Arbre des suffixes*

Arbre des suffixes: Structure de données reflétant les caractéristiques internes des séquences

Application naturelle: Recherche exacte d'un mot dans un texte

- Phase de prétraitement: Construction de l'arbre des suffixes: $\mathcal{O}(n)$ en temps et en espace pour un texte de taille n
- Phase de recherche: En temps $\mathcal{O}(m)$ pour un mot de taille m

Autres applications: Recherche de répétitions, recherche de palindromes

* Transparents basés sur les notes de cours de Nadia El-Mabrouk

Arbre des suffixes: Définition

Un arbre des suffixes pour un texte T, dénoté $\mathcal{T}$, est un arbre enraciné et orienté, tel que:

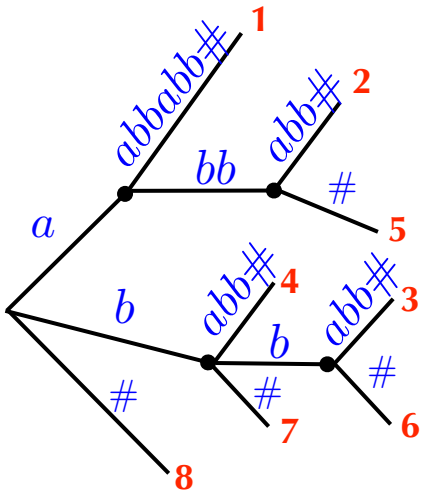
- $\mathcal{T}$ à n feuilles numérotées de 1 à n
- Chaque noeud interne a au moins 2 fils
- Chaque arc est étiquetté par un facteur (sous-mot) de T
- Deux arcs sortant d'un même noeud ont des étiquettes débutant par des caractères différents
- Le chemin de la racine à une feuille i est étiquetté par le suffixe $T[i..n]$

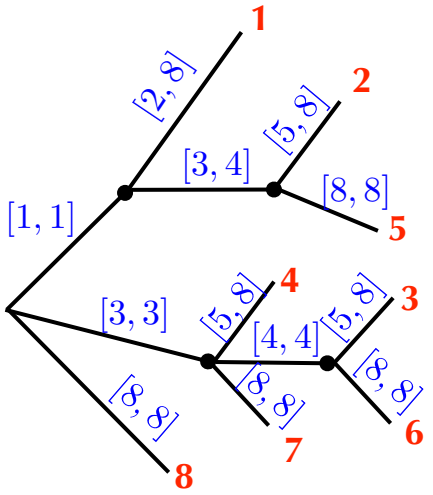
Problème: Si un suffixe coïncide avec un facteur du texte, aucune feuille ne correspondra au suffixe

Solution: On ajoute un caractère artificiel à la fin du texte. Par exemple, #

Espace en $\mathcal{O}(n)$

$$T = \overset{1}{a}\ \overset{2}{a}\ \overset{3}{b}\ \overset{4}{b}\ \overset{5}{a}\ \overset{6}{b}\ \overset{7}{b}\ \overset{8}{\#}$$





Pour réduire l'espace, représenter chaque étiquette par deux positions plutôt que par un facteur

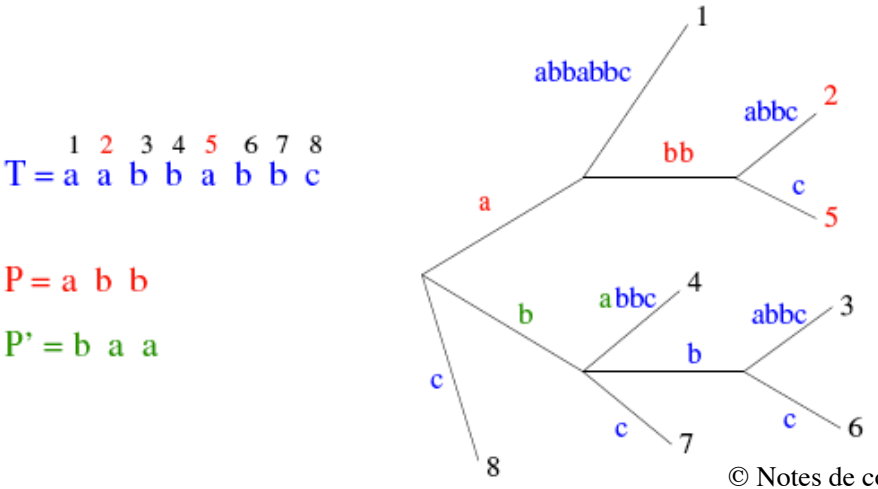
Recherche exacte de P dans T

- Construire l'arbre des suffixes $\mathcal{T}$ de T en temps $\mathcal{O}(n)$

- En partant de la racine de $\mathcal{T}$ lire les caractères de P en suivant un chemin unique:

  - Si les caractères de P ne sont pas épuisés et qu'on ne peut plus rien lire alors, il y a aucune occurrence de P dans T: temps $\mathcal{O}(m)$

  - Si on réussit à lire complètement P alors les feuilles du sous-arbre déterminé par P donne toutes les positions du mot dans T: temps $\mathcal{O}(m+k)$ où k est le nombre d'occurences de P dans T



Construction naïve

- Insertion successive des suffixes du plus long au plus court
- Complexité: $\mathcal{O}(n^2)$

Construction en temps linéaire

- Weiner 1973: Premier algorithme linéaire

  Weiner, P., "Linear pattern matching algorithm", *14th Annual IEEE Symposium on Switching and Automata Theory*, pp. 1–11, 1973

- McCreight et Edward 1976: Algorithme linéaire utilisant moins d'espace

  McCreight, Edward M , "A Space-Economical Suffix Tree Construction Algorithm", *Journal of the ACM* **23** (2): 262–272, 1976

- Ukkonen 1995: Algorithme linéaire, plus simple

  Ukkonen, E., "On-line construction of suffix trees", *Algorithmica* **14** (3): 249–260, 1995

Algorithme d'Ukkonen

- Construction d'arbres de suffixes implicites: Arbre pour T et non pour $T\#$. Contient moins d'information que l'arbre des suffixes.

- $\mathcal{I}_i$ arbre implicite pour $T[1..i]$

Méthode générale:

- n phases:

  Phase $i+1$: Construction de $\mathcal{I}_{i+1}$ à partir de $\mathcal{I}_i$ en ajoutant $T[i+1]$

  $i+1$ extensions, une pour chaque suffixe de $T[1..i+1]$:

  $$T[1..i+1],\ T[2..i+1],\quad \ldots \quad ,T[i+1..i+1]$$

Algorithme d'Ukkonen (suite)

Extensions $T[j..i+1]$:

Règle 1: Le chemin étiquetté par $T[j..i]$ se termine à une feuille:

Ajouter $T[i+1]$ à la fin de cette étiquette

Règle 2: Le chemin étiquetté par $T[j..i]$ ne se termine pas à une feuille et il n'y a pas de chemin d'étiquette $T[j..i]T[i+1]$:

Créer une bifurcation supplémentaire à la fin de $T[j..i]$ d'étiquette $T[i+1]$

Règle 3: Il existe un chemin d'étiquette $T[j..i+1]$:

Rien à faire

Complexité: Une fois le chemin $T[j..i]$ trouvé, l'extension se fait en temps constant

Approche naïve: Lire tous les caractères: extension j de la phase $i+1$ en $\mathcal{O}(i-j+1)$

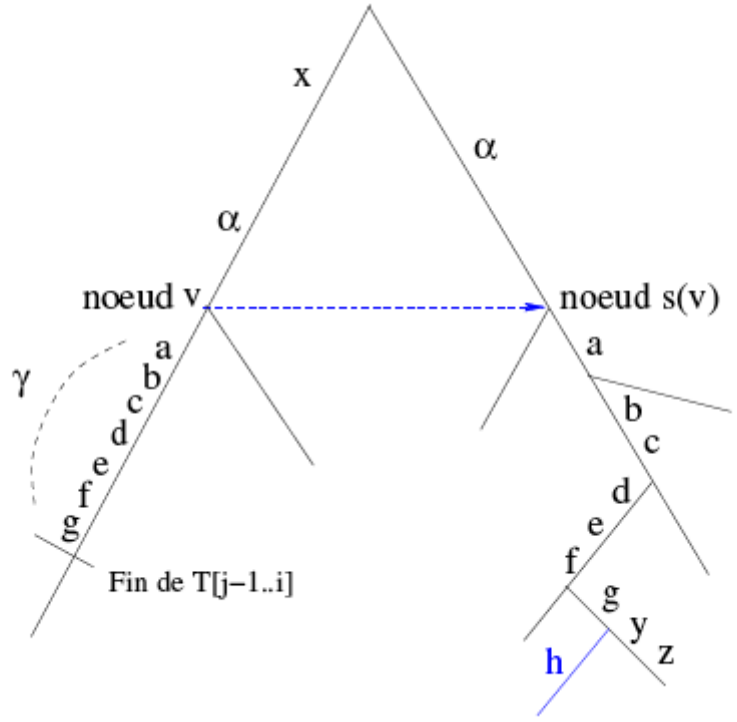
Construction de $\mathcal{I}_{i+1}$: $\mathcal{O}(i^2)$

Algorithme complet i.e. construction de $\mathcal{I}_1$ à $\mathcal{I}_n$: $\mathcal{O}(n^3)$

Liens suffixes

Soit v un noeud interne d'étiquette $x\alpha$ (x: caractère, α : mot). S'il existe un noeud $s(v)$ d'étiquette α, alors créer un lien suffixe entre v et $s(v)$

Un noeud interne d'étiquette x à un lien suffixe vers la racine. La racine n'a pas de lien suffixe



Algorithme d'Ukkonen et lien suffixe

Extensions $T[j..i+1]$:

Pour $j = 1$:

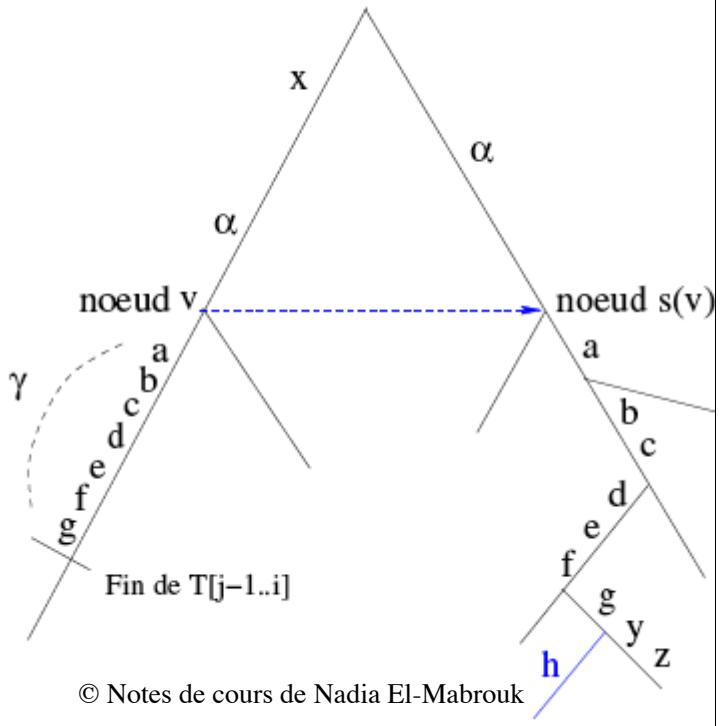
f feuille d'étiquette $T[1..i]$

Prolonger l'étiquette de f par $T[i+1]$

Pour $j > 1$:

$v \leftarrow$ Dernier noeud sur le chemin d'étiquette $T[j-1..i]$ qui est soit la racine ou possède un lien suffixe

$\gamma \leftarrow$ Mot entre v et la fin de $T[j-1..i]$



Si v n'est pas la racine, suivre le lien suffixe de v à $s(v)$ et continuer dans l'arbre en suivant les caractères de γ

Si v est la racine, parcourir l'arbre en suivant les caractères de $T[j..i]$

Utiliser les règles d'extensions pour ajouter le caractère $T[i+1]$

$w \leftarrow$ noeud crée (s'il y a lieu) à l'extension $j-1$

Créer le lien suffixe $(w, s(w))$

Complexité

Remarque: Pour aligner γ, sauter de noeud en noeud en considérant la taille des étiquettes

- Chaque phase i de l'algorithme prend un temps $\mathcal{O}(i)$
- Algorithme complet en $\mathcal{O}(n^2)$

Optimisation

À chaque phase de l'algorithme, effectuer un nombre réduit d'extensions:

Observation 1: Lorsque la règle 3 est appliquée à l'extension j, alors elle est appliquée à toutes les extensions suivantes: $j+1, \ldots, i+1$

Observation 2: Une feuille créée à une étape, reste une feuille jusqu'à la fin de l'algorithme

Algorithme simplifié pour la phase $i+1$:

Ajouter $T[i+1]$ à la fin des étiquettes des k feuilles de $\mathcal{T}_i$

Calculer explicitement les extensions pour $j' = k+1$ à j^*, où j^* est la première extension pour laquelle la règle 3 est appliquée ou $i+1$

Théorème: La construction des arbres implicites $\mathcal{T}_1$ à $\mathcal{T}_n$ à l'aide de l'algorithme simplifié prend un temps $\mathcal{O}(n)$

Finalement, l'arbre implicite $\mathcal{T}_n$ est transformé en arbre des suffixes en ajoutant le caractère # à la fin du texte et en appliquant une dernière fois l'algorithme d'Ukkonen.

Arbre des suffixes généralisé

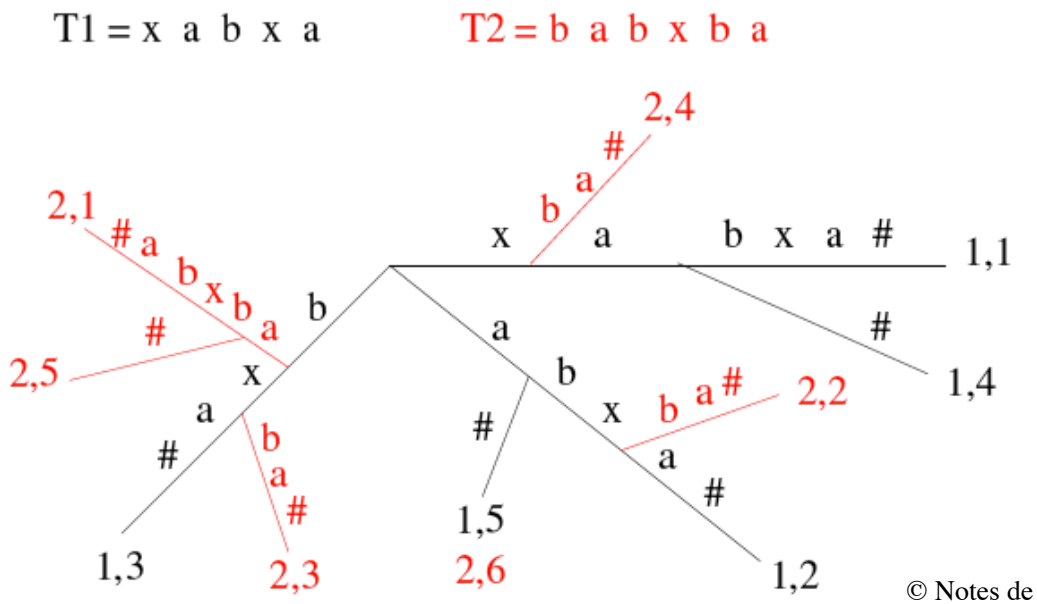
Arbre des suffixes pour un ensemble de séquences $\{T_1, T_2, \ldots, T_k\}$

Simple extension de l'algorithme d'Ukkonen

2 subtilités:

Chaque feuille doit être étiquettée par les positions des suffixes dans les séquences concernées

Si on étiquette les arcs par des positions plutôt que des facteurs, il faut garder en mémoire les positions dans toutes les séquences concernées



Différentes utilisations de l'arbre des suffixes

1) Recherche du plus long facteur commun à deux séquences:

Par exemple, pour les mots $xabxa\#$ et $babxba\#$, le plus long facteur commun est abx

Soient S_1 et S_2 les deux séquences pour lesquelles nous voulons trouver ce facteur:

Méthode:

1) Construire l'arbre des suffixes généralisé $\mathcal{T}$ pour $\{S_1, S_2\}$

2) Marquer chaque noeud interne v par une étiquette: 1 (respectivement 2) si toutes les feuilles du sous-arbre de racine v représentent des suffixes de S_1(resp. S_2) ou (1,2) si les feuilles du sous-arbre représentent des suffixes des deux séquences

3) Trouver le noeud d'étiquette (1,2) de profondeur en caractères maximale

Différentes utilisations de l'arbre des suffixes

1) Recherche du plus long facteur commun à deux séquences:

Complexité:

Si $n = |S_1| + |S_2|$ alors

On peut marquer tous les noeuds internes en effectuant un parcours suffixe de l'arbre en $\mathcal{O}(n)$

On peut trouver le noeud d'étiquette (1,2) de profondeur en caractères maximale en effectuant un parcours préfixe de l'arbre en $\mathcal{O}(n)$

Fait intéressant: En 1970, Knuth avait conjecturé qu'il était impossible de résoudre ce problème en temps linéaire.

Différentes utilisations de l'arbre des suffixes

2) Recherche de répétitions maximales:

Une répétition maximale α est une séquence qui apparaît au moins à 2 positions différentes du texte T et si on rallonge vers la gauche ou vers la droite en ces positions, on n'a plus de répétition

Si α est une répétition maximale $\Rightarrow$ α est l'étiquette d'un noeud interne de l'arbre des suffixes $\mathcal{T}$ de T

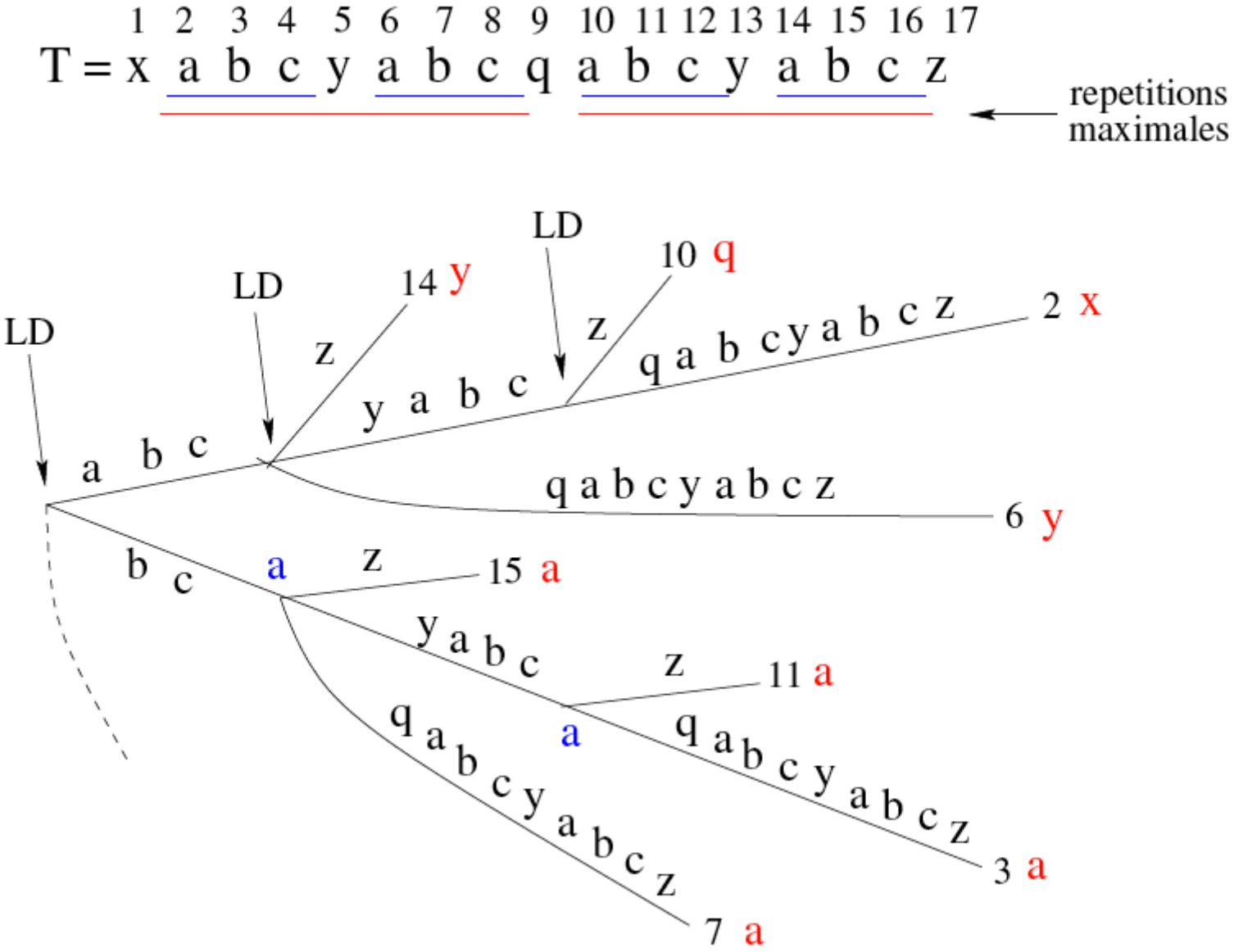
α est l'étiquette d'un noeud interne de l'arbre des suffixes $\mathcal{T}$ de T $\not\Rightarrow$ α est une répétition maximale

f : feuille représentant le suffixe $T[i..n]$

$T[i-1]$: caractère gauche de f

Un noeud interne v est varié à gauche (left diverse) si au moins deux feuilles du sous-arbre de racine v ont des caractères gauches différents.

Si un noeud v est varié à gauche, alors tous ses ancêtres le sont aussi.



Différentes utilisations de l'arbre des suffixes

2) Recherche de répétitions maximales:

Théorème:

α est une répétition maximale $\iff$ α est l'étiquette d'un noeud varié à gauche

Théorème: Toutes les répétitions maximales d'une séquence T peuvent être trouvées en temps $\mathcal{O}(n)$

- Construire l'arbres des suffixes en gardant en mémoire le caractère gauche de chaque feuille

- Faire un parcours suffixe de l'arbre pour étiquetter les noeuds internes de l'arbre soit par "LD" si le noeud est varié à gauche ou par l'unique caractère gauche des feuilles, sinon

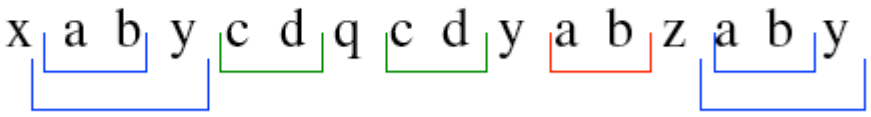
Différentes utilisations de l'arbre des suffixes

3) Recherche de répétitions supermaximales:

Répétition supermaximale: Répétition maximale qui n'est jamais comme facteur d'une autre répétition

Répétition presque supermaximale: Répétition maximale qui apparaît à au moins une position dans le texte où elle n'est pas facteur d'une autre répétition maximale. Cette position est appelée la position témoin

Exemple



aby et cd sont des répétitions supermaximales

ab est un répétition presque supermaximale: position témoin en rouge

Différentes utilisations de l'arbre des suffixes

3) Recherche de répétitions supermaximales:

Théorème: Un noeud v varié à gauche représente une répétition supermaximale ssi tous les fils de v sont des feuilles et tous les caractères gauches de ces feuilles sont différents

Complexité: Toutes les répétitions supermaximales d'un texte T de longueur n peuvent être trouvées en temps $\mathcal{O}(n)$

Théorème: Un noeud v varié à gauche représente une répétition presque supermaximale ssi v n'est pas une répétition supermaximale et si un des fils de v est une feuille dont le caractère gauche est différent de toutes les autres feuilles du sous-arbre enraciné en v

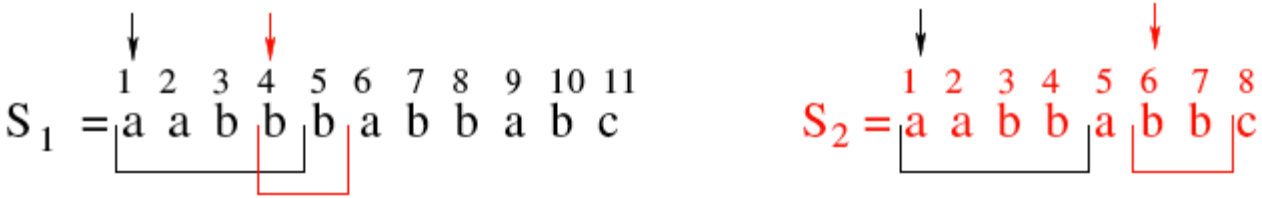
Complexité: Toutes les répétitions presque supermaximales d'un texte T de longueur n peuvent être trouvées en temps $\mathcal{O}(n)$

Différentes utilisations de l'arbre des suffixes

4) Plus longue extension commune:

Soient S_1 et S_2 deux séquences, $|S_1| = n_1, |S_2| = n_2$

La plus longue extension commune pour (i, j) : plus long préfixe commun de $S[i..n_1]$ et $S[j..n_2]$



Méthode:

1) Construire l'arbre des suffixes généralisé pour S_1 et S_2 et conserver la profondeur en caractères de chaque noeud.

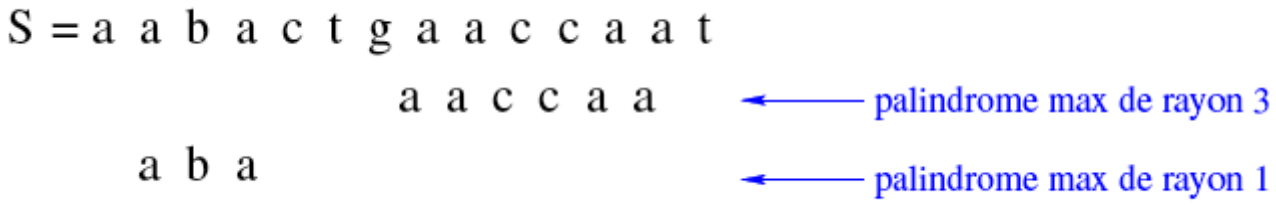
2) Pour tout (i, j), trouver le dernier ancêtre commun (lowest commun ancestor: lca) v des feuilles i, j. La profondeur en caractères de v est la plus longue extension commune de (i, j).

Différentes utilisations de l'arbre des suffixes

5) Recherche de palindromes:

Palindrome P de rayon k de S : Sous-séquence de S, telle que si on commence la lecture au milieu de cette séquence (en excluant le caractère du milieu si $|P|$ est impair), la lecture des k caractères vers la gauche ou vers la droite donne le même mot.

Le palindrome est dit maximal s'il ne peut être étendu.



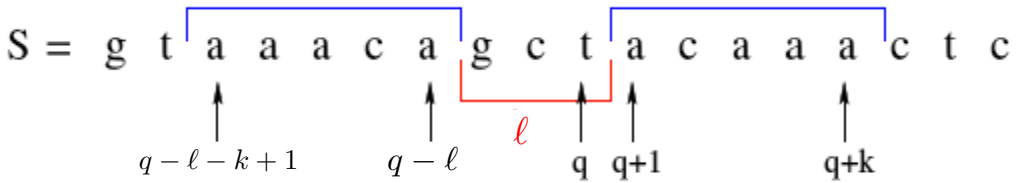
Palindrome séparé de ℓ caractères:

a b a a g t c a a b a : palindrome de rayon 4 séparé par 3 caractères

Différentes utilisations de l'arbre des suffixes

5) Recherche de palindromes:

Problème: Trouver tous les palindromes maximaux d'une séquence S séparés par ℓ caractères



Méthode:

Construire l'arbre des suffixes généralisés de S et S^R

Pour tout ℓ, pour q de $\ell+1$ à $m-1$, la plus longue extension commune (lce) de la paire $(q+1, m-q+\ell+1)$ où $q+1$ est l'indice dans S et $m-q+\ell+1$ celui dans S^R

Si la longueur du lce $= k \neq 0$, alors on a trouvé un palindrome de rayon k, séparé de ℓ caractères, dont la deuxième moitié commence en $q+1$

Différentes utilisations de l'arbre des suffixes

5) Recherche de palindromes:

Palindromes au sens biologique:

a a a c a g c t t g t t t plutôt que

a a a c a g c t a c a a a

Méthode:

Même algorithme mais en considérant $c(S^r)$ plutôt que S^r

$$S = \text{a g a t a g c c t g a}$$

$$S^r = \text{a g t c c g a t a g a}$$

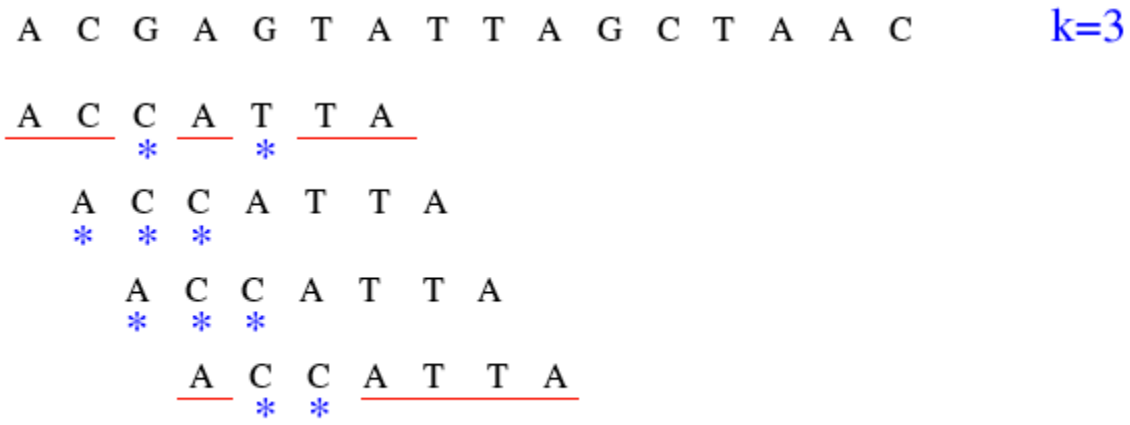
$$c(S^r) = \text{t c a g g c t a t c t}$$

Différentes utilisations de l'arbre des suffixes

6) Recherche avec "mismatches":

Recherche de P de taille m dans T de taille n à k mismatches près

Idée: À chaque position j dans T, exécuter au plus k recherches de lce



Différentes utilisations de l'arbre des suffixes

6) Recherche avec "mismatches":

Algorithme:

(1) Pour tout j faire

(2) $\quad i \leftarrow 1$; $j' \leftarrow j$; $count \leftarrow 0$;

(3) $\quad$ Tant que $count \leq k$ et $i \leq m$ faire

(3) $\quad\quad l \leftarrow$ taille de la plus longue extension des positions i dans P et j' dans T;

(4) $\quad\quad i \leftarrow i + l$;

(5) $\quad\quad$ Si $i > m$, occurrence de P à la position i;

(6) $\quad\quad$ Si non $count \leftarrow count + 1$; $j' \leftarrow j' + l + 1$;

Complexité en temps: Pour chaque position j dans T, $O(k)$ recherches de plus longues extensions $\Longrightarrow O(kn)$