Exemples d'applications de l'algorithme de KMP

Application 1 : périodicité

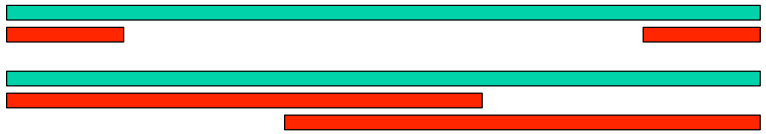
Période d'un mot T[1..*n*] : entier p tel que T[*i*]=T[*i*+*p*] pour tout *i*∈[1..*n*-*p*]

Exemples :
`abababa` : périodes 2,4,6
`bbabb` : périodes 3,4
`bbabaa` : pas de période

Problème : étant donné un mot T, trouver sa période *minimale*

Application 1 : périodicité

Un **bord** d'un mot T : facteur qui apparaît en préfixe et en suffixe



Lemme : *p* est une période de T[1..*n*] ssi T possède un bord de longueur *n*-*p*

La fonction d'échec (non-optimisée) de l'algorithme KMP calcule le bord maximal pour chaque préfixe T[1..*i*]

Application 1 : périodicité

⇒ calcul de la période minimale en temps O(*n*) à l'aide de l'algorithme de calcul de la fonction d'échec

	T =	a	b	a	b	a	c	a
q		0	1	2	3	4	5	6	7
f(*q*)		−1	0	0	1	2	3	0	1

période minimale = 7-1=6

	T =	a	b	a	a	b	a	a	b
q		0	1	2	3	4	5	6	7	8
f(*q*)		−1	0	0	1	1	2	3	4	5

période minimale = 8-5=3

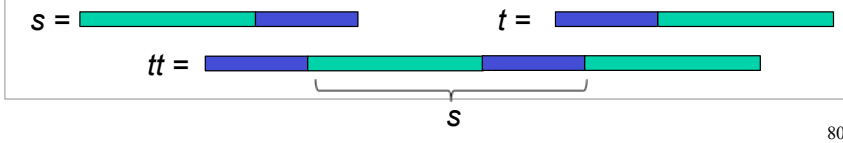
Application 2 : mots conjugués

Deux mots *s* et *t* sont conjugués si *s*=*uv* et *t*=*vu* pour certains *v* et *u*

Exemples : `lanver` et `verlan`, `bête` et `tebê`, `cigarette` et `garetteci`

Problème : étant donné deux mots *s* et *t*, tester s'ils sont conjugués

Lemme : *s* et *t* sont conjugués ssi ils sont de même longueur et *s* est un facteur de *tt* (ou *t* est un facteur de *ss*)



Application 2 : mots conjugués

⇒ test de conjugaison peut se faire en temps O(*n*) à l'aide de l'algorithme de KMP

Structures pour index.
Arbre des suffixes

Structures pour index

Index : structure de données permettant de accéder rapidement aux facteurs de texte T[1..*n*]

Ce qu'on attend d'un bon index :
- qu'il prenne peu de mémoire (idéalement O(*n*))
- qu'il puisse être construit rapidement (idéalement en O(*n*))
- qu'il permette de localiser un facteur P[1..*m*] rapidement (idéalement en O(*m*+*occ*))

Trie des suffixes (arbre des suffixes non-compacté): trie stockant tous les suffixes du texte

T = a b a b b b

suffixes de T :

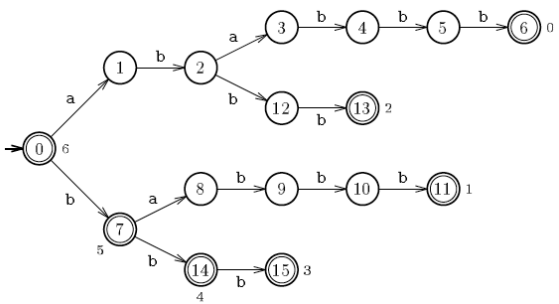
```
a b a b b b
  b a b b b
    a b b b
      b b b
        b b
          b
```


Trie des suffixes (arbre des suffixes non-compacté): trie stockant tous les suffixes du texte

T = a b a b b b

suffixes de T :

```
a b a b b b
  b a b b b
    a b b b
      b b b
        b b
          b
```

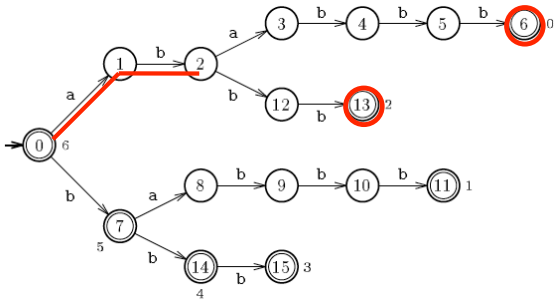


Observation : il est facile de localiser un motif à l'aide d'un trie des suffixes : « épeler » le motif en partant de la racine ; si réussi, rapporter tous les nœuds terminaux se trouvant dans le sous-arbre correspondant

Observation : il est facile de localiser un motif à l'aide d'un trie des suffixes : « épeler » le motif en partant de la racine ; si réussi, rapporter tous les nœuds terminaux se trouvant dans le sous-arbre correspondant

T = a b a b b b

Exemple :
localisation du motif P = a b



Arbre des suffixes

Problème : le trie des suffixes peut avoir la taille $O(n^2)$ (n la longueur de texte)

Exemple : le trie des suffixes pour $a^k b^k$ a $\Theta(k^2)$ nœuds

Arbre des suffixes

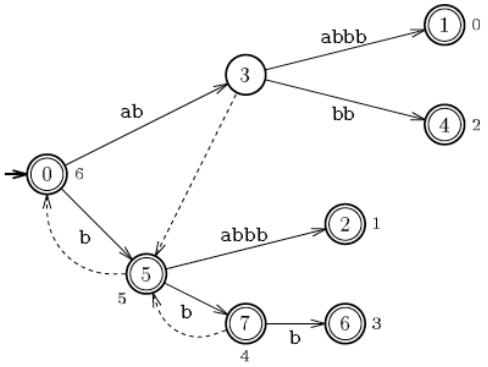
Problème : le trie des suffixes peut avoir la taille $O(n^2)$ (n la longueur de texte)

Exemple : le trie des suffixes pour $a^k b^k$ a $\Theta(k^2)$ nœuds

Idée : compacter le trie des suffixes : « compresser » chaque branche sans bifurcation et sans nœuds terminaux en un seul arc

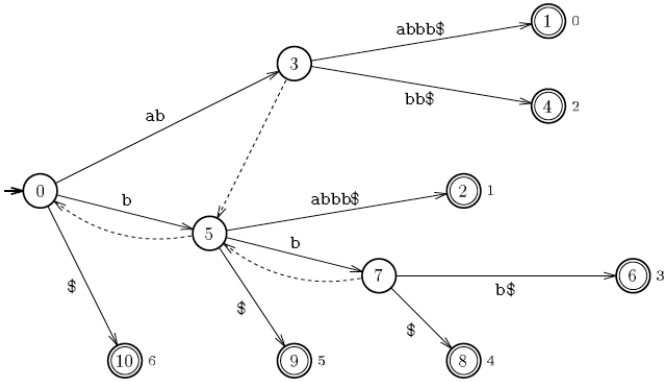
Arbre des suffixes

trie des suffixes compacté de `a b a b b b` :



Arbre des suffixes

trie des suffixes compacté de `a b a b b b $` :



Arbre des suffixes

Théorème : le trie des suffixes compacté ne peut pas avoir plus de 2*n* nœuds

Preuve : il ne peut pas y avoir plus de *n* feuilles (une feuille par suffixe). Comme chaque nœud interne de l'arbre est branchant ou terminal (ou les deux), il n'y a au plus *n* nœuds internes branchant.

Arbre des suffixes

Théorème : le trie des suffixes compacté ne peut pas avoir plus de 2*n* nœuds

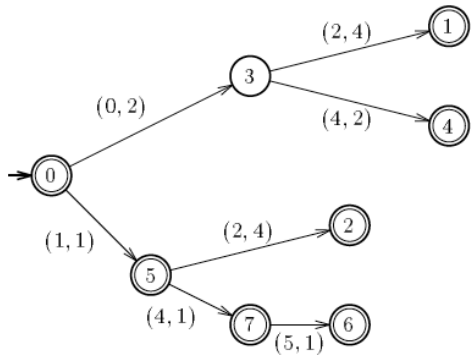
Preuve : il ne peut pas y avoir plus de *n* feuilles (une feuille par suffixe). Comme chaque nœud interne de l'arbre est branchant ou terminal (ou les deux), il n'y a au plus *n* nœuds internes branchant.

Problème : la représentation n'est pas linéaire car la taille totale des étiquettes reste quadratique dans le pire cas

Remède : remplacer chaque étiquette par un couple (début,longueur) ➠ arbre des suffixes

Arbre des suffixes

arbre des suffixes de `a b a b b b :`

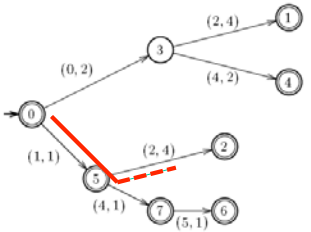


Arbre des suffixes

- occupe O(*n*) mémoire
- peut être construit en temps O(*n*)
  - Weiner (1973) (parcours du texte de droite à gauche)
  - McCreight (1976)
  - Ukkonen (1992) algorithme ***on-line***
- motif P[1..*m*] peut être localisé en temps O(*m*)

T = `a b a b b b`

P = `b a b`



Arbre des suffixes : applications

• avec un pré-traitement en O(n) les requêtes suivantes peuvent être répondues en O(m) :
  • première occurrence de P
  • dernière occurrence de P
  • nombre d'occurrences de P

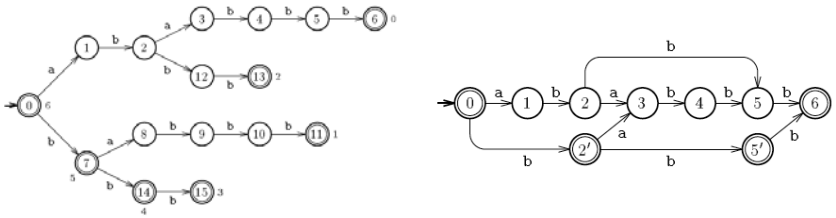
• le plus long facteur répété dans T peut être trouvé en O(n)

• le plus long facteur commun à deux mots peut être trouvé en O(n_1+n_2)

Structures pour index.
Automate des suffixes
(*DAWG: Directed Acyclic Word Graph*)

Automate des suffixes

Autre façon de compacter le trie des suffixes



Automate des suffixes : automate minimal acceptant tous les suffixes de T. Contient ≤2n-2 états et ≤3n-4 transitions

Peut être construit en temps O(n) [Blumer et al 1983, Crochemore 1984]

Structures pour index.
Table des suffixes
(*Suffix Array*)

Table des suffixes

UMLV ©

A T C A C A T C A T C A

0 1 2 3 4 5 6 7 8 9 1011

Table des suffixes

UMLV ©

A T C A C A T C A T C A

0 1 2 3 4 5 6 7 8 9 1011

	pos
ATCACATCATCA	*0*
TCACATCATCA	*1*
CACATCATCA	*2*
ACATCATCA	*3*
CATCATCA	*4*
ATCATCA	*5*
TCATCA	*6*
CATCA	*7*
ATCA	*8*
TCA	*9*
CA	*10*
A	*11*

Table des suffixes

UMLV ©

A T C A C A T C A T C A

0 1 2 3 4 5 6 7 8 9 1011

	pos
ATCACATCATCA	*0*
TCACATCATCA	*1*
CACATCATCA	*2*
ACATCATCA	*3*
CATCATCA	*4*
ATCATCA	*5*
TCATCA	*6*
CATCA	*7*
ATCA	*8*
TCA	*9*
CA	*10*
A	*11*

rang		pos
0	A	*11*
1	ACATCATCA	*3*
2	ATCA	*8*
3	ATCACATCATCA	*0*
4	ATCATCA	*5*
5	CA	*10*
6	CACATCATCA	*2*
7	CATCA	*7*
8	CATCATCA	*4*
9	TCA	*9*
10	TCACATCATCA	*1*
11	TCATCA	*6*

Table des suffixes

UMLV ©

A T C A C A T C A T C A

0 1 2 3 4 5 6 7 8 9 1011

	pos
ATCACATCATCA	*0*
TCACATCATCA	*1*
CACATCATCA	*2*
ACATCATCA	*3*
CATCATCA	*4*
ATCATCA	*5*
TCATCA	*6*
CATCA	*7*
ATCA	*8*
TCA	*9*
CA	*10*
A	*11*

rang		pos
0	A	*11*
1	ACATCATCA	*3*
2	ATCA	*8*
3	ATCACATCATCA	*0*
4	ATCATCA	*5*
5	CA	*10*
6	CACATCATCA	*2*
7	CATCA	*7*
8	CATCATCA	*4*
9	TCA	*9*
10	TCACATCATCA	*1*
11	TCATCA	*6*

Table des suffixes

A T C A C A T C A T C A

0 1 2 3 4 5 6 7 8 9 1011

	pos
ATCACATCATCA	*0*
TCACATCATCA	*1*
CACATCATCA	*2*
ACATCATCA	*3*
CATCATCA	*4*
ATCATCA	*5*
TCATCA	*6*
CATCA	*7*
ATCA	*8*
TCA	*9*
CA	*10*
A	*11*

rang		pos
0	A	*11*
1	ACATCATCA	*3*
2	ATCA	*8*
3	ATCACATCATCA	*0*
4	ATCATCA	*5*
5	CA	*10*
6	CACATCATCA	*2*
7	CATCA	*7*
8	CATCATCA	*4*
9	TCA	*9*
10	TCACATCATCA	*1*
11	TCATCA	*6*

A T C

Table des suffixes

- Recherche de motif en temps O($m \cdot \log(n)$), peut être optimisé en O($m+\log(n)$)
- Construction en temps O($n \cdot \log(n)$), peut être optimisé en O(n) (2003)