Tables

Définition (Automate à états finis)

$\equiv$ quintuplet $(\mathcal{Q}, \mathcal{I}, \mathcal{F}, \Sigma, \delta)$

- $\mathcal{Q}$: ensemble fini d'états
- $\mathcal{I}(\subseteq Q)$: ensemble des états initiaux
- $\mathcal{F}(\subseteq Q)$: ensemble des états finaux
- Σ : alphabet
- $\delta \in \mathcal{P}(\mathcal{Q} \times \Sigma \times \mathcal{Q})$: relation de transition

Propriétés sympathiques (ou pas !)

$\Rightarrow$ Déterminisme

$\Rightarrow$ Minimalité

Définition (Automate à états finis)

$\equiv$ quintuplet $(\mathcal{Q}, \mathcal{I}, \mathcal{F}, \Sigma, \delta)$

- $\mathcal{Q}$: ensemble fini d'états
- $\mathcal{I}(\subseteq Q)$: ensemble des états initiaux
- $\mathcal{F}(\subseteq Q)$: ensemble des états finaux
- Σ : alphabet
- $\delta \in \mathcal{P}(\mathcal{Q} \times \Sigma \times \mathcal{Q})$: relation de transition

Propriétés sympathiques (ou pas !)

$\Rightarrow$ Déterminisme

$\Rightarrow$ Cyclicité

$\Rightarrow$ Minimalité

Définition (Automate à états finis)

$\equiv$ quintuplet $(\mathcal{Q}, \mathcal{I}, \mathcal{F}, \Sigma, \delta)$

- $\mathcal{Q}$: ensemble fini d'états
- $\mathcal{I}(\subseteq Q)$: ensemble des états initiaux
- $\mathcal{F}(\subseteq Q)$: ensemble des états finaux
- Σ : alphabet
- $\delta \in \mathcal{P}(\mathcal{Q} \times \Sigma \times \mathcal{Q})$: relation de transition

Propriétés sympathiques (ou pas !)

$\Rightarrow$ Déterminisme

$\Rightarrow$ Cyclicité

$\Rightarrow$ Minimalité

$\Rightarrow$ Homogénéité

Alignement visuel de séquences

Recherche de motif

Extraction de motif

Contexte

Quelques programmes [presque] naïfs

Algorithme KMR

Structures d'index

Hachage

Arbres

Automates

Tables

Définition (Automate à états finis)

$\equiv$ quintuplet $(\mathcal{Q}, \mathcal{I}, \mathcal{F}, \Sigma, \delta)$

- $\mathcal{Q}$: ensemble fini d'états
- $\mathcal{I}(\subseteq Q)$: ensemble des états initiaux
- $\mathcal{F}(\subseteq Q)$: ensemble des états finaux
- Σ : alphabet
- $\delta \in \mathcal{P}(\mathcal{Q} \times \Sigma \times \mathcal{Q})$: relation de transition

Propriétés sympathiques (ou pas !)

$\Rightarrow$ Déterminisme

$\Rightarrow$ Cyclicité

$\Rightarrow$ Minimalité

$\Rightarrow$ Homogénéité

Automate $\Leftrightarrow$ Langage

Définition (Automate à états finis)

$\equiv$ quintuplet $(\mathcal{Q}, \mathcal{I}, \mathcal{F}, \Sigma, \delta)$

- $\mathcal{Q}$: ensemble fini d'états
- $\mathcal{I}(\subseteq Q)$: ensemble des états initiaux
- $\mathcal{F}(\subseteq Q)$: ensemble des états finaux
- Σ : alphabet
- $\delta \in \mathcal{P}(\mathcal{Q} \times \Sigma \times \mathcal{Q})$: relation de transition

Propriétés sympathiques (ou pas !)

$\Rightarrow$ Déterminisme

$\Rightarrow$ Cyclicité

$\Rightarrow$ Minimalité

$\Rightarrow$ Homogénéité

Automate $\Leftrightarrow$ Langage

- Automates des facteurs/suffixes

Définition (Automate à états finis)

$\equiv$ quintuplet $(\mathcal{Q}, \mathcal{I}, \mathcal{F}, \Sigma, \delta)$

- $\mathcal{Q}$: ensemble fini d'états
- $\mathcal{I}(\subseteq Q)$: ensemble des états initiaux
- $\mathcal{F}(\subseteq Q)$: ensemble des états finaux
- Σ : alphabet
- $\delta \in \mathcal{P}(\mathcal{Q} \times \Sigma \times \mathcal{Q})$: relation de transition

Propriétés sympathiques (ou pas !)

$\Rightarrow$ Déterminisme

$\Rightarrow$ Cyclicité

$\Rightarrow$ Minimalité

$\Rightarrow$ Homogénéité

Automate $\Leftrightarrow$ Langage

- Automates des facteurs/suffixes
- Automates des sous-séquences

Définition (Automate à états finis)

$\equiv$ quintuplet $(\mathcal{Q}, \mathcal{I}, \mathcal{F}, \Sigma, \delta)$

- $\mathcal{Q}$: ensemble fini d'états
- $\mathcal{I}(\subseteq Q)$: ensemble des états initiaux
- $\mathcal{F}(\subseteq Q)$: ensemble des états finaux
- Σ : alphabet
- $\delta \in \mathcal{P}(\mathcal{Q} \times \Sigma \times \mathcal{Q})$: relation de transition

Propriétés sympathiques (ou pas !)

$\Rightarrow$ Déterminisme

$\Rightarrow$ Cyclicité

$\Rightarrow$ Minimalité

$\Rightarrow$ Homogénéité

Automate $\Leftrightarrow$ Langage

- Automates des facteurs/suffixes
- Automates des sous-séquences
- Algorithmes (AHO-CORASIK, ...)

Définition (Automate à états finis)

≡ quintuplet $(\mathcal{Q}, \mathcal{I}, \mathcal{F}, \Sigma, \delta)$

- $\mathcal{Q}$: ensemble fini d'états
- $\mathcal{I}(\subseteq \mathcal{Q})$: ensemble des états initiaux
- $\mathcal{F}(\subseteq \mathcal{Q})$: ensemble des états finaux
- Σ : alphabet
- $\delta \in \mathcal{P}(\mathcal{Q} \times \Sigma \times \mathcal{Q})$: relation de transition

Automate des suffixes du mot ROUDOUDOU

Propriétés sympathiques (ou pas !)

⇒ Déterminisme

⇒ Minimalité

⇒ Cyclicité

⇒ Homogénéité

Automate ⇔ Langage

- Automates des facteurs/suffixes
- Automates des sous-séquences
- Algorithmes (AHO-CORASIK, . . .)

Définition (Automate à états finis)

≡ quintuplet $(\mathcal{Q}, \mathcal{I}, \mathcal{F}, \Sigma, \delta)$

- $\mathcal{Q}$: ensemble fini d'états
- $\mathcal{I}(\subseteq \mathcal{Q})$: ensemble des états initiaux
- $\mathcal{F}(\subseteq \mathcal{Q})$: ensemble des états finaux
- Σ : alphabet
- $\delta \in (\mathcal{Q} \times \Sigma \times \mathcal{Q})$: relation de transition

Automate des suffixes du mot
ROUDOUDOU

Propriétés sympathiques (ou pas !)

⇒ Déterminisme ⇒ Minimalité

⇒ Cyclicité ⇒ Homogénéité

Automate ⇔ Langage

- Automates des facteurs/suffixes
- Automates des sous-séquences
- Algorithmes (AHO-CORASIK, . . .)

Définition (Automate à états finis)

≡ quintuplet $(\mathcal{Q}, \mathcal{I}, \mathcal{F}, \Sigma, \delta)$

- $\mathcal{Q}$: ensemble fini d'états
- $\mathcal{I}(\subseteq \mathcal{Q})$: ensemble des états initiaux
- $\mathcal{F}(\subseteq \mathcal{Q})$: ensemble des états finaux
- Σ : alphabet
- $\delta \in (\mathcal{Q} \times \Sigma \times \mathcal{Q})$: relation de transition

Automate des suffixes du mot
ROUDOUDOU

Propriétés sympathiques (ou pas !)

⇒ Déterminisme

⇒ Minimalité

⇒ Cyclicité

⇒ Homogénéité

Automate ⇔ Langage

- Automates des facteurs/suffixes
- Automates des sous-séquences
- Algorithmes (AHO-CORASIK, . . .)

Définition (Automate à états finis)

≡ quintuplet $(\mathcal{Q}, \mathcal{I}, \mathcal{F}, \Sigma, \delta)$

- $\mathcal{Q}$: ensemble fini d'états
- $\mathcal{I}(\subseteq \mathcal{Q})$: ensemble des états initiaux
- $\mathcal{F}(\subseteq \mathcal{Q})$: ensemble des états finaux
- Σ : alphabet
- $\delta \in (\mathcal{Q} \times \Sigma \times \mathcal{Q})$: relation de transition

Automate des suffixes du mot ROUDOUDOU

Propriétés sympathiques (ou pas !)

⇒ Déterminisme ⇒ Minimalité

⇒ Cyclicité ⇒ Homogénéité

Automate ⇔ Langage

- Automates des facteurs/suffixes
- Automates des sous-séquences
- Algorithmes (AHO-CORASIK, . . .)

Définition (Automate à états finis)

≡ quintuplet $(\mathcal{Q}, \mathcal{I}, \mathcal{F}, \Sigma, \delta)$

- $\mathcal{Q}$: ensemble fini d'états
- $\mathcal{I}(\subseteq \mathcal{Q})$: ensemble des états initiaux
- $\mathcal{F}(\subseteq \mathcal{Q})$: ensemble des états finaux
- Σ : alphabet
- $\delta \in (\mathcal{Q} \times \Sigma \times \mathcal{Q})$: relation de transition

Automate des suffixes du mot ROUDOUDOU

Propriétés sympathiques (ou pas !)

⇒ Déterminisme ⇒ Minimalité

⇒ Cyclicité ⇒ Homogénéité

Automate ⇔ Langage

- Automates des facteurs/suffixes
- Automates des sous-séquences
- Algorithmes (AHO-CORASIK, . . .)

Définition (Automate à états finis)

≡ quintuplet $(\mathcal{Q}, \mathcal{I}, \mathcal{F}, \Sigma, \delta)$

- $\mathcal{Q}$: ensemble fini d'états
- $\mathcal{I}(\subseteq \mathcal{Q})$: ensemble des états initiaux
- $\mathcal{F}(\subseteq \mathcal{Q})$: ensemble des états finaux
- Σ : alphabet
- $\delta \in (\mathcal{Q} \times \Sigma \times \mathcal{Q})$: relation de transition

Automate des suffixes du mot ROUDOUDOU

Propriétés sympathiques (ou pas !)

⇒ Déterminisme ⇒ Minimalité

⇒ Cycles ⇒ Homogénéité

Automate ⇔ Langage

- Automates des facteurs/suffixes
- Automates des sous-séquences
- Algorithmes (AHO-CORASIK, . . .)

Définition (Automate à états finis)

≡ quintuplet $(\mathcal{Q}, \mathcal{I}, \mathcal{F}, \Sigma, \delta)$

- $\mathcal{Q}$: ensemble fini d'états
- $\mathcal{I}(\subseteq \mathcal{Q})$: ensemble des états initiaux
- $\mathcal{F}(\subseteq \mathcal{Q})$: ensemble des états finaux
- Σ : alphabet
- $\delta \in (\mathcal{Q} \times \Sigma \times \mathcal{Q})$: relation de transition

Automate des suffixes du mot ROUDOUDOU

Propriétés sympathiques (ou pas !)

⇒ Déterminisme ⇒ Minimalité

⇒ Cycles ⇒ Homogénéité

Automate ⇔ Langage

- Automates des facteurs/suffixes
- Automates des sous-séquences
- Algorithmes (AHO-CORASIK, . . .)

Définition (Automate à états finis)

≡ quintuplet $(\mathcal{Q}, \mathcal{I}, \mathcal{F}, \Sigma, \delta)$

- $\mathcal{Q}$: ensemble fini d'états
- $\mathcal{I}(\subseteq \mathcal{Q})$: ensemble des états initiaux
- $\mathcal{F}(\subseteq \mathcal{Q})$: ensemble des états finaux
- Σ : alphabet
- $\delta \in (\mathcal{Q} \times \Sigma \times \mathcal{Q})$: relation de transition

Automate des suffixes du mot ROUDOUDOU

Propriétés sympathiques (ou pas !)

⇒ Déterminisme ⇒ Minimalité

⇒ Cycles ⇒ Homogénéité

Automate ⇔ Langage

- Automates des facteurs/suffixes
- Automates des sous-séquences
- Algorithmes (AHO-CORASIK, . . .)

Définition (Automate à états finis)

≡ quintuplet $(\mathcal{Q}, \mathcal{I}, \mathcal{F}, \Sigma, \delta)$

- $\mathcal{Q}$: ensemble fini d'états
- $\mathcal{I}(\subseteq \mathcal{Q})$: ensemble des états initiaux
- $\mathcal{F}(\subseteq \mathcal{Q})$: ensemble des états finaux
- Σ : alphabet
- $\delta \in (\mathcal{Q} \times \Sigma \times \mathcal{Q})$: relation de transition

Propriétés sympathiques (ou pas !)

⇒ Déterminisme ⇒ Minimalité

⇒ Cycles ⇒ Homogénéité

Automate ⇔ Langage

Automate des suffixes du mot ROUDOUDOU

- Automates des facteurs/suffixes
- Automates des sous-séquences
- Algorithmes (AHO-CORASIK, . . .)

Définition (Automate à états finis)

≡ quintuplet $(\mathcal{Q}, \mathcal{I}, \mathcal{F}, \Sigma, \delta)$

- $\mathcal{Q}$: ensemble fini d'états
- $\mathcal{I}(\subseteq \mathcal{Q})$: ensemble des états initiaux
- $\mathcal{F}(\subseteq \mathcal{Q})$: ensemble des états finaux
- Σ : alphabet
- $\delta \in (\mathcal{Q} \times \Sigma \times \mathcal{Q})$: relation de transition

Propriétés sympathiques (ou pas !)

⇒ Déterminisme ⇒ Minimalité

⇒ Cycles ⇒ Homogénéité

Automate ⇔ Langage

Automate des suffixes du mot ROUDOUDOU

- Automates des facteurs/suffixes
- Automates des sous-séquences
- Algorithmes (AHO-CORASIK, . . .)

Définition (Automate à états finis)

≡ quintuplet $(\mathcal{Q}, \mathcal{I}, \mathcal{F}, \Sigma, \delta)$

- $\mathcal{Q}$: ensemble fini d'états
- $\mathcal{I}(\subseteq \mathcal{Q})$: ensemble des états initiaux
- $\mathcal{F}(\subseteq \mathcal{Q})$: ensemble des états finaux
- Σ : alphabet
- $\delta \in (\mathcal{Q} \times \Sigma \times \mathcal{Q})$: relation de transition

Automate des suffixes du mot ROUDOUDOU

Propriétés sympathiques (ou pas !)

⇒ Déterminisme ⇒ Minimalité

⇒ Cycles ⇒ Homogénéité

Automate ⇔ Langage

- Automates des facteurs/suffixes
- Automates des sous-séquences
- Algorithmes (AHO-CORASIK, . . .)

HLIN608 – *Algorithmique du texte.*

Alban MANCHERON

Alignement visuel de séquences

Recherche de motif

Extraction de motif

Contexte

Quelques programmes [presque] naïfs

Algorithme KMR

Structures d'index

Hachage

Arbres

Automates

Tables

Définition (Table des suffixes)

$\equiv$ tableau des positions des suffixes d'un mot dans l'ordre lexicographique.

HLIN608 – *Algorithmique du texte.*

Alban MANCHERON

Alignement visuel de séquences

Recherche de motif

Extraction de motif

Contexte

Quelques programmes [presque] naïfs

Algorithme KMR

Structures d'index

Hachage

Arbres

Automates

Tables

Définition (Table des suffixes)

$\equiv$ tableau des positions des suffixes d'un mot dans l'ordre lexicographique.

+

Extraction de motif
Structures d'index [Tables]

Alignement visuel de séquences
Recherche de motif
Extraction de motif
Contexte
Quelques programmes [presque] naïfs
Algorithme KMR
Structures d'index
Hachage
Arbres
Automates
Tables

Définition (Table des suffixes)

$\equiv$ tableau des positions des suffixes d'un mot dans l'ordre lexicographique.

$+$

Définition (Table des plus long préfixes communs)

$\equiv$ longueur du plus long préfixe commun entre deux suffixes consécutifs de la table des suffixes.

$\Rightarrow$ Amélioration des performances de recherche

Définition (Table des suffixes)

≡ tableau des positions des suffixes d'un mot dans l'ordre lexicographique.

Table des suffixes du mot ROUDOUDOU$

+

Définition (Table des plus long préfixes communs)

≡ longueur du plus long préfixe commun entre deux suffixes consécutifs de la table des suffixes.

⇒ Amélioration des performances de recherche

Définition (Table des suffixes)

≡ tableau des positions des suffixes d'un mot dans l'ordre lexicographique.

Table des suffixes du mot
ROUDOUDOU$

1	*ROUDOUDOU$*

Définition (Table des plus long préfixes communs)

≡ longueur du plus long préfixe commun entre deux suffixes consécutifs de la table des suffixes.

⇒ Amélioration des performances de recherche

Définition (Table des suffixes)

≡ tableau des positions des suffixes d'un mot dans l'ordre lexicographique.

Table des suffixes du mot ROUDOUDOU$

1	*ROUDOUDOU$*
2	*OUDOUDOU$*

+

Définition (Table des plus long préfixes communs)

≡ longueur du plus long préfixe commun entre deux suffixes consécutifs de la table des suffixes.

⇒ Amélioration des performances de recherche

Définition (Table des suffixes)

≡ tableau des positions des suffixes d'un mot dans l'ordre lexicographique.

Table des suffixes du mot ROUDOUDOU$

1	*ROUDOUDOU$*
2	*OUDOUDOU$*
3	*UDOUDOU$*

+

Définition (Table des plus long préfixes communs)

≡ longueur du plus long préfixe commun entre deux suffixes consécutifs de la table des suffixes.

⇒ Amélioration des performances de recherche

Définition (Table des suffixes)

≡ tableau des positions des suffixes du mot dans l'ordre lexicographique.

Table des suffixes du mot ROUDOUDOU$

1	*ROUDOUDOU$*
2	*OUDOUDOU$*
3	*UDOUDOU$*
4	*DOUDOU$*

+

Définition (Table des plus long préfixes communs)

≡ longueur du plus long préfixe commun entre deux suffixes consécutifs de la table des suffixes.

⇒ Amélioration des performances de recherche

Définition (Table des suffixes)

≡ tableau des positions des suffixes d'un mot dans l'ordre lexicographique.

Table des suffixes du mot ROUDOUDOU$

1	*ROUDOUDOU$*
2	*OUDOUDOU$*
3	*UDOUDOU$*
4	*DOUDOU$*
5	*OUDOU$*

+

Définition (Table des plus long préfixes communs)

≡ longueur du plus long préfixe commun entre deux suffixes consécutifs de la table des suffixes.

⇒ Amélioration des performances de recherche

Définition (Table des suffixes)

≡ tableau des positions des suffixes d'un mot dans l'ordre lexicographique.

+

Définition (Table des plus long préfixes communs)

≡ longueur du plus long préfixe commun entre deux suffixes consécutifs de la table des suffixes.

→ Amélioration des performances de recherche

Table des suffixes du mot ROUDOUDOU$

1	*ROUDOUDOU$*
2	*OUDOUDOU$*
3	*UDOUDOU$*
4	*DOUDOU$*
5	*OUDOU$*
6	*UDOU$*

Définition (Table des suffixes)

≡ tableau des positions des suffixes d'un mot dans l'ordre lexicographique.

+

Définition (Table des plus long préfixes communs)

≡ longueur du plus long préfixe commun entre deux suffixes consécutifs de la table des suffixes.

→ Amélioration des performances de recherche

Table des suffixes du mot ROUDOUDOU$

1	*ROUDOUDOU$*
2	*OUDOUDOU$*
3	*UDOUDOU$*
4	*DOUDOU$*
5	*OUDOU$*
6	*UDOU$*
7	*DOU$*

Définition (Table des suffixes)

≡ tableau des positions des suffixes d'un mot dans l'ordre lexicographique.

+

Définition (Table des plus long préfixes communs)

≡ longueur du plus long préfixe commun entre deux suffixes consécutifs de la table des suffixes.

⇒ Amélioration des performances de recherche

Table des suffixes du mot ROUDOUDOU$

1	*ROUDOUDOU$*
2	*OUDOUDOU$*
3	*UDOUDOU$*
4	*DOUDOU$*
5	*OUDOU$*
6	*UDOU$*
7	*DOU$*
8	*OU$*

Définition (Table des suffixes)

≡ tableau des positions des suffixes d'un mot dans l'ordre lexicographique.

+

Définition (Table des plus long préfixes communs)

≡ longueur du plus long préfixe commun entre deux suffixes consécutifs de la table des suffixes.

⇒ Amélioration des performances de recherche

Table des suffixes du mot ROUDOUDOU$

1	*ROUDOUDOU$*
2	*OUDOUDOU$*
3	*UDOUDOU$*
4	*DOUDOU$*
5	*OUDOU$*
6	*UDOU$*
7	*DOU$*
8	*OU$*
9	*U$*

Définition (Table des suffixes)

≡ tableau des positions des suffixes d'un mot dans l'ordre lexicographique.

+

Définition (Table des plus long préfixes communs)

≡ longueur du plus long préfixe commun entre deux suffixes consécutifs de la table des suffixes.

⇒ Amélioration des performances de recherche

Table des suffixes du mot ROUDOUDOU$

1	*ROUDOUDOU$*
2	*OUDOUDOU$*
3	*UDOUDOU$*
4	*DOUDOU$*
5	*OUDOU$*
6	*UDOU$*
7	*DOU$*
8	*OU$*
9	*U$*
10	*$*

Définition (Table des suffixes)

≡ tableau des positions des suffixes d'un mot dans l'ordre lexicographique.

+

Définition (Table des plus long préfixes communs)

≡ longueur du plus long préfixe commun entre deux suffixes consécutifs de la table des suffixes.

→ Amélioration des performances de recherche

Table des suffixes du mot ROUDOUDOU$

1	*ROUDOUDOU$*
2	*OUDOUDOU$*
3	*UDOUDOU$*
4	*DOUDOU$*
5	*OUDOU$*
6	*UDOU$*
7	*DOU$*
8	*OU$*
9	*U$*
10	*$*

Définition (Table des suffixes)

≡ tableau des positions des suffixes du mot dans l'ordre lexicographique.

+

Définition (Table des plus long préfixes communs)

≡ longueur du plus long préfixe commun entre deux suffixes consécutifs de la table des suffixes.

⇒ Amélioration des performances de recherche

Table des suffixes du mot ROUDOUDOU$

1	*ROUDOUDOU$*
2	*OUDOUDOU$*
3	*UDOUDOU$*
4	*DOUDOU$*
5	*OUDOU$*
6	*UDOU$*
7	*DOU$*
8	*OU$*
9	*U$*
10	*$*

$\xrightarrow[\mathcal{O}(n)]{\text{tri}}$

Définition (Table des suffixes)

≡ tableau des positions des suffixes du mot dans l'ordre lexicographique.

Table des suffixes du mot ROUDOUDOU$

1	*ROUDOUDOU$*
2	*OUDOUDOU$*
3	*UDOUDOU$*
4	*DOUDOU$*
5	*OUDOU$*
6	*UDOU$*
7	*DOU$*
8	*OU$*
9	*U$*
10	*$*

tri $\mathcal{O}(n)$ →

10 *$*

+

Définition (Table des plus long préfixes communs)

≡ longueur du plus long préfixe commun entre deux suffixes consécutifs de la table des suffixes.

⇒ Amélioration des performances de recherche

Définition (Table des suffixes)

≡ tableau des positions des suffixes d'un mot dans l'ordre lexicographique.

Définition (Table des plus long préfixes communs)

≡ longueur du plus long préfixe commun entre deux suffixes consécutifs de la table des suffixes.

⇒ Amélioration des performances de recherche

Table des suffixes du mot ROUDOUDOU$

1	*ROUDOUDOU$*
2	*OUDOUDOU$*
3	*UDOUDOU$*
4	*DOUDOU$*
5	*OUDOU$*
6	*UDOU$*
7	*DOU$*
8	*OU$*
9	*U$*
10	*$*

$\xrightarrow[\mathcal{O}(n)]{\text{tri}}$

10	*$*
7	*DOU$*

Définition (Table des suffixes)

≡ tableau des positions des suffixes du mot dans l'ordre lexicographique.

Table des suffixes du mot ROUDOUDOU$

1	*ROUDOUDOU$*
2	*OUDOUDOU$*
3	*UDOUDOU$*
4	*DOUDOU$*
5	*OUDOU$*
6	*UDOU$*
7	*DOU$*
8	*OU$*
9	*U$*
10	*$*

$\xrightarrow[\mathcal{O}(n)]{\text{tri}}$

10	*$*
7	*DOU$*
4	*DOUDOU$*

+

Définition (Table des plus long préfixes communs)

≡ longueur du plus long préfixe commun entre deux suffixes consécutifs de la table des suffixes.

⇒ Amélioration des performances de recherche

Définition (Table des suffixes)

≡ tableau des positions des suffixes d'un mot dans l'ordre lexicographique.

Table des suffixes du mot ROUDOUDOU$

1	*ROUDOUDOU$*
2	*OUDOUDOU$*
3	*UDOUDOU$*
4	*DOUDOU$*
5	*OUDOU$*
6	*UDOU$*
7	*DOU$*
8	*OU$*
9	*U$*
10	*$*

$\xrightarrow[\mathcal{O}(n)]{\text{tri}}$

10	*$*
7	*DOU$*
4	*DOUDOU$*
8	*OU$*

+

Définition (Table des plus long préfixes communs)

≡ longueur du plus long préfixe commun entre deux suffixes consécutifs de la table des suffixes.

⇒ Amélioration des performances de recherche

Définition (Table des suffixes)

≡ tableau des positions des suffixes du mot dans l'ordre lexicographique.

Définition (Table des plus long préfixes communs)

≡ longueur du plus long préfixe commun entre deux suffixes consécutifs de la table des suffixes.

⇒ Amélioration des performances de recherche

Table des suffixes du mot ROUDOUDOU$

1	*ROUDOUDOU$*
2	*OUDOUDOU$*
3	*UDOUDOU$*
4	*DOUDOU$*
5	*OUDOU$*
6	*UDOU$*
7	*DOU$*
8	*OU$*
9	*U$*
10	*$*

tri $\mathcal{O}(n)$ →

10	*$*
7	*DOU$*
4	*DOUDOU$*
8	*OU$*
5	*OUDOU$*

Définition (Table des suffixes)

≡ tableau des positions des suffixes du mot dans l'ordre lexicographique.

Définition (Table des plus long préfixes communs)

≡ longueur du plus long préfixe commun entre deux suffixes consécutifs de la table des suffixes.

⇒ Amélioration des performances de recherche.

Table des suffixes du mot ROUDOUDOU$

1	*ROUDOUDOU$*
2	*OUDOUDOU$*
3	*UDOUDOU$*
4	*DOUDOU$*
5	*OUDOU$*
6	*UDOU$*
7	*DOU$*
8	*OU$*
9	*U$*
10	*$*

$\xrightarrow[\mathcal{O}(n)]{\text{tri}}$

10	*$*
7	*DOU$*
4	*DOUDOU$*
8	*OU$*
5	*OUDOU$*
2	*OUDOUDOU$*

Définition (Table des suffixes)

≡ tableau des positions des suffixes d'un mot dans l'ordre lexicographique.

Table des suffixes du mot ROUDOUDOU$

1	*ROUDOUDOU$*
2	*OUDOUDOU$*
3	*UDOUDOU$*
4	*DOUDOU$*
5	*OUDOU$*
6	*UDOU$*
7	*DOU$*
8	*OU$*
9	*U$*
10	*$*

$\xrightarrow[\mathcal{O}(n)]{\text{tri}}$

10	*$*
7	*DOU$*
4	*DOUDOU$*
8	*OU$*
5	*OUDOU$*
2	*OUDOUDOU$*
1	*ROUDOUDOU$*

+

Définition (Table des plus long préfixes communs)

≡ longueur du plus long préfixe commun entre deux suffixes consécutifs de la table des suffixes.

⇒ Amélioration des performances de recherche.

Définition (Table des suffixes)

≡ tableau des positions des suffixes d'un mot dans l'ordre lexicographique.

+

Définition (Table des plus long préfixes communs)

≡ longueur du plus long préfixe commun entre deux suffixes consécutifs de la table des suffixes.

⇒ Amélioration des performances de recherche.

Table des suffixes du mot ROUDOUDOU$

1	*ROUDOUDOU$*
2	*OUDOUDOU$*
3	*UDOUDOU$*
4	*DOUDOU$*
5	*OUDOU$*
6	*UDOU$*
7	*DOU$*
8	*OU$*
9	*U$*
10	*$*

$\xrightarrow[\mathcal{O}(n)]{\text{tri}}$

10	*$*
7	*DOU$*
4	*DOUDOU$*
8	*OU$*
5	*OUDOU$*
2	*OUDOUDOU$*
1	*ROUDOUDOU$*
9	*U$*

Définition (Table des suffixes)

≡ tableau des positions des suffixes d'un mot dans l'ordre lexicographique.

+

Définition (Table des plus long préfixes communs)

≡ longueur du plus long préfixe commun entre deux suffixes consécutifs de la table des suffixes.

⇒ Amélioration des performances de recherche.

Table des suffixes du mot ROUDOUDOU$

1	*ROUDOUDOU$*
2	*OUDOUDOU$*
3	*UDOUDOU$*
4	*DOUDOU$*
5	*OUDOU$*
6	*UDOU$*
7	*DOU$*
8	*OU$*
9	*U$*
10	*$*

$\xrightarrow[\mathcal{O}(n)]{\text{tri}}$

10	*$*
7	*DOU$*
4	*DOUDOU$*
8	*OU$*
5	*OUDOU$*
2	*OUDOUDOU$*
1	*ROUDOUDOU$*
9	*U$*
6	*UDOU$*

Définition (Table des suffixes)

≡ tableau des positions des suffixes d'un mot dans l'ordre lexicographique.

Table des suffixes du mot ROUDOUDOU$

1	*ROUDOUDOU$*
2	*OUDOUDOU$*
3	*UDOUDOU$*
4	*DOUDOU$*
5	*OUDOU$*
6	*UDOU$*
7	*DOU$*
8	*OU$*
9	*U$*
10	*$*

$\xrightarrow[\mathcal{O}(n)]{\text{tri}}$

10	*$*
7	*DOU$*
4	*DOUDOU$*
8	*OU$*
5	*OUDOU$*
2	*OUDOUDOU$*
1	*ROUDOUDOU$*
9	*U$*
6	*UDOU$*
3	*UDOUDOU$*

+

Définition (Table des plus long préfixes communs)

≡ longueur du plus long préfixe commun entre deux suffixes consécutifs de la table des suffixes.

⇒ Amélioration des performances de recherche.

Définition (Table des suffixes)

≡ tableau des positions des suffixes d'un mot dans l'ordre lexicographique.

+

Définition (Table des plus long préfixes communs)

≡ longueur du plus long préfixe commun entre deux suffixes consécutifs de la table des suffixes.

⇒ Amélioration des performances de recherche.

Table des suffixes du mot ROUDOUDOU$

1	*ROUDOUDOU$*
2	*OUDOUDOU$*
3	*UDOUDOU$*
4	*DOUDOU$*
5	*OUDOU$*
6	*UDOU$*
7	*DOU$*
8	*OU$*
9	*U$*
10	*$*

$\xrightarrow[\mathcal{O}(n)]{\text{tri}}$

10	*$*
7	*DOU$*
4	*DOUDOU$*
8	*OU$*
5	*OUDOU$*
2	*OUDOUDOU$*
1	*ROUDOUDOU$*
9	*U$*
6	*UDOU$*
3	*UDOUDOU$*

Définition (Table des suffixes)

≡ tableau des positions des suffixes d'un mot dans l'ordre lexicographique.

+

Définition (Table des plus long préfixes communs)

≡ longueur du plus long préfixe commun entre deux suffixes consécutifs de la table des suffixes.

⇒ Amélioration des performances de recherche.

Table des suffixes du mot ROUDOUDOU$

LCP

10	$
7	DOU$
4	DOUDOU$
8	OU$
5	OUDOU$
2	OUDOUDOU$
1	ROUDOUDOU$
9	U$
6	UDOU$
3	UDOUDOU$

Définition (Table des suffixes)

≡ tableau des positions des suffixes d'un mot dans l'ordre lexicographique.

+

Définition (Table des plus long préfixes communs)

≡ longueur du plus long préfixe commun entre deux suffixes consécutifs de la table des suffixes.

⇒ Amélioration des performances de recherche.

Table des suffixes du mot ROUDOUDOU$

1	0	
		10 $
		7 DOU$
		4 DOUDOU$
		8 OU$
		5 OUDOU$
		2 OUDOUDOU$
		1 ROUDOUDOU$
		9 U$
		6 UDOU$
		3 UDOUDOU$

LCP

Définition (Table des suffixes)

≡ tableau des positions des suffixes d'un mot dans l'ordre lexicographique.

+

Définition (Table des plus long préfixes communs)

≡ longueur du plus long préfixe commun entre deux suffixes consécutifs de la table des suffixes.

⇒ Amélioration des performances de recherche.

Table des suffixes du mot ROUDOUDOU$

1	0
2	0

LCP

10	$
7	DOU$
4	DOUDOU$
8	OU$
5	OUDOU$
2	OUDOUDOU$
1	ROUDOUDOU$
9	U$
6	UDOU$
3	UDOUDOU$

Définition (Table des suffixes)

≡ tableau des positions des suffixes d'un mot dans l'ordre lexicographique.

+

Définition (Table des plus long préfixes communs)

≡ longueur du plus long préfixe commun entre deux suffixes consécutifs de la table des suffixes.

⇒ Amélioration des performances de recherche.

Table des suffixes du mot ROUDOUDOU$

1	0
2	0
3	3

LCP

10	$
7	DOU$
4	DOUDOU$
8	OU$
5	OUDOU$
2	OUDOUDOU$
1	ROUDOUDOU$
9	U$
6	UDOU$
3	UDOUDOU$

Définition (Table des suffixes)

≡ tableau des positions des suffixes d'un mot dans l'ordre lexicographique.

+

Définition (Table des plus long préfixes communs)

≡ longueur du plus long préfixe commun entre deux suffixes consécutifs de la table des suffixes.

⇒ Amélioration des performances de recherche.

Table des suffixes du mot ROUDOUDOU$

1	0
2	0
3	3
4	0

LCP

10	$
7	DOU$
4	DOUDOU$
8	OU$
5	OUDOU$
2	OUDOUDOU$
1	ROUDOUDOU$
9	U$
6	UDOU$
3	UDOUDOU$

Définition (Table des suffixes)

≡ tableau des positions des suffixes du mot dans l'ordre lexicographique.

Définition (Table des plus long préfixes communs)

≡ longueur du plus long préfixe commun entre deux suffixes consécutifs de la table des suffixes.

⇒ Amélioration des performances de recherche.

Table des suffixes du mot ROUDOUDOU$

1	0
2	0
3	3
4	0
5	2

LCP

10	$
7	DOU$
4	DOUDOU$
8	OU$
5	OUDOU$
2	OUDOUDOU$
1	ROUDOUDOU$
9	U$
6	UDOU$
3	UDOUDOU$

Définition (Table des suffixes)

≡ tableau des positions des suffixes du mot dans l'ordre lexicographique.

+

Définition (Table des plus long préfixes communs)

≡ longueur du plus long préfixe commun entre deux suffixes consécutifs de la table des suffixes.

⇒ Amélioration des performances de recherche.

Table des suffixes du mot ROUDOUDOU$

1	0
2	0
3	3
4	0
5	2
6	5

LCP

10	$
7	DOU$
4	DOUDOU$
8	OU$
5	OUDOU$
2	OUDOUDOU$
1	ROUDOUDOU$
9	U$
6	UDOU$
3	UDOUDOU$

Définition (Table des suffixes)

≡ tableau des positions des suffixes d'un mot dans l'ordre lexicographique.

Définition (Table des plus long préfixes communs)

≡ longueur du plus long préfixe commun entre deux suffixes consécutifs de la table des suffixes.

⇒ Amélioration des performances de recherche.

Table des suffixes du mot ROUDOUDOU$

1	0
2	0
3	3
4	0
5	2
6	5
7	0

LCP

10	$
7	DOU$
4	DOUDOU$
8	OU$
5	OUDOU$
2	OUDOUDOU$
1	ROUDOUDOU$
9	U$
6	UDOU$
3	UDOUDOU$

Définition (Table des suffixes)

≡ tableau des positions des suffixes d'un mot dans l'ordre lexicographique.

+

Définition (Table des plus long préfixes communs)

≡ longueur du plus long préfixe commun entre deux suffixes consécutifs de la table des suffixes.

⇒ Amélioration des performances de recherche.

Table des suffixes du mot ROUDOUDOU$

1	0
2	0
3	3
4	0
5	2
6	5
7	0
8	0

LCP

10	$
7	DOU$
4	DOUDOU$
8	OU$
5	OUDOU$
2	OUDOUDOU$
1	ROUDOUDOU$
9	U$
6	UDOU$
3	UDOUDOU$

Définition (Table des suffixes)

≡ tableau des positions des suffixes d'un mot dans l'ordre lexicographique.

+

Définition (Table des plus long préfixes communs)

≡ longueur du plus long préfixe commun entre deux suffixes consécutifs de la table des suffixes.

⇒ Amélioration des performances de recherche.

Table des suffixes du mot ROUDOUDOU$

1	0
2	0
3	3
4	0
5	2
6	5
7	0
8	0
9	1

LCP

10	$
7	DOU$
4	DOUDOU$
8	OU$
5	OUDOU$
2	OUDOUDOU$
1	ROUDOUDOU$
9	U$
6	UDOU$
3	UDOUDOU$

Définition (Table des suffixes)

≡ tableau des positions des suffixes d'un mot dans l'ordre lexicographique.

+

Définition (Table des plus long préfixes communs)

≡ longueur du plus long préfixe commun entre deux suffixes consécutifs de la table des suffixes.

⇒ Amélioration des performances de recherche.

Table des suffixes du mot ROUDOUDOU$

1	0
2	0
3	3
4	0
5	2
6	5
7	0
8	0
9	1
10	4

LCP

10	$
7	DOU$
4	DOUDOU$
8	OU$
5	OUDOU$
2	OUDOUDOU$
1	ROUDOUDOU$
9	U$
6	UDOU$
3	UDOUDOU$

Définition (Table des suffixes)

≡ tableau des positions des suffixes d'un mot dans l'ordre lexicographique.

+

Définition (Table des plus long préfixes communs)

≡ longueur du plus long préfixe commun entre deux suffixes consécutifs de la table des suffixes.

⇒ Amélioration des performances de recherche.

Table des suffixes du mot ROUDOUDOU$

1	0
2	0
3	3
4	0
5	2
6	5
7	0
8	0
9	1
10	4

LCP

10	$
7	DOU$
4	DOUDOU$
8	OU$
5	OUDOU$
2	OUDOUDOU$
1	ROUDOUDOU$
9	U$
6	UDOU$
3	UDOUDOU$

Extraction de motif
Structures d'index [Tables]

Alignement visuel de séquences
Recherche de motif
Extraction de motif
Contexte
Quelques programmes [presque] naïfs
Algorithme KMR
Structures d'index
Hachage
Arbres
Automates
Tables

Définition (Table des suffixes)

≡ tableau des positions des suffixes d'un mot dans l'ordre lexicographique.

+

Définition (Table des plus long préfixes communs)

≡ longueur du plus long préfixe commun entre deux suffixes consécutifs de la table des suffixes.

⇒ Amélioration des performances de recherche

Définition (FM-index)

Le FM-index (Ferragina–Manzini index) est la structure résultant de la compression de la transformée de BURROWS-WHEELER. Sa construction est basée sur la table des suffixes pour le calcul de la transformée.

Définition (Table des suffixes)

≡ tableau des positions des suffixes du mot triés par ordre lexicographique.

Transformée de BURROWS-WHEELER du mot ROUDOUDOU$

R O U D O U D O U $

+

Définition (Table des plus long préfixes communs)

≡ longueur du plus long préfixe commun entre deux suffixes consécutifs de la table des suffixes.

⇒ Amélioration des performances de recherche

Définition (FM-index)

Le FM-index (Ferragina–Manzini index) est la structure résultant de la compression de la transformée de BURROWS-WHEELER. Sa construction est basée sur la table des suffixes pour le calcul de la transformée.

Définition (Table des suffixes)

≡ tableau des suffixes du mot ordre lexicographique.

+

Définition (Table des plus long préfixes communs)

≡ longueur du plus long préfixe commun entre deux suffixes consécutifs de la table des suffixes.

⇒ Amélioration des performances de recherche

Définition (FM-index)

Le FM-index (Ferragina–Manzini index) est la structure résultant de la compression de la transformée de BURROWS-WHEELER. Sa construction est basée sur la table des suffixes pour le calcul de la transformée.

Transformée de BURROWS-WHEELER du mot ROUDOUDOU$

R O U D O U D O U $
$ R O U D O U D O U

Définition (Table des suffixes)

≡ tableau des indices des suffixes du mot trié par ordre lexicographique.

+

Définition (Tableau des plus long préfixes communs)

≡ longueur du plus long préfixe commun entre deux suffixes consécutifs de la table des suffixes.

⇒ Amélioration des performances de recherche

Définition (FM-index)

Le FM-index (Ferragina–Manzini index) est la structure résultant de la compression de la transformée de BURROWS-WHEELER. Sa construction est basée sur la table des suffixes pour le calcul de la transformée.

Transformée de BURROWS-WHEELER du mot ROUDOUDOU$

R	O	U	D	O	U	D	O	U	$
$	R	O	U	D	O	U	D	O	U
U	$	R	O	U	D	O	U	D	O

Définition (Table des suffixes)

≡ tableau des indices des suffixes du mot dans l'ordre lexicographique.

+

Définition (Table des plus long préfixes communs)

≡ longueur du plus long préfixe commun entre deux suffixes consécutifs de la table des suffixes.

⇒ Amélioration des performances de recherche

Définition (FM-index)

Le FM-index (Ferragina–Manzini index) est la structure résultant de la compression de la transformée de BURROWS-WHEELER. Sa construction est basée sur la table des suffixes pour le calcul de la transformée.

Transformée de BURROWS-WHEELER du mot ROUDOUDOU$

R	O	U	D	O	U	D	O	U	$
$	R	O	U	D	O	U	D	O	U
U	$	R	O	U	D	O	U	D	O
O	U	$	R	O	U	D	O	U	D

Définition (Table des suffixes)

≡ tableau des indices des suffixes du mot triés par ordre lexicographique.

+

Définition (Tableau des plus longs préfixes communs)

≡ longueur du plus long préfixe commun entre deux suffixes consécutifs de la table des suffixes.

⇒ Amélioration des performances de recherche

Définition (FM-index)

Le FM-index (Ferragina–Manzini index) est la structure résultant de la compression de la transformée de BURROWS-WHEELER. Sa construction est basée sur la table des suffixes pour le calcul de la transformée.

Transformée de BURROWS-WHEELER du mot ROUDOUDOU$

R	O	U	D	O	U	D	O	U	$
$	R	O	U	D	O	U	D	O	U
U	$	R	O	U	D	O	U	D	O
O	U	$	R	O	U	D	O	U	D
D	O	U	$	R	O	U	D	O	U

Définition (Table des suffixes)

≡ tableau des indices des suffixes du mot trié par ordre lexicographique.

+

Définition (Tableau des plus longs préfixes communs)

≡ longueur du plus long préfixe commun entre deux suffixes consécutifs de la table des suffixes.

⇒ Amélioration des performances de recherche

Définition (FM-index)

Le FM-index (Ferragina–Manzini index) est la structure résultant de la compression de la transformée de BURROWS-WHEELER.

Sa construction est basée sur la table des suffixes pour le calcul de la transformée.

Transformée de BURROWS-WHEELER du mot ROUDOUDOU$

R	O	U	D	O	U	D	O	U	$
$	R	O	U	D	O	U	D	O	U
U	$	R	O	U	D	O	U	D	O
O	U	$	R	O	U	D	O	U	D
D	O	U	$	R	O	U	D	O	U
U	D	O	U	$	R	O	U	D	O

Définition (Table des suffixes)

≡ tableau des indices des suffixes du mot trié par ordre lexicographique.

Définition (Tableau des plus long préfixes communs)

≡ longueur du plus long préfixe commun entre deux suffixes consécutifs de la table des suffixes.

⇒ Amélioration des performances de recherche

Définition (FM-index)

Le FM-index (Ferragina–Manzini index) est la structure résultant de la compression de la transformée de BURROWS-WHEELER.

Sa construction est basée sur la table des suffixes pour le calcul de la transformée.

Transformée de BURROWS-WHEELER du mot ROUDOUDOU$

R	O	U	D	O	U	D	O	U	$
$	R	O	U	D	O	U	D	O	U
U	$	R	O	U	D	O	U	D	O
O	U	$	R	O	U	D	O	U	D
D	O	U	$	R	O	U	D	O	U
U	D	O	U	$	R	O	U	D	O
O	U	D	O	U	$	R	O	U	D

Définition (Table des suffixes)

≡ tableau des indices des suffixes du mot trié par ordre lexicographique.

+

Définition (Tableau des plus long préfixes communs)

≡ longueur du plus long préfixe commun entre deux suffixes consécutifs de la table des suffixes.

⇒ Amélioration des performances de recherche

Définition (FM-index)

Le FM-index (Ferragina–Manzini index) est la structure résultant de la compression de la transformée de BURROWS-WHEELER.

Sa construction est basée sur la table des suffixes pour le calcul de la transformée.

Transformée de BURROWS-WHEELER du mot ROUDOUDOU$

R	O	U	D	O	U	D	O	U	$
$	R	O	U	D	O	U	D	O	U
U	$	R	O	U	D	O	U	D	O
O	U	$	R	O	U	D	O	U	D
D	O	U	$	R	O	U	D	O	U
U	D	O	U	$	R	O	U	D	O
O	U	D	O	U	$	R	O	U	D
D	O	U	D	O	U	$	R	O	U

Définition (Table des suffixes)

≡ tableau des indices des suffixes du mot trié par ordre lexicographique.

+

Définition (Table des plus long préfixes communs)

≡ longueur du plus long préfixe commun entre deux suffixes consécutifs de la table des suffixes.

⇒ Amélioration des performances de recherche

Définition (FM-index)

Le FM-index (Ferragina–Manzini index) est la structure résultant de la compression de la transformée de BURROWS-WHEELER.

Sa construction est basée sur la table des suffixes pour le calcul de la transformée.

Transformée de BURROWS-WHEELER du mot ROUDOUDOU$

R	O	U	D	O	U	D	O	U	$
$	R	O	U	D	O	U	D	O	U
U	$	R	O	U	D	O	U	D	O
O	U	$	R	O	U	D	O	U	D
D	O	U	$	R	O	U	D	O	U
U	D	O	U	$	R	O	U	D	O
O	U	D	O	U	$	R	O	U	D
D	O	U	D	O	U	$	R	O	U
U	D	O	U	D	O	U	$	R	O

Définition (Table des suffixes)

≡ tableau des indices des suffixes du mot trié par ordre lexicographique.

Définition (Tableau des plus long préfixes communs)

≡ longueur du plus long préfixe commun entre deux suffixes consécutifs de la table des suffixes.

⇒ Amélioration des performances de recherche

Définition (FM-index)

Le FM-index (Ferragina–Manzini index) est la structure résultant de la compression de la transformée de BURROWS-WHEELER. Sa construction est basée sur la table des suffixes pour le calcul de la transformée.

Transformée de BURROWS-WHEELER du mot ROUDOUDOU$

R	O	U	D	O	U	D	O	U	$
$	R	O	U	D	O	U	D	O	U
U	$	R	O	U	D	O	U	D	O
O	U	$	R	O	U	D	O	U	D
D	O	U	$	R	O	U	D	O	U
U	D	O	U	$	R	O	U	D	O
O	U	D	O	U	$	R	O	U	D
D	O	U	D	O	U	$	R	O	U
U	D	O	U	D	O	U	$	R	O
O	U	D	O	U	D	O	U	$	R

Définition (Table des suffixes)

≡ tableau des indices des suffixes du mot trié par ordre lexicographique.

+

Définition (Tableau des plus long préfixes communs)

≡ longueur du plus long préfixe commun entre deux suffixes consécutifs de la table des suffixes.

⇒ Amélioration des performances de recherche

Définition (Transformée de Burrows-Wheeler)

Le FM-index (Ferragina–Manzini index) est la structure résultant de la compression de la transformée de BURROWS-WHEELER. Sa construction est basée sur la table des suffixes pour le calcul de la transformée.

Transformée de BURROWS-WHEELER du mot ROUDOUDOU$

R	O	U	D	O	U	D	O	U	$
$	R	O	U	D	O	U	D	O	U
U	$	R	O	U	D	O	U	D	O
O	U	$	R	O	U	D	O	U	D
D	O	U	$	R	O	U	D	O	U
U	D	O	U	$	R	O	U	D	O
O	U	D	O	U	$	R	O	U	D
D	O	U	D	O	U	$	R	O	U
U	D	O	U	D	O	U	$	R	O
O	U	D	O	U	D	O	U	$	R

$\xrightarrow[\mathcal{O}(n)]{\text{tri}}$

Définition (Table des suffixes)

≡ tableau des indices des suffixes du mot ordre lexicographique.

Définition (Table des plus long préfixes communs)

≡ longueur du plus long préfixe commun entre deux suffixes consécutifs de la table des suffixes.

⇒ Amélioration des performances de recherche

Définition (Table de BWT)

Le FM-index (Ferragina–Manzini index) est la structure résultant de la compression de la transformée de BURROWS-WHEELER. Sa construction est basée sur la table des suffixes pour le calcul de la transformée.

Transformée de BURROWS-WHEELER du mot ROUDOUDOU$

R	O	U	D	O	U	D	O	U	$
$	R	O	U	D	O	U	D	O	U
U	$	R	O	U	D	O	U	D	O
O	U	$	R	O	U	D	O	U	D
D	O	U	$	R	O	U	D	O	U
U	D	O	U	$	R	O	U	D	O
O	U	D	O	U	$	R	O	U	D
D	O	U	D	O	U	$	R	O	U
U	D	O	U	D	O	U	$	R	O
O	U	D	O	U	D	O	U	$	R

$\xrightarrow[\mathcal{O}(n)]{\text{tri}}$

$ R O U D O U D O U

Définition (Table des suffixes)

≡ tableau des indices des suffixes du mot par ordre lexicographique.

Définition (Table des plus long préfixes communs)

≡ longueur du plus long préfixe commun entre deux suffixes consécutifs de la table des suffixes.

⇒ Amélioration des performances de recherche

Définition (FM-index)

Le FM-index (Ferragina–Manzini index) est la structure résultant de la compression de la transformée de BURROWS-WHEELER. Sa construction est basée sur la table des suffixes pour le calcul de la transformée.

Transformée de BURROWS-WHEELER du mot ROUDOUDOU$

R	O	U	D	O	U	D	O	U	$
$	R	O	U	D	O	U	D	O	U
U	$	R	O	U	D	O	U	D	O
O	U	$	R	O	U	D	O	U	D
D	O	U	$	R	O	U	D	O	U
U	D	O	U	$	R	O	U	D	O
O	U	D	O	U	$	R	O	U	D
D	O	U	D	O	U	$	R	O	U
U	D	O	U	D	O	U	$	R	O
O	U	D	O	U	D	O	U	$	R

$\xrightarrow[\mathcal{O}(n)]{\text{tri}}$

$	R	O	U	D	O	U	D	O	U
D	O	U	$	R	O	U	D	O	U

Définition (Table des suffixes)

≡ tableau des indices des suffixes du mot ordre lexicographique.

Définition (Tableau des plus long préfixes communs)

≡ longueur du plus long préfixe commun entre deux suffixes consécutifs de la table des suffixes.

⇒ Amélioration des performances de recherche

Définition (Transformée de Burrows-Wheeler)

Le FM-index (Ferragina–Manzini index) est la structure résultant de la compression de la transformée de BURROWS-WHEELER. Sa construction est basée sur la table des suffixes pour le calcul de la transformée.

Transformée de BURROWS-WHEELER du mot ROUDOUDOU$

R	O	U	D	O	U	D	O	U	$
$	R	O	U	D	O	U	D	O	U
U	$	R	O	U	D	O	U	D	O
O	U	$	R	O	U	D	O	U	D
D	O	U	$	R	O	U	D	O	U
U	D	O	U	$	R	O	U	D	O
O	U	D	O	U	$	R	O	U	D
D	O	U	D	O	U	$	R	O	U
U	D	O	U	D	O	U	$	R	O
O	U	D	O	U	D	O	U	$	R

$\xrightarrow[\mathcal{O}(n)]{\text{tri}}$

$	R	O	U	D	O	U	D	O	U
D	O	U	$	R	O	U	D	O	U
D	O	U	D	O	U	$	R	O	U

Définition (Table des suffixes)

≡ tableau des indices des suffixes du mot ordre lexicographique.

Définition (Tableau des plus longs préfixes communs)

≡ longueur du plus long préfixe commun entre deux suffixes consécutifs de la table des suffixes.

⇒ Amélioration des performances de recherche

Définition (Table de Burrows-Wheeler)

Le FM-index (Ferragina–Manzini index) est la structure résultant de la compression de la transformée de BURROWS-WHEELER. Sa construction est basée sur la table des suffixes pour le calcul de la transformée.

Transformée de BURROWS-WHEELER du mot ROUDOUDOU$

R	O	U	D	O	U	D	O	U	$
$	R	O	U	D	O	U	D	O	U
U	$	R	O	U	D	O	U	D	O
O	U	$	R	O	U	D	O	U	D
D	O	U	$	R	O	U	D	O	U
U	D	O	U	$	R	O	U	D	O
O	U	D	O	U	$	R	O	U	D
D	O	U	D	O	U	$	R	O	U
U	D	O	U	D	O	U	$	R	O
O	U	D	O	U	D	O	U	$	R

$\xrightarrow[\mathcal{O}(n)]{\text{tri}}$

$	R	O	U	D	O	U	D	O	U
D	O	U	$	R	O	U	D	O	U
D	O	U	D	O	U	$	R	O	U
O	U	$	R	O	U	D	O	U	D

Définition (Table des suffixes)

≡ tableau des indices des suffixes du mot ordre lexicographique.

Définition (Tableau des plus longs préfixes communs)

≡ longueur du plus long préfixe commun entre deux suffixes consécutifs de la table des suffixes.

⇒ Amélioration des performances de recherche

Définition (Transformée de Burrows-Wheeler)

Le FM-index (Ferragina–Manzini index) est la structure résultant de la compression de la transformée de BURROWS-WHEELER. Sa construction est basée sur la table des suffixes pour le calcul de la transformée.

Transformée de BURROWS-WHEELER du mot ROUDOUDOU$

R	O	U	D	O	U	D	O	U	$
$	R	O	U	D	O	U	D	O	U
U	$	R	O	U	D	O	U	D	O
O	U	$	R	O	U	D	O	U	D
D	O	U	$	R	O	U	D	O	U
U	D	O	U	$	R	O	U	D	O
O	U	D	O	U	$	R	O	U	D
D	O	U	D	O	U	$	R	O	U
U	D	O	U	D	O	U	$	R	O
O	U	D	O	U	D	O	U	$	R

$\xrightarrow[\mathcal{O}(n)]{\text{tri}}$

$	R	O	U	D	O	U	D	O	U
D	O	U	$	R	O	U	D	O	U
D	O	U	D	O	U	$	R	O	U
O	U	$	R	O	U	D	O	U	D
O	U	D	O	U	$	R	O	U	D

Définition (Table des suffixes)

≡ tableau des indices des suffixes du mot ordre lexicographique.

Définition (Tableau des plus longs préfixes communs)

≡ longueur du plus long préfixe commun entre deux suffixes consécutifs de la table des suffixes.

⇒ Amélioration des performances de recherche

Définition (Transformée de Burrows-Wheeler)

Le FM-index (Ferragina–Manzini index) est la structure résultant de la compression de la transformée de BURROWS-WHEELER. Sa construction est basée sur la table des suffixes pour le calcul de la transformée.

Transformée de BURROWS-WHEELER du mot ROUDOUDOU$

R	O	U	D	O	U	D	O	U	$
$	R	O	U	D	O	U	D	O	U
U	$	R	O	U	D	O	U	D	O
O	U	$	R	O	U	D	O	U	D
D	O	U	$	R	O	U	D	O	U
U	D	O	U	$	R	O	U	D	O
O	U	D	O	U	$	R	O	U	D
D	O	U	D	O	U	$	R	O	U
U	D	O	U	D	O	U	$	R	O
O	U	D	O	U	D	O	U	$	R

$\xrightarrow[\mathcal{O}(n)]{\text{tri}}$

$	R	O	U	D	O	U	D	O	U
D	O	U	$	R	O	U	D	O	U
D	O	U	D	O	U	$	R	O	U
O	U	$	R	O	U	D	O	U	D
O	U	D	O	U	$	R	O	U	D
O	U	D	O	U	D	O	U	$	R

Définition (Table des suffixes)

≡ tableau des indices des suffixes du mot dans l'ordre lexicographique.

Définition (Table des plus long préfixes communs)

≡ longueur du plus long préfixe commun entre deux suffixes consécutifs de la table des suffixes.

⇒ Amélioration des performances de recherche.

Définition (Table de la transformée)

Le FM-index (Ferragina–Manzini index) est la structure résultant de la compression de la transformée de BURROWS-WHEELER. Sa construction est basée sur la table des suffixes pour le calcul de la transformée.

Transformée de BURROWS-WHEELER du mot ROUDOUDOU$

R	O	U	D	O	U	D	O	U	$
$	R	O	U	D	O	U	D	O	U
U	$	R	O	U	D	O	U	D	O
O	U	$	R	O	U	D	O	U	D
D	O	U	$	R	O	U	D	O	U
U	D	O	U	$	R	O	U	D	O
O	U	D	O	U	$	R	O	U	D
D	O	U	D	O	U	$	R	O	U
U	D	O	U	D	O	U	$	R	O
O	U	D	O	U	D	O	U	$	R

$\xrightarrow[\mathcal{O}(n)]{\text{tri}}$

$	R	O	U	D	O	U	D	O	U
D	O	U	$	R	O	U	D	O	U
D	O	U	D	O	U	$	R	O	U
O	U	$	R	O	U	D	O	U	D
O	U	D	O	U	$	R	O	U	D
O	U	D	O	U	D	O	U	$	R
R	O	U	D	O	U	D	O	U	$

Définition (Table des suffixes)

≡ tableau des indices des suffixes du mot ordre lexicographique.

Définition (Table des plus long préfixes communs)

≡ longueur du plus long préfixe commun entre deux suffixes consécutifs de la table des suffixes.

⇒ Amélioration des performances de recherche

Définition (Table de BURROWS-WHEELER)

Transformée de BURROWS-WHEELER du mot ROUDOUDOU$

R	O	U	D	O	U	D	O	U	$
$	R	O	U	D	O	U	D	O	U
U	$	R	O	U	D	O	U	D	O
O	U	$	R	O	U	D	O	U	D
D	O	U	$	R	O	U	D	O	U
U	D	O	U	$	R	O	U	D	O
O	U	D	O	U	$	R	O	U	D
D	O	U	D	O	U	$	R	O	U
U	D	O	U	D	O	U	$	R	O
O	U	D	O	U	D	O	U	$	R

$\xrightarrow[\mathcal{O}(n)]{\text{tri}}$

$	R	O	U	D	O	U	D	O	U
D	O	U	$	R	O	U	D	O	U
D	O	U	D	O	U	$	R	O	U
O	U	$	R	O	U	D	O	U	D
O	U	D	O	U	$	R	O	U	D
O	U	D	O	U	D	O	U	$	R
R	O	U	D	O	U	D	O	U	$
U	$	R	O	U	D	O	U	D	O

Le FM-index (Ferragina–Manzini index) est la structure résultant de la compression de la transformée de BURROWS-WHEELER. Sa construction est basée sur la table des suffixes pour le calcul de la transformée.

Définition (Table des suffixes)

≡ tableau des indices des suffixes du mot dans l'ordre lexicographique.

Définition (Table des plus long préfixes communs)

≡ longueur du plus long préfixe commun entre deux suffixes consécutifs de la table des suffixes.

⇒ Amélioration des performances de recherche.

Définition (Transformée de Burrows-Wheeler)

Le FM-index (Ferragina–Manzini index) est la structure résultant de la compression de la transformée de BURROWS-WHEELER. Sa construction est basée sur la table des suffixes pour le calcul de la transformée.

Transformée de BURROWS-WHEELER du mot ROUDOUDOU$

R	O	U	D	O	U	D	O	U	$
$	R	O	U	D	O	U	D	O	U
U	$	R	O	U	D	O	U	D	O
O	U	$	R	O	U	D	O	U	D
D	O	U	$	R	O	U	D	O	U
U	D	O	U	$	R	O	U	D	O
O	U	D	O	U	$	R	O	U	D
D	O	U	D	O	U	$	R	O	U
U	D	O	U	D	O	U	$	R	O
O	U	D	O	U	D	O	U	$	R

$\xrightarrow[\mathcal{O}(n)]{\text{tri}}$

$	R	O	U	D	O	U	D	O	U
D	O	U	$	R	O	U	D	O	U
D	O	U	D	O	U	$	R	O	U
O	U	$	R	O	U	D	O	U	D
O	U	D	O	U	$	R	O	U	D
O	U	D	O	U	D	O	U	$	R
R	O	U	D	O	U	D	O	U	$
U	$	R	O	U	D	O	U	D	O
U	D	O	U	$	R	O	U	D	O

Définition (Table des suffixes)

≡ tableau des indices des suffixes du mot dans l'ordre lexicographique.

Définition (Table des plus long préfixes communs)

≡ longueur du plus long préfixe commun entre deux suffixes consécutifs de la table des suffixes.

⇒ Amélioration des performances de recherche.

Définition (Transformée de Burrows-Wheeler)

Le FM-index (Ferragina–Manzini index) est la structure résultant de la compression de la transformée de BURROWS-WHEELER. Sa construction est basée sur la table des suffixes pour le calcul de la transformée.

Transformée de BURROWS-WHEELER du mot ROUDOUDOU$

R	O	U	D	O	U	D	O	U	$
$	R	O	U	D	O	U	D	O	U
U	$	R	O	U	D	O	U	D	O
O	U	$	R	O	U	D	O	U	D
D	O	U	$	R	O	U	D	O	U
U	D	O	U	$	R	O	U	D	O
O	U	D	O	U	$	R	O	U	D
D	O	U	D	O	U	$	R	O	U
U	D	O	U	D	O	U	$	R	O
O	U	D	O	U	D	O	U	$	R

$\xrightarrow[\mathcal{O}(n)]{\text{tri}}$

$	R	O	U	D	O	U	D	O	U
D	O	U	$	R	O	U	D	O	U
D	O	U	D	O	U	$	R	O	U
O	U	$	R	O	U	D	O	U	D
O	U	D	O	U	$	R	O	U	D
O	U	D	O	U	D	O	U	$	R
R	O	U	D	O	U	D	O	U	$
U	$	R	O	U	D	O	U	D	O
U	D	O	U	$	R	O	U	D	O
U	D	O	U	D	O	U	$	R	O

Définition (Table des suffixes)

≡ tableau des indices des suffixes du mot dans l'ordre lexicographique.

+

Définition (Tableau des plus longs préfixes communs)

≡ longueur du plus long préfixe commun entre deux suffixes consécutifs de la table des suffixes.

⇒ Amélioration des performances de recherche

Définition (Table de Burrows-Wheeler)

Le FM-index (Ferragina–Manzini index) est la structure résultant de la compression de la transformée de BURROWS-WHEELER. Sa construction est basée sur la table des suffixes pour le calcul de la transformée.

Transformée de BURROWS-WHEELER du mot ROUDOUDOU$

R	O	U	D	O	U	D	O	U	$
$	R	O	U	D	O	U	D	O	U
U	$	R	O	U	D	O	U	D	O
O	U	$	R	O	U	D	O	U	D
D	O	U	$	R	O	U	D	O	U
U	D	O	U	$	R	O	U	D	O
O	U	D	O	U	$	R	O	U	D
D	O	U	D	O	U	$	R	O	U
U	D	O	U	D	O	U	$	R	O
O	U	D	O	U	D	O	U	$	R

tri $\rightsquigarrow$ $\mathcal{O}(n)$

$	R	O	U	D	O	U	D	O	U	B
D	O	U	$	R	O	U	D	O	U	W
D	O	U	D	O	U	$	R	O	U	T
O	U	$	R	O	U	D	O	U	D	
O	U	D	O	U	$	R	O	U	D	
O	U	D	O	U	D	O	U	$	R	
R	O	U	D	O	U	D	O	U	$	← 7
U	$	R	O	U	D	O	U	D	O	
U	D	O	U	$	R	O	U	D	O	
U	D	O	U	D	O	U	$	R	O	

$\simeq SA$

Définition (Table des suffixes)

≡ tableau des suffixes du mot triés par ordre lexicographique.

Transformée de BURROWS-WHEELER du mot ROUDOUDOU$

B
W ≃ SA
T
U
U
U
D
D
R
$ ← 7
O
O
O

+

Définition (Table des plus long préfixes communs)

≡ longueur du plus long préfixe commun entre deux suffixes consécutifs de la table des suffixes.

⇒ Amélioration des performances de recherche

Définition (FM-index)

Le FM-index (Ferragina–Manzini index) est la structure résultant de la compression de la transformée de BURROWS-WHEELER. Sa construction est basée sur la table des suffixes pour le calcul de la transformée.

Définition (Table des suffixes)

≡ tableau des suffixes du mot ordre lexicographique.

+

Définition (Table des plus long préfixes communs)

≡ longueur du plus long préfixe commun entre deux suffixes consecutifs de la table des suffixes.

⇒ Amélioration des performances de recherche

Définition (FM-index)

Le FM-index (Ferragina–Manzini index) est la structure résultant de la compression de la transformée de BURROWS-WHEELER. Sa construction est basée sur la table des suffixes pour le calcul de la transformée.

Transformée de BURROWS-WHEELER du mot ROUDOUDOU$

Texte d'origine	R_0 O_1 U_2 D_3 O_4 U_5 D_6 O_7 U_8 $\$_9$
BWT	U_8 U_5 U_2 D_6 D_3 R_0 $\$_9$ O_7 O_4 O_1
SA	$\$_9$ D_6 D_3 O_7 O_4 O_1 R_0 U_8 U_5 U_2

Définition (Table des suffixes)

≡ tableau des suffixes du mot ordre lexicographique.

+

Définition (Table des plus long préfixes communs)

≡ longueur du plus long préfixe commun entre deux suffixes consécutifs de la table des suffixes.

⇒ Amélioration des performances de recherche

Définition (FM-index)

Le FM-index (Ferragina–Manzini index) est la structure résultant de la compression de la transformée de BURROWS-WHEELER. Sa construction est basée sur la table des suffixes pour le calcul de la transformée.

Transformée de BURROWS-WHEELER du mot ROUDOUDOU$

Texte d'origine	R_0 O_1 U_2 D_3 O_4 U_5 D_6 O_7 U_8 $\$_9$
BWT	U_8 U_5 U_2 D_6 D_3 R_0 $\$_9$ O_7 O_4 O_1
SA	$\$_9$ D_6 D_3 O_7 O_4 O_1 R_0 U_8 U_5 U_2

Recherche du mot UDOU

Définition (Table des suffixes)

≡ tableau des suffixes du mot triés par ordre lexicographique.

Définition (Table des plus long préfixes communs)

≡ longueur du plus long préfixe commun entre deux suffixes consécutifs de la table des suffixes.

⇒ Amélioration des performances de la recherche

Définition (FM-index)

Le FM-index (Ferragina–Manzini index) est la structure résultant de la compression de la transformée de BURROWS-WHEELER.

Sa construction est basée sur la table des suffixes pour le calcul de la transformée.

Transformée de BURROWS-WHEELER du mot ROUDOUDOU$

Texte d'origine	R_0 O_1 U_2 D_3 O_4 U_5 D_6 O_7 U_8 $\$_9$
BWT	U_8 U_5 U_2 D_6 D_3 R_0 $\$_9$ O_7 O_4 O_1
SA	$\$_9$ D_6 D_3 O_7 O_4 O_1 R_0 U_8 U_5 U_2

Recherche du mot UDOU ←

Définition (Table des suffixes)

≡ tableau des suffixes du mot triés par ordre lexicographique.

Définition (Table des plus long préfixes communs)

≡ longueur du plus long préfixe commun entre deux suffixes consécutifs de la table des suffixes.

⇒ Amélioration des performances de la recherche

Définition (FM-index)

Le FM-index (Ferragina–Manzini index) est la structure résultant de la compression de la transformée de BURROWS-WHEELER. Sa construction est basée sur la table des suffixes pour le calcul de la transformée.

Transformée de BURROWS-WHEELER du mot ROUDOUDOU$

Texte d'origine	R_0	O_1	U_2	D_3	O_4	U_5	D_6	O_7	U_8	$\$_9$
BWT	U_8	U_5	U_2	D_6	D_3	R_0	$\$_9$	O_7	O_4	O_1
SA	$\$_9$	D_6	D_3	O_7	O_4	O_1	R_0	U_8	U_5	U_2

Recherche du mot UDOU ←

Définition (Table des suffixes)

≡ tableau des suffixes du mot ordre lexicographique.

Définition (Table des plus long préfixes communs)

≡ longueur du plus long préfixe commun entre deux suffixes consécutifs de la table des suffixes.

⇒ Amélioration des performances de la recherche

Définition (FM-index)

Le FM-index (Ferragina–Manzini index) est la structure résultant de la compression de la transformée de BURROWS-WHEELER.

Sa construction est basée sur la table des suffixes pour le calcul de la transformée.

Transformée de BURROWS-WHEELER du mot ROUDOUDOU$

Texte d'origine	R_0 O_1 U_2 D_3 O_4 U_5 D_6 O_7 U_8 $\$_9$
BWT	U_8 U_5 U_2 D_6 D_3 R_0 $\$_9$ O_7 O_4 O_1
SA	$\$_9$ D_6 D_3 O_7 O_4 O_1 R_0 U_8 U_5 U_2

U_8
U_5
U_2

Recherche du mot UDOU ←

Définition (Table des suffixes)

≡ tableau des suffixes du mot trié par ordre lexicographique.

+

Définition (Table des plus long préfixes communs)

≡ longueur du plus long préfixe commun entre deux suffixes consécutifs de la table des suffixes.

⇒ Amélioration des performances de recherche

Définition (FM-index)

Le FM-index (Ferragina–Manzini index) est la structure résultant de la compression de la transformée de BURROWS-WHEELER. Sa construction est basée sur la table des suffixes pour le calcul de la transformée.

Transformée de BURROWS-WHEELER du mot ROUDOUDOU$

Texte d'origine	R_0 O_1 U_2 D_3 O_4 U_5 D_6 O_7 U_8 $\$_9$
BWT	U_8 U_5 U_2 D_6 D_3 R_0 $\$_9$ O_7 O_4 O_1
SA	$\$_9$ D_6 D_3 O_7 O_4 O_1 R_0 U_8 U_5 U_2

U_8
U_5
U_2

Recherche du mot UDOU ←

Définition (Table des suffixes)

≡ tableau des suffixes du mot ordre lexicographique.

Définition (Table des plus long préfixes communs)

≡ longueur du plus long préfixe commun entre deux suffixes consécutifs de la table des suffixes.

⇒ Amélioration des performances de recherche

Définition (FM-index)

Le FM-index (Ferragina–Manzini index) est la structure résultant de la compression de la transformée de BURROWS-WHEELER.

Sa construction est basée sur la table des suffixes pour le calcul de la transformée.

Transformée de BURROWS-WHEELER du mot ROUDOUDOU$

Texte d'origine $R_0\ O_1\ U_2\ D_3\ O_4\ U_5\ D_6\ O_7\ U_8\ \$_9$

BWT $U_8\ U_5\ U_2\ D_6\ D_3\ R_0\ \$_9\ O_7\ O_4\ O_1$

SA $\$_9\ D_6\ D_3\ O_7\ O_4\ O_1\ R_0\ U_8\ U_5\ U_2$

U_8
U_5

Recherche du mot UDOU ←

Définition (Table des suffixes)

≡ tableau des suffixes du mot ordre lexicographique.

+

Définition (Table des plus long préfixes communs)

≡ longueur du plus long préfixe commun entre deux suffixes consécutifs de la table des suffixes.

⇒ Amélioration des performances de recherche

Définition (FM-index)

Le FM-index (Ferragina–Manzini index) est la structure résultant de la compression de la transformée de BURROWS-WHEELER. Sa construction est basée sur la table des suffixes pour le calcul de la transformée.

Transformée de BURROWS-WHEELER du mot ROUDOUDOU$

Texte d'origine $R_0\ O_1\ U_2\ D_3\ O_4\ U_5\ D_6\ O_7\ U_8\ \$_9$

BWT $U_8\ U_5\ U_2\ D_6\ D_3\ R_0\ \$_9\ O_7\ O_4\ O_1$

SA $\$_9\ D_6\ D_3\ O_7\ O_4\ O_1\ R_0\ U_8\ U_5\ U_2$

$O_7\ U_8$
$O_4\ U_5$

Recherche du mot UDOU ←

Définition (Table des suffixes)

≡ tableau des suffixes du mot triés par ordre lexicographique.

+

Définition (Table des plus long préfixes communs)

≡ longueur du plus long préfixe commun entre deux suffixes consécutifs de la table des suffixes.

⇒ Amélioration des performances de la recherche

Définition (FM-index)

Le FM-index (Ferragina–Manzini index) est la structure résultant de la compression de la transformée de BURROWS-WHEELER. Sa construction est basée sur la table des suffixes pour le calcul de la transformée.

Transformée de BURROWS-WHEELER du mot ROUDOUDOU$

Texte d'origine	R_0 O_1 U_2 D_3 O_4 U_5 D_6 O_7 U_8 $\$_9$
BWT	U_8 U_5 U_2 D_6 D_3 R_0 $\$_9$ O_7 O_4 O_1
SA	$\$_9$ D_6 D_3 O_7 O_4 O_1 R_0 U_8 U_5 U_2

O_7 U_8
O_4 U_5

Recherche du mot UDOU ←

Définition (Table des suffixes)

≡ tableau des suffixes du mot triés par ordre lexicographique.

Définition (Table des plus long préfixes communs)

≡ longueur du plus long préfixe commun entre deux suffixes consécutifs de la table des suffixes.

⇒ Amélioration des performances de la recherche

Définition (FM-index)

Le FM-index (Ferragina–Manzini index) est la structure résultant de la compression de la transformée de BURROWS-WHEELER. Sa construction est basée sur la table des suffixes pour le calcul de la transformée.

Transformée de BURROWS-WHEELER du mot ROUDOUDOU$

Texte d'origine	R_0	O_1	U_2	D_3	O_4	U_5	D_6	O_7	U_8	$\$_9$
BWT	U_8	U_5	U_2	D_6	D_3	R_0	$\$_9$	O_7	O_4	O_1
SA	$\$_9$	D_6	D_3	O_7	O_4	O_1	R_0	U_8	U_5	U_2

O_7 U_8
O_4 U_5

Recherche du mot UDOU ←

Définition (Table des suffixes)

≡ tableau des suffixes du mot triés par ordre lexicographique.

Définition (Table des plus long préfixes communs)

≡ longueur du plus long préfixe commun entre deux suffixes consécutifs de la table des suffixes.

⇒ Amélioration des performances de recherche

Définition (FM-index)

Le FM-index (Ferragina–Manzini index) est la structure résultant de la compression de la transformée de BURROWS-WHEELER.

Sa construction est basée sur la table des suffixes pour le calcul de la transformée.

Transformée de BURROWS-WHEELER du mot ROUDOUDOU$

Texte d'origine	R_0	O_1	U_2	D_3	O_4	U_5	D_6	O_7	U_8	$\$_9$
BWT	U_8	U_5	U_2	D_6	D_3	R_0	$\$_9$	O_7	O_4	O_1
SA	$\$_9$	D_6	D_3	O_7	O_4	O_1	R_0	U_8	U_5	U_2

D_6	O_7	U_8
D_3	O_4	U_5

Recherche du mot UDOU ⟵

Définition (Table des suffixes)

≡ tableau des suffixes du mot ordre lexicographique.

+

Définition (Table des plus long préfixes communs)

≡ longueur du plus long préfixe commun entre deux suffixes consécutifs de la table des suffixes.

⇒ Amélioration des performances de la recherche

Définition (FM-index)

Le FM-index (Ferragina–Manzini index) est la structure résultant de la compression de la transformée de BURROWS-WHEELER. Sa construction est basée sur la table des suffixes pour le calcul de la transformée.

Transformée de BURROWS-WHEELER du mot ROUDOUDOU$

Texte d'origine	R_0	O_1	U_2	D_3	O_4	U_5	D_6	O_7	U_8	$\$_9$
BWT	U_8	U_5	U_2	D_6	D_3	R_0	$\$_9$	O_7	O_4	O_1
SA	$\$_9$	D_6	D_3	O_7	O_4	O_1	R_0	U_8	U_5	U_2

D_6	O_7	U_8
D_3	O_4	U_5

Recherche du mot UDOU ⟵

Définition (Table des suffixes)

≡ tableau des suffixes du mot triés par ordre lexicographique.

Définition (Table des plus long préfixes communs)

≡ longueur du plus long préfixe commun entre deux suffixes consécutifs de la table des suffixes.

⇒ Amélioration des performances de recherche

Définition (FM-index)

Le FM-index (Ferragina–Manzini index) est la structure résultant de la compression de la transformée de BURROWS-WHEELER. Sa construction est basée sur la table des suffixes pour le calcul de la transformée.

Transformée de BURROWS-WHEELER du mot ROUDOUDOU$

Texte d'origine	R_0	O_1	U_2	D_3	O_4	U_5	D_6	O_7	U_8	$\$_9$
BWT	U_8	U_5	U_2	D_6	D_3	R_0	$\$_9$	O_7	O_4	O_1
SA	$\$_9$	D_6	D_3	O_7	O_4	O_1	R_0	U_8	U_5	U_2

U_5	D_6	O_7	U_8
U_2	D_3	O_4	U_5

Recherche du mot UDOU ⟵

Définition (Table des suffixes)

≡ tableau des suffixes du mot ordre lexicographique.

+

Définition (Table des plus long préfixes communs)

≡ longueur du plus long préfixe commun entre deux suffixes consécutifs de la table des suffixes.

⇒ Amélioration des performances de recherche

Définition (FM-index)

Le FM-index (Ferragina–Manzini index) est la structure résultant de la compression de la transformée de BURROWS-WHEELER. Sa construction est basée sur la table des suffixes pour le calcul de la transformée.

Transformée de BURROWS-WHEELER du mot ROUDOUDOU$

Texte d'origine	R_0 O_1 U_2 D_3 O_4 U_5 D_6 O_7 U_8 $\$_9$
BWT	U_8 U_5 U_2 D_6 D_3 R_0 $\$_9$ O_7 O_4 O_1
SA	$\$_9$ D_6 D_3 O_7 O_4 O_1 R_0 U_8 U_5 U_2

U_5	D_6	O_7	U_8
U_2	D_3	O_4	U_5

Recherche du mot UDOU

Alignement visuel de séquences

Recherche de motif

Extraction de motif

Contexte

Quelques programmes [presque] naïfs

Algorithme KMR

Structures d'index

Hachage

Arbres

Automates

Tables

That's All, Folks !

Merci de votre attention.

Vous pouvez rentrer chez vous.

C'est fini.

Au revoir.

Bon courage.

Ceci n'est pas un test ophtalmologique,

mais si vous arrivez à lire jusqu'au bout...

...c'est que vous avez du temps.

Profitez-en pour programmer alors ! ! !