Module HMIN121M

Algorithme d'Ukkonen

Construction d'un arbre des suffixes en temps linéaire

Sèverine Bérard

12 novembre 2020

Institut des Sciences de l'Evolution-Montpellier

ISE-M – FDS, Université de Montpellier

Plan du cours

Introduction

- Rappel : 3 algorithmes en temps linéaires pour construire l'arbre des suffixes : [Weiner, 73], [McCraight, 76] et [Ukkonen, 95]
- Algorithme d'Ukkonen : le plus simple à comprendre et propriété *online*
- https://www.cs.helsinki.fi/u/ukkonen/

Esko Ukkonen

- PhD, Professor (Emeritus) of Computer Science
- *Address*: Department of Computer Science, PO Box 68, FI-00014 University of Helsinki, Finland
- *Street address*: Pietari Kalmin katu 5, Helsinki
- *Phone*: +358 50 4151712 (mobile), +358 294151280 (office), +358 9 4128524 (home)
- *Fax*: +358 294151120
- *Email*: esko.ukkonen [at] helsinki.fi
- *Office hour/vastaanotto*: By appointment, room D211, Exactum Building

- https://en.wikipedia.org/wiki/Esko_Ukkonen

Esko Ukkonen

From Wikipedia, the free encyclopedia

Esko Juhani Ukkonen (b. 1950) is a Finnish theoretical computer scientist known for his contributions to string algorithms, and particularly for Ukkonen's algorithm[1] for suffix tree construction. He is a professor at the University of Helsinki.

Plan du cours

Arbres des suffixes implicites

L'algorithme d'Ukkonen construit une séquence d'arbres des suffixes implicites, le dernier étant ensuite converti en le vrai arbre des suffixes de la chaîne S.

Définition

Un *arbre des suffixes implicite* (STI) pour une chaîne S est un arbre obtenu à partir de l'arbre des suffixes de $S\$$

- en retirant chaque copie terminale du caractère \$ des étiquettes des arêtes de l'arbre
- en retirant ensuite chaque arête qui n'a plus d'étiquette
- et enfin en retirant chaque nœud qui n'a pas au moins deux enfants

On note $\mathcal{I}_i$ l'arbre des suffixes implicite de la chaîne $S[1..i]$ pour tout i de 1 à n

Propriétés

- Nombre de feuilles de l'arbre des suffixes implicite de S par rapport à l'arbre des suffixes de $S\$$?

Exemple

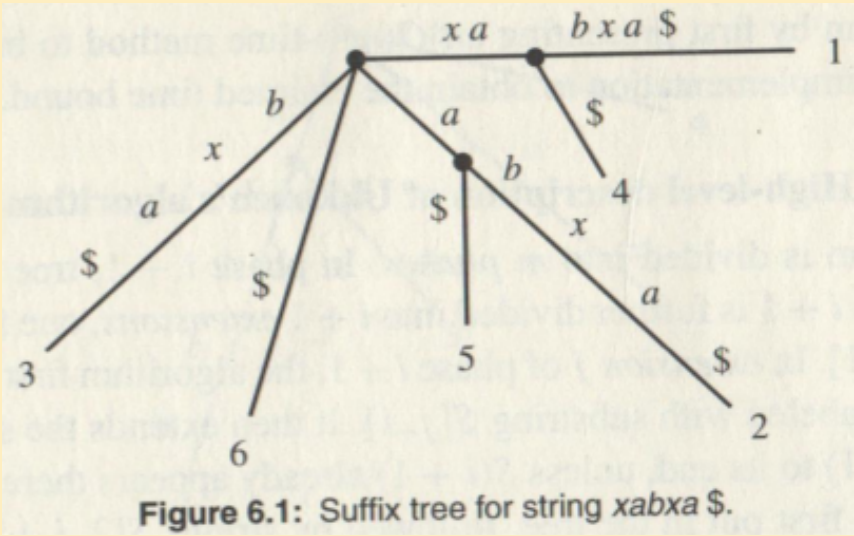


Figure 6.1: Suffix tree for string *xabxa* $.

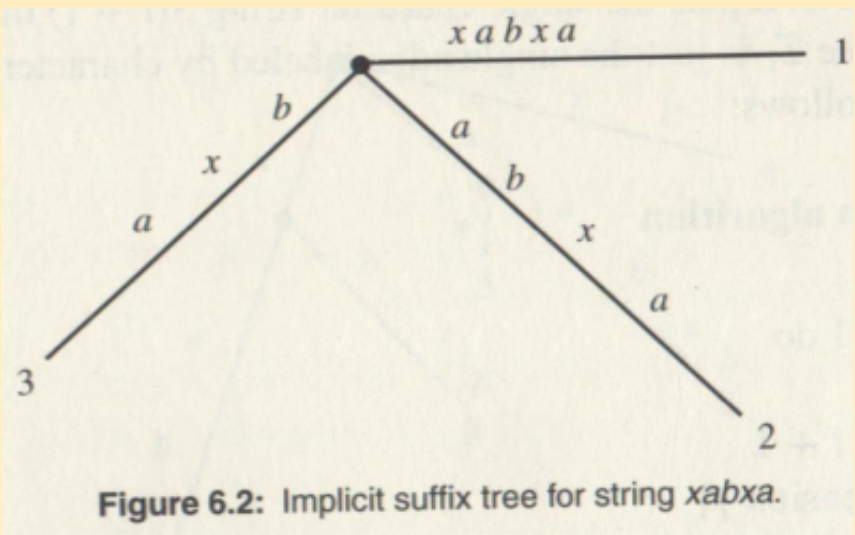


Figure 6.2: Implicit suffix tree for string *xabxa*.

- Même si un arbre des suffixes implicite n'a pas une feuille pour chaque suffixe, il encode bien tous les suffixes

Plan du cours

Une première version en $O(n^3)$

L'algorithme d'Ukkonen construit un STI $\mathcal{I}_i$ pour chaque préfixe $S[1..i]$ de S, en partant de $\mathcal{I}_1$ et jusqu'à $\mathcal{I}_n$. Le *"vrai"* arbre des suffixes est construit à partir de $\mathcal{I}_n$, et le temps total est $O(n)$

- L'algorithme d'Ukkonen est divisé en n phases : lors de la phase $i+1$, l'arbre $\mathcal{I}_{i+1}$ est construit à partir de $\mathcal{I}_i$
- Chaque phase $i+1$ est subdivisée en $i+1$ extensions, une pour chacun des $i+1$ suffixes de $S[1..i+1]$
- Dans une extension j de la phase $i+1$, on cherche la fin du chemin depuis la racine étiqueté avec $S[j..i]$ puis y ajoute le caractère $S[i+1]$ s'il n'y est pas déjà
- L'arbre $\mathcal{I}_1$ est constitué juste d'un arc étiqueté parle caractère $S[1]$

Algorithme : Algorithme d'Ukkonen à un haut niveau

Construire l'arbre $\mathcal{I}_1$;

pour *i de* 1 *jusqu'à* $n-1$ **faire**

`/* Commencer la phase` $i+1$ `*/`

pour *j de* 1 *jusqu'à* $i+1$ **faire**

`/* Commencer l'extension` j `*/`

Trouver la fin du chemin depuis la racine étiqueté avec $S[j..i]$ dans l'arbre courant ;

Si nécessaire, étendre ce chemin en ajoutant le caractère $S[i+1]$, assurant ainsi que la chaîne $S[j..i+1]$ est dans l'arbre ;

Règles d'extension des suffixes

Soit $S[j..i] = \beta$ un suffixe de $S[1..i]$

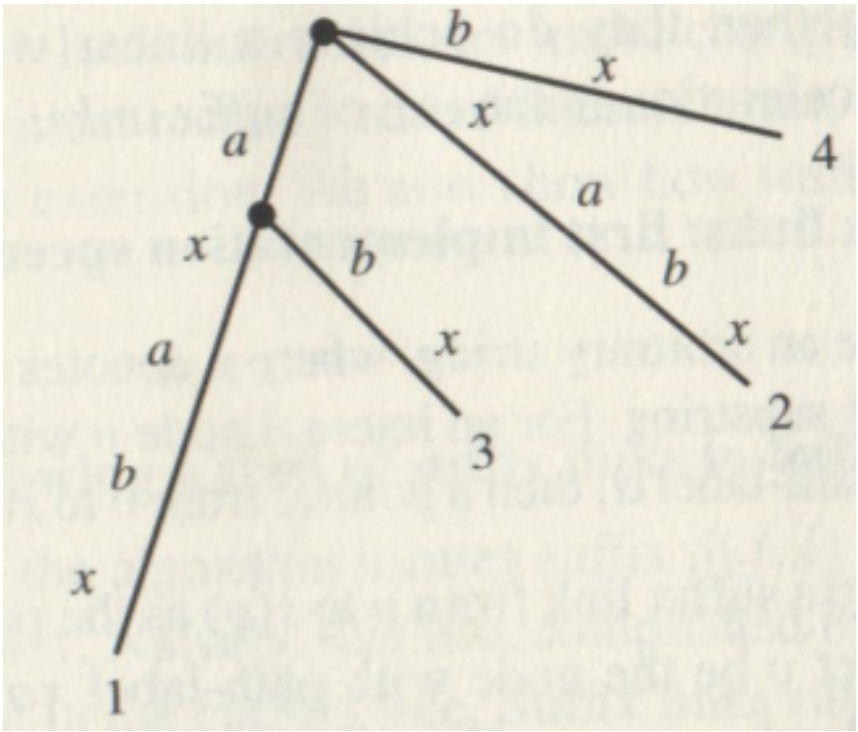
Dans l'extension j, quand l'algorithme trouve la fin du chemin β dans l'arbre courant, on étend β pour être sûr que le suffixe $\beta S[i+1]$ est dans l'arbre selon une des 3 règles suivantes :

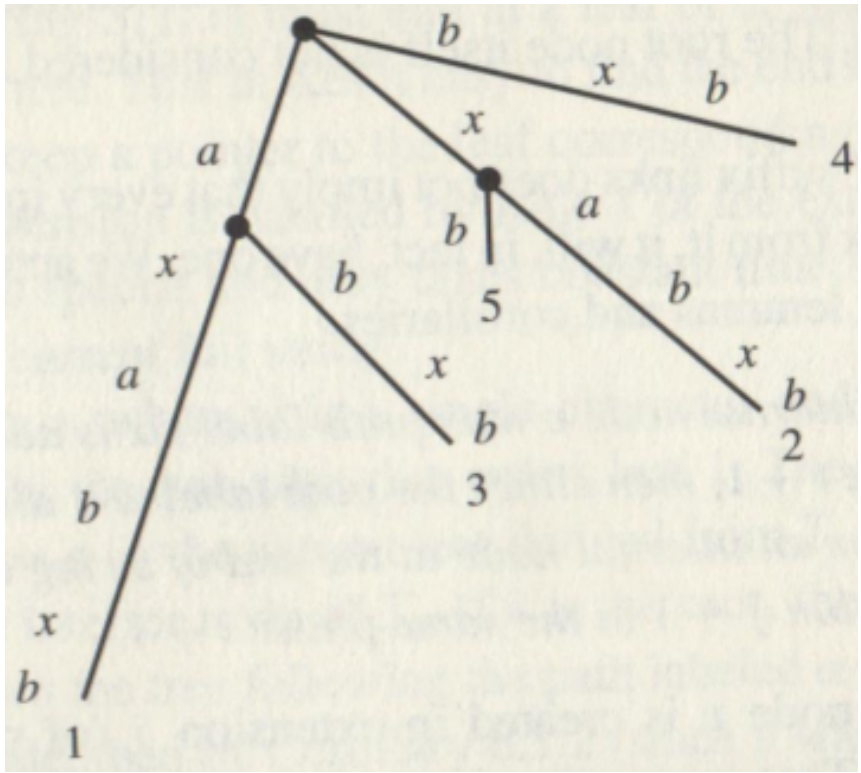
1. Si β termine sur une feuille, alors le caractère $S[i+1]$ est ajouté à la fin de l'étiquette de l'arc pointant sur cette feuille

2. Si aucun chemin à la fin de β ne commence par $S[i+1]$, alors un nouvel arc pointant sur une nouvelle feuille j et commençant à la fin de β est créé et étiqueté avec $S[i+1]$
   Un nouveau nœud doit également être créé si β finit au milieu d'un arc

3. S'il existe un chemin à la fin de β commençant par $S[i+1]$, ne rien faire

STI pour $S = axabxb$ avant et après l'insertion du dernier b





Avant Après

Quand le dernier caractère b est ajouté à la chaîne :

- les 4 premiers suffixes sont étendus en application de la Règle 1
- le 5^e suffixe est étendu en application de la Règle 2
- et le 6^e, en application de la Règle 3

Plan du cours

Un point sur la complexité

- Règles d'extension : trouver la fin du suffixe β dans l'arbre courant, le reste est en temps constant
- Il faut trouver toutes les fins des $i+1$ suffixes de $S[1..i]$ à chaque phase $i+1$
- Naïvement, on peut trouver la fin de β en $O(|\beta|) \Rightarrow$
  - extension j de la phase $i+1$ prendrait $O(i+1-j)$
  - $\mathcal{I}_{i+1}$ serait construit depuis $\mathcal{I}_i$ en $O(i^2)$
  - $\mathcal{I}_n$ serait construit en $O(n^3)$
- Pire que l'algorithme naïf en $O(n^2)$!

Les liens suffixes : une première accélération

Définition

Soit $x\alpha$ un chaîne quelconque, où x est un caractère et α une chaîne éventuellement vide
Pour un nœud interne v avec une étiquette-chemin $x\alpha$, s'il existe un autre nœud $s(v)$ avec une étiquette-chemin α, alors on appelle *lien suffixe* un pointeur de v à $s(v)$

- Notation : $(v, s(v))$
- Cas particulier : α vide, alors le lien suffixe d'un nœud $x\alpha$ pointe vers la racine
- Tous les nœuds internes des STI ont des liens suffixes

Propriétés des liens suffixes 1/2

Lemme 1

Si un nouveau nœud interne v avec une étiquette-chemin $x\alpha$ est ajouté dans l'arbre courant lors d'une extension j d'une phase $i+1$, alors soit :

- le chemin étiqueté α termine déjà à un nœud interne de l'arbre courant
- ou un nœud interne à la fin de la chaîne α va être créé (par les règles d'extensions) à l'extension $j+1$ de la même phase $i+1$

Preuve :

- Un nœud interne ne peut être créé que par la Règle 2
- $\Rightarrow$ dans l'ext. j, le chemin $x\alpha$ continuait avec un caractère autre que $S[i+1]$, disons c
- Il existe donc déjà un chemin α dans l'ext. $j+1$ se poursuivant par c
  1. soit seulement par $c \Rightarrow$ *Règle 2 crée* $s(v)$ *à la fin du chemin* α
  2. soit aussi par au moins 1 autre caractère $\Rightarrow$ *le nœud* $s(v)$ *existe déjà à la fin du chemin* α

Propriétés des liens suffixes 2/2

Corollaire 1 (Lemme 1)

Dans l'algorithme d'Ukkonen, chaque nouveau nœud interne créé aura un lien suffixe au plus tard à la fin de la prochaine extension

Preuve par induction :

- Base : Vrai pour $\mathcal{I}_1$ car pas de nœud interne
- Induction : supposons la propriété vraie jusqu'à la fin de la phase i et considérons la phase $i+1$
  - Lemme 1 $\Rightarrow$ si un nœud v est créé, $s(v)$ existe déjà ou sera créé à l'extension suivante
  - Pas de nouveau nœud interne créé à la dernière extension
- Donc tous les liens suffixes des nœuds internes créés lors de la phase $i+1$ sont connus à la fin de la phase $i+1$ et $\mathcal{I}_{i+1}$ a tous ses liens suffixes

Corollaire 1*bis* (Lemme 1)

Dans tout arbre des suffixes implicite $\mathcal{I}_i$, si un nœud interne v a une étiquette-chemin $x\alpha$, alors il existe un nœud $s(v)$ dans $\mathcal{I}_i$ d'étiquette-chemin α

Construction de $\mathcal{I}_{i+1}$ en suivant les liens suffixes 1/2

- La phase $i+1$ cherche les suffixes $S[j..i]$ dans l'ext. j, pour j de 1 à $i+1$
- Considérons les 2 premières extensions :

→ **j = 1**. La fin de la chaîne $S[1..i]$ doit finir sur une feuille de $\mathcal{I}_i$ car c'est la plus longue chaîne de cet arbre. Idée : garder un pointeur sur la feuille correspondant à la chaîne complète courante $S[1..i]$. Son extension est toujours faite par la Règle 1, donc temps constant

→ **j = 2**. Soit $S[1..i] = x\alpha$ et $(v, 1)$ l'arc entrant sur la feuille 1. Il faut trouver $S[2..i] = \alpha$. v est soit :

- la racine, alors on suit naïvement α depuis la racine
- un nœud interne, alors v a un lien suffixe (Corollaire 1*bis*) et puisque $s(v)$ a pour étiquette un préfixe de α, on peut commencer la recherche à ce nœud Soit γ l'étiquette de l'arc $(v, 1)$. Pour trouver la fin de α :
  1. Remonter de la feuille 1 au nœud v
  2. Suivre le lien suffixe de v à $s(v)$
  3. Suivre le chemin γ depuis $s(v)$, la fin du chemin est la fin de α
  4. Mettre à jour l'arbre selon les règles d'extension

Construction de $\mathcal{I}_{i+1}$ en suivant les liens suffixes 2/2

- Pour étendre n'importe quelle chaîne de $S[j..i]$ à $S[j..i+1]$ pour $j > 2$, suivre la même idée générale :

  1. Commencer à la fin de la chaîne $S[j-1..i]$ de l'arbre courant
  2. Remonter d'au plus un nœud jusqu'à la racine ou un certain nœud v (soit γ l'étiquette de cet arc)
  3. Suivre le lien suffixe de v à $s(v)$ (si v n'est pas la racine)
  4. Suivre le chemin γ depuis $s(v)$, la fin du chemin est la fin de $S[j..i]$
  5. Mettre à jour l'arbre selon les règles d'extension

- Particularité : la fin de $S[j-1..i]$ peut être un nœud avec un lien suffixe, alors on le suit directement

- Notez qu'on ne remonte donc jamais de plus d'un arc

Algorithme : Single Extension Algorithm (SEA)

`/* 1 "remontée" */`

Trouver le premier nœud v *à* ou *au-dessus* de la fin de $S[j-1..i]$ qui est soit la racine, soit a un lien suffixe (soit γ la chaîne entre v et la fin de $S[j-1..i]$) ;

`/* 2 "marche descendante" */`

si *(v n'est pas la racine)* **alors**
    Traverser le lien suffixe jusqu'à $s(v)$ puis suivre le chemin γ ;
sinon
    Suivre le chemin $S[j..i]$ depuis la racine ;

`/* 3 "extension" */`

Règles d'extensions pour s'assurer que $S[j..i]S[i+1]$ est dans l'arbre ;

`/* 4 "mise à jour des lien suffixe" */`

si *(un nœud interne w a été créé à l'extension* $j-1$*)* **alors**
    Créer un lien suffixe $(w, s(w))$;

Première astuce : “skip/count trick”

- Suivre le chemin γ se fait en temps $O(|\gamma|)$ naïvement
- Cette astuce nous permet de réduire ce temps aux nombres de nœuds sur le chemin $\Rightarrow$ toutes les marches descendantes d'une même phase prendront au plus $O(n)$

Astuce 1

Soit $g = |\gamma|$, $h = 1$ et g' le nb de caractères sur l'arc sortant de $s(v)$ dont la 1^{re} lettre est $\gamma[h]$

Tant que $(g > g')$ on “skippe” (saute) le nœud à l'extrémité de l'arc courant :
$g = g - g'$, $h = h + g'$, arc courant devient celui commençant par $\gamma[h]$

Quand $(g \leqslant g')$ l'algo saute au caractère g sur l'arc courant et s'arrête

Profondeurs de nœud

Définition

La *profondeur de nœud* d'un nœud v, $p_n(v)$, est le nombre de nœuds de la racine jusqu'à v

Définition

La *profondeur de nœud courante (pnc)* de l'algorithme est la profondeur de nœud du dernier nœud visité par l'algorithme

Lemme 2

Soit $(v, s(v))$ un lien suffixe traversé pendant l'algorithme d'Ukkonen. À ce moment, la profondeur de nœud de v est au plus 1 de plus que celle de $s(v)$

- Chaque nœud interne ancêtre de v (d'étiquette-chemin $x\beta$) a un lien suffixe vers un nœud d'étiquette-chemin β
- Mais $x\beta$ est un préfixe du chemin menant jusqu'à v, donc β est un préfixe du chemin menant jusqu'à $s(v)$
- Donc chaque lien suffixe d'un ancêtre de v va vers un ancêtre de $s(v)$
- De plus si $\beta \neq \varepsilon$, le nœud étiqueté β est un nœud interne
- Les profondeurs de nœud des ancêtres de v doivent être différentes, donc chaque ancêtre de v a un lien suffixe vers un ancêtre distinct de $s(v)$
- Donc $p_n(s(v))$ est au moins 1 (pour la racine) + plus le nb d'ancêtres internes de v qui ont une étiquette-chemin de plus d'un caractère
- Le seul ancêtre de v sans ancêtre correspondant chez $s(v)$ que v peut avoir est un nœud interne d'étiquette-chemin de longueur 1 (étiquette x)
- Donc v ne peut avoir une profondeur de nœud supérieure que de 1 à celle de $s(v)$, en d'autres terme $p_n(v) \in [1..p_n(s(v)) + 1]$

Illustration Lemme 2

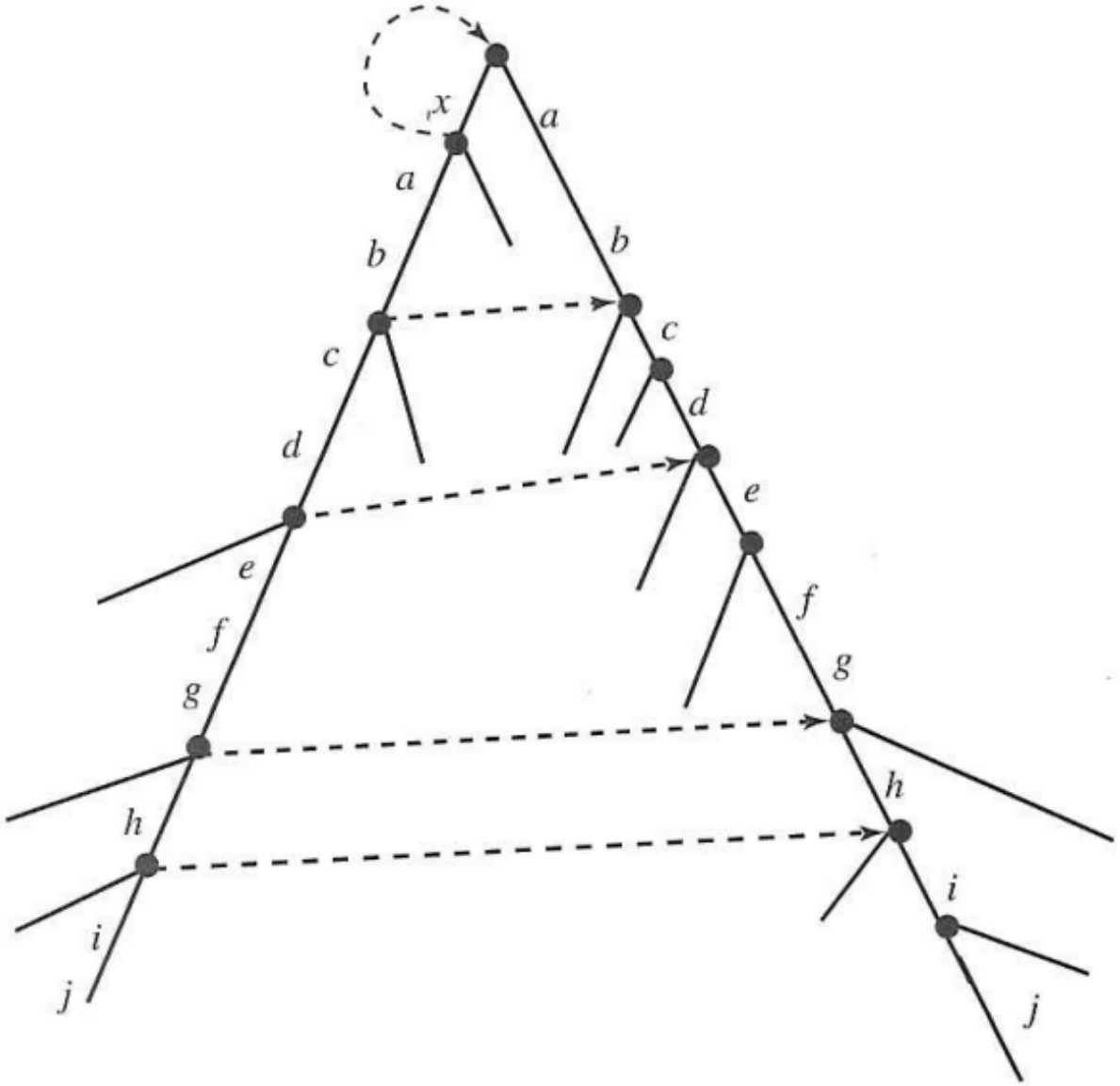


Figure 6.7: For every node v on the path $x\alpha$, the corresponding node $s(v)$ is on the path α. However, the node-depth of $s(v)$ can be one less than the node-depth of v, it can be equal, or it can be greater. For

Retour sur la complexité

Théorème 1

En utilisant l'Astuce 1, chaque phase de l'Algorithme d'Ukkonen est en $O(n)$

- Chaque phase i a $i+1 \leqslant n$ extensions
- On a déjà vu qu'à chaque extension, seules les marches descendantes ne sont pas en temps constant
- La remontée fait décroître la *pnc* d'au plus 1, tout comme la traversée du lien suffixe (Lemme 2)
- Chaque arc traversé dans la marche descendante incrémente cette *pnc*
- Donc sur une phase entière, la *pnc* est décrémentée au plus $2n$ fois, et comme aucun nœud ne peut avoir une profondeur de nœud supérieure à n, l'incrémentation totale de la *pnc* est bornée par $3n$
- Avec l'Astuce 1, chaque phase peut être exécutée en $O(n)$

Retour sur la complexité

Corollaire 3 (théorème 1)

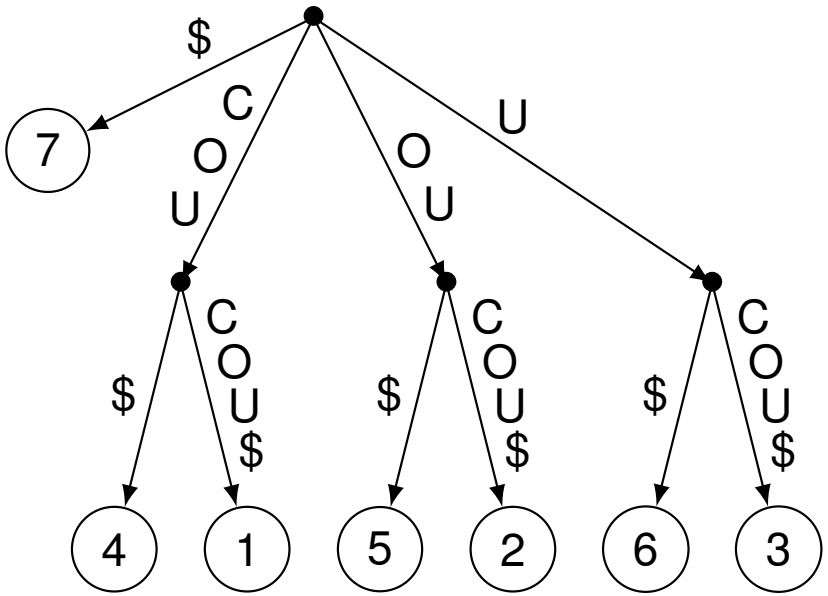
L'algorithme d'Ukkonen peut être implémenté en $O(n^2)$

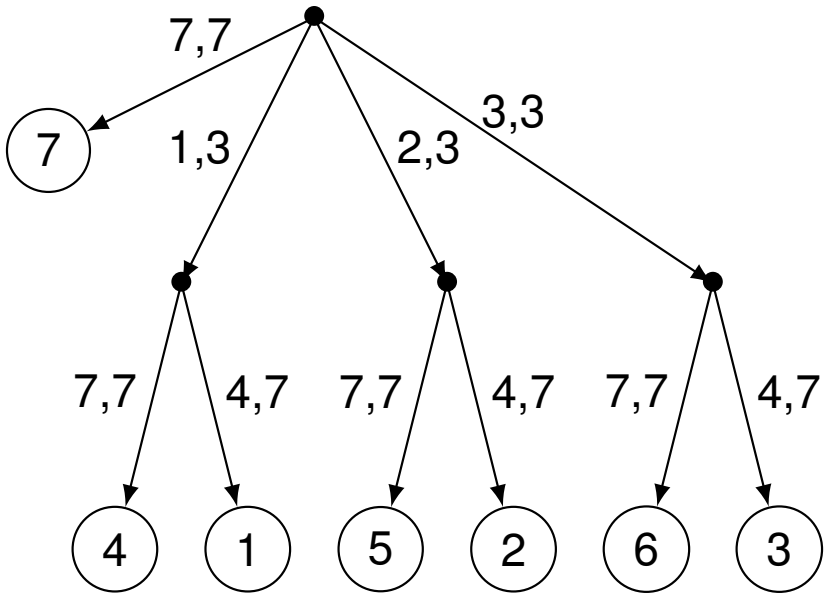
- L'algorithme a n phases et chaque phase est bornée par $O(n)$
- Analyse plus fine à effectuer en ne considérant pas les phases de manières séparées
- Jusqu'ici pas de progrès dans la complexité ☹
- Mais des progrès conceptuels ☺
- qui vont nous permettre d'atteindre la complexité attendue de $O(n)$ avec un petit détail d'implémentation et deux astuces de plus

Plan du cours

Compression des étiquettes des arcs

- L'arbre des suffixe peut occuper un espace $O(n^2)$ (ex : $abcd \dots z$)
- Au lieu d'écrire explicitement les sous-chaîne sur les arcs : les remplacer par deux indices
- Nombre d'arcs au plus $2m - 1$, 2 entiers par arcs : arbre encodé en $O(n)$





Plan du cours

La Règle 3 est une règle d'arrêt

- Dans n'importe quelle phase $i+1$, si la Règle 3 s'applique dans une extension j, c'est que le caractère $S[i+1]$ est à la suite du chemin $S[j..i]$
- Elle s'appliquera alors à toutes les extensions suivantes, de $j+1$ à $i+1$
- Quand la Règle 3 s'applique : rien à faire (même pas de liens suffixes à gérer)

Astuce 2

Finir chaque phase $i+1$ dès que la Règle 3 s'applique

- Les extensions terminées après la première application de la Règle 3, sont dit terminées *implicitement*, les autres le sont *explicitement*

« *Once a leaf, always a leaf* »

- Si une feuille étiquetée j est créée à un moment de l'algorithme, alors elle restera feuille dans tous les STI créés ultérieurement
  Et c'est la Règle 1 qui s'appliquera à toutes les extensions j des phases suivantes

- Les premières extensions de n'importe quelle phase i sont étendues par les Règles 1 ou 2, notons j_i la dernière de ces extensions

- Comme la Règle 2 crée une feuille : $j_i \leqslant j_{i+1}$, c'est-à-dire que le nombre d'applications des Règles 1 et 2 en début de phase ne diminue pas

$\rightarrow$ Mais on pourrait alors éviter dans la phase $i+1$ de faire explicitement toutes les extensions de 1 à j_1

Actions implicites en début de phase

Astuce 3

Dans la phase $i+1$, lorsqu'une feuille est créée, au lieu d'étiqueter son arc $p, i+1$, écrire p, e où e désigne la fin courante.
e est une variable globale, mise à $i+1$ à chaque phase (une seule fois/phase).

- Dans une phase $i+1$, la Règle 1 s'applique à toutes les extensions de 1 à j_1 au moins
  $\Rightarrow$ pas de travail explicite à faire pour les j_i premières, sauf incrémenter e et traiter les extensions suivantes

Traitement d'une phase

- Finalement pour une phase $i+1$, en utilisant les astuces 2 et 3, il ne reste plus qu'à traiter explicitement les extensions à partir de j_i+1 à et jusqu'à la première application de la Règle 3 (ou la fin de la phase)
  Traitement explicite par l'algorithme SEA vu précédemment

Algorithme : Single Phase Algorithm (SPA)

`/* 1 "extensions implicites par la Règle 1" */`

$e := i+1$;

`/* 2 "traitement explicite des phases" */`

pour *j de* j_i+1 *à* ($j^* = i+1$ *ou* 1^{re} *appli Règle 3)* **faire**
  Traiter la phase j avec l'algorithme SEA ;

`/* 3 "mise à jour" */`

$j_{i+1} := j^* - 1$;

Algorithme SEA

- La phase $i+2$ commencera par calculer explicitement l'extension j^*, qui est la dernière extension explicite calculée pour la phase $i+1$
  $\Rightarrow$ 2 phases successives partagent au plus un indice, j^*, pour lequel une extension explicite est calculée !

- De plus, la phase $i+1$ termine en connaissant la fin de $S[j^*..i+1]$, donc l'extension j^* de la phase $i+2$ peut se faire sans déplacement
  $\Rightarrow$ La première extension de chaque phase se fait en temps constant !

Théorème 2

En utilisant les liens suffixes et en implémentant les Astuces 1, 2 et 3, l'Algorithme d'Ukkonen construit les arbres des suffixes implicites de $\mathcal{I}_1$ à $\mathcal{I}_n$ en temps $O(n)$

Preuve théorème 2

- Le temps pour toutes les extensions implicites dans chaque phase est constant, donc elles comptent pour $O(n)$ à travers tout l'algo

- Soit $\bar{j}$ l'extension courante exécutée par l'algo : $\bar{j}$ ne décroît jamais et est borné par $m \Rightarrow$ l'algo exécute au maximum $2n$ extensions explicites

- Le temps d'exécution d'une extension explicite est un temps constant + une partie proportionnelle au nombre de nœuds "sautés"
  Comme la profondeur de nœud ne change pas entre deux phases (on reste sur l'extension j^*), on peut appliquer la même idée que pour la preuve du Th. 1 : le nombre maximum de nœuds traversés durant tout l'algo est bornée par $O(n)$

Plan du cours

Obtenir le vrai arbre de suffixes pour S

Le dernier arbre des suffixes implicite, $\mathcal{I}_n$, peut être transformé en vrai arbre des suffixes en temps $O(n)$:

- Ajouter le caractère \$ à la fin de S et laisser l'algorithme d'Ukkonen continuer avec ce caractère
- Cela a pour effet que plus aucun suffixe n'est préfixe d'un autre, l'arbre de suffixe implicite construit possède alors une feuille par suffixe
- Remplacer la variable e sur chaque arc entrant sur une feuille par n (on peut l'effectuer en faisant un parcours en profondeur en $O(n)$)

Théorème 3

L'Algorithme d'Ukkonen construit le vrai arbre des suffixes en temps $O(n)$

Plan du cours

Références

Toute cette présentation est basée sur le chapitre 6 du livre suivant :

[Gusfield, 97] Dan Gusfield, **Algorithms on Strings, Trees and Sequences** - Computer Science and Computational Biology, University of California, Davis. ISBN :9780521585194. Août 1997. *En anglais*