Суфиксни дървета част 1

11.12.2020 г.

(Алгоритъм на Уконен (Ukkonen))

Връзката между суфиксното дърво от суфиксния автомат от алгоритъма на Блумер и суфиксното дърво от алгоритъма на Уконен е много тясна. Въпреки това алгоритъма на Блумер е публикуван около 10 години по-рано от този на Уконен. Идеята тук отново е да се поддържа линейно множество от всички инфикси на дадена дума. Алгоритъма е *on-line* и при обработка на следваща буква имаме дървото с точност до финални състояния. Основната полза от суфиксните дървета през 90-те години е била за биолозите и биоинформатиците, които са изследвали геномните последователности и за тях е било важно да имат ефективно структурно представяне, което да работи върху големи данни от символи. С времето това отстъпва място на суфиксните масиви, след като хората са се научили да ги строят директно (около 5-6 години след алгоритъма на Уконен), но има една друга полза от алгоритъма на Уконен и това, че той е on-line, която е свързана с още един резултат на Lempel-Ziv от края на 70-те години. Техния алгоритъм все още се използва за компресия на текстове и хубавото на този алгоритъм на Уконен (имащ нетривиален анализ) е, че показва, че алгоритъма на Lempel-Ziv постига възможно най-доброто - постига ентропията на ниво символ, независимо от разглеждания контекст. Това е абсолютната мярка на отчитане на закономерностите в този текст.

Този алгоритъм на Lempel-Ziv прави грубо казано следното: Той кодира поредния фрагмент от текста с парче от текста, което се е срещало по-рано и точно му пасва и това го прави по следната алчна схема - избира най-дългото парче, което пасва на текщия суфикс и не може да бъде продължено (т.е. следващата буква вече не се среща в текста след тои контекст, т.е. тя в някакъв смисъл води до нашата изненада, тъй като не следва регулярностите, които сме наблюдавали по-рано в текста). Поради тази причина този алгоритъм води до хубавата мярка за компресия, която постига ентропия.

Алгоритъма на Уконен и суфиксните дървета дават достъп точно до тези най-дълги парзета в текста, които ние обработваме в момента и които може би ще продължат със следващия символ да се срещат в нашия текст, а може би ние ще бъдем изненадани и този символ ще се срещне за първи път в този ляв контекст и ние съответно ще трябва да го кодираме експлицитно.

С други думи, on-line алгоритъма на Уконен в комбинация с този на Lempel-Ziv дава възможност за on-line компресия на текст, която постига теоритичния оптимум за смачкване на текста.

<u>Дадено</u>: азбука Σ, и текст/дума $w \in \Sigma^*$.

<u>Търси се</u>: Суфиксно дърво за w. Интуитивно: това е дърво, в което листата съответстват на суфиксите на w, а вътрешните върхове съответстват на разклонения (т.е. инфикси, които се срещат в различни десни контексти)

<u>Цел</u>: On-line *линеен* алгоритъм за построяване на такова суфиксно дърво.

(Трябва да се внимава когато употребяваме думата линеен, тъй като както тук, така и при суфиксния автомат, тя се употребява с точност до някакъв фактор, който зависи от азбуката Σ и представянето на съответните преходи)

<u>Дефиниция</u>: Суфиксен $Trie$ за $w \in \Sigma^*$ е просто $Trie$ за крайното множество от думите, които са суфикси на w, т.е. за $\mathcal{D} = Suff(w)$.

$$Trie(\mathcal{D}) = <\Sigma, Pref(\mathcal{D}), \varepsilon, \delta_{\mathcal{D}}, \mathcal{D}>$$
$$\delta_{\mathcal{D}}(u, a) = ua \Leftrightarrow ua \in Pref(\mathcal{D}).$$

Проблема на този $Trie$ е, че макар и с ограничен брой на върховете отгоре - със сумата от дължинитие на всички думи в $\mathcal{D}$, то броя на символите (върховете) в $Suff(\mathcal{D})$ може да бъде пропорционален на квадрата на дължината на w. Поради тази причина, решението да построим $Trie$ за суфиксите на w не е оптимално.

Да обърнем внимание, че префиксите на суфиксите на една дума не са нищо друго освен инфикси на една дума: $Pref(Suff(\mathscr{D})) = Inf(\mathscr{D})$.

<u>Дефиниция</u>: Суфиксно дърво за една дума $w \in \Sigma^*$ е:

$$S_w = < \underbrace{V_w}_{\text{върхове}}, \underbrace{\tilde{\delta}_w}_{\text{родителска функция}}, \underbrace{\varepsilon}_{\text{корен}} >$$

$$V_w = \{\varepsilon\} \cup \left\{v \in Suff(w) \,|\, v \text{ е листо в } Trie\big(Suff(w)\big)\right\} \cup \left\{v \in Inf(w) \,|\, \exists a \neq b \ (a, b \in \Sigma \ \& \ va \text{ и } vb \in Inf(w)\right\}$$

Грубо казано това е окастрен $Trie(Suff(\mathscr{D}))$. Второто множество от върхове са листата от $Trie(Suff(\mathscr{D}))$, а третото множество от върхове са тази върхове от $Trie(Suff(\mathscr{D}))$, които се разклоняват.

Смисъла на тази функция на бащите $\tilde{\delta}_w$ ще бъде да ни каже къде отиват тези вътрешни върхове със съответната буква.

$\tilde{\delta}_w : V_w \times \Sigma \nrightarrow V_w$ (частична функция), така че $\tilde{\delta}_w(v, a) = \overrightarrow{va}^{w}$ е най-късия инфикс на w, който:

1. започва с va
2. е елемент на множеството V_w

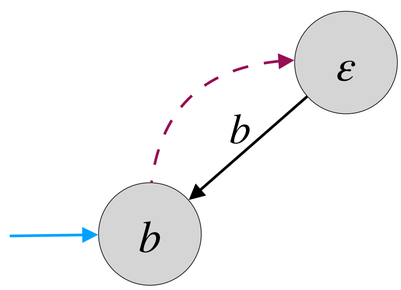
<u>Забележка</u>: Да обърнем внимание на дуалността, която ср получава между алгоритма на Блумер и този на Уконен. Единствената разлика между това да строим суфиксно дърво за обърнатата дума w^R и суфиксен автомат за w е тази, че в алгоритъма на Уконен не участват всички суфикси (префикси за w^R), така както всички префикси участват в алгоритъма на Блумер, а само тези, които са листа на $Trie(Suff(\mathscr{D}))$.

Пример: $w = babaaaa$

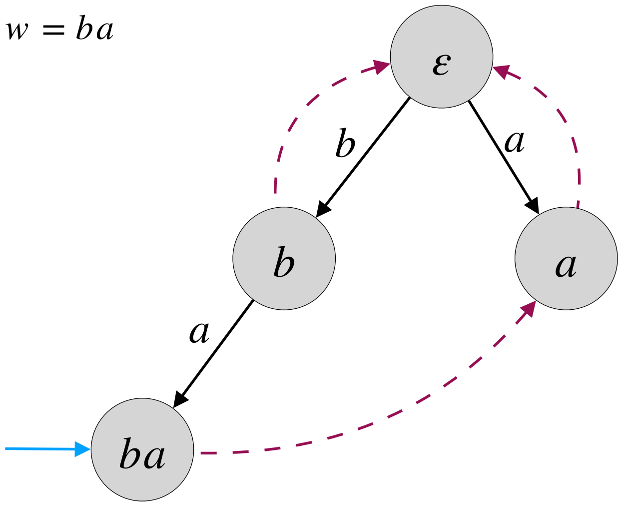
$$\begin{array}{r} babaaaa \\ abaaaa \\ baaaa \\ aaaa \\ aaa \\ aa \\ a \\ \varepsilon \end{array}$$

Суфиксен $Trie$ (on-line). $w = \emptyset$.

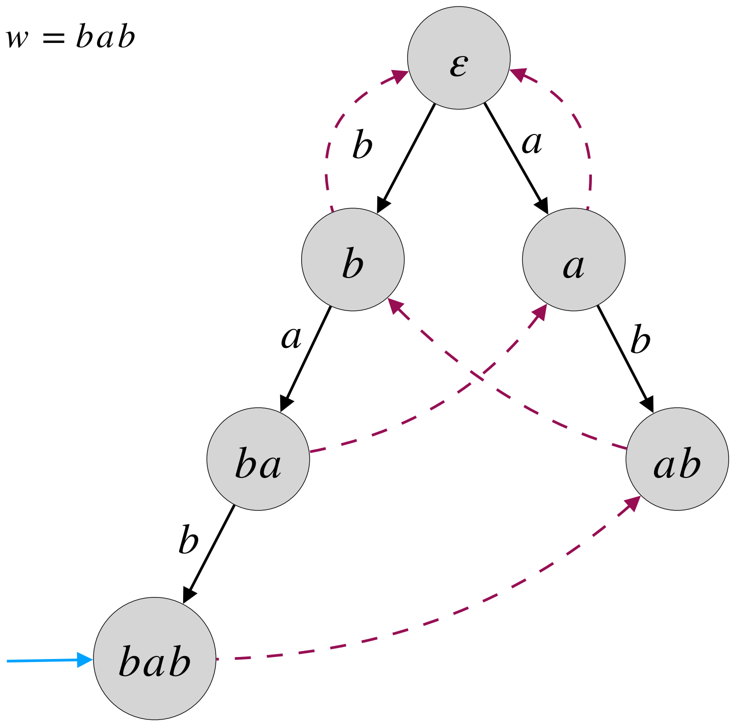
$w = b$



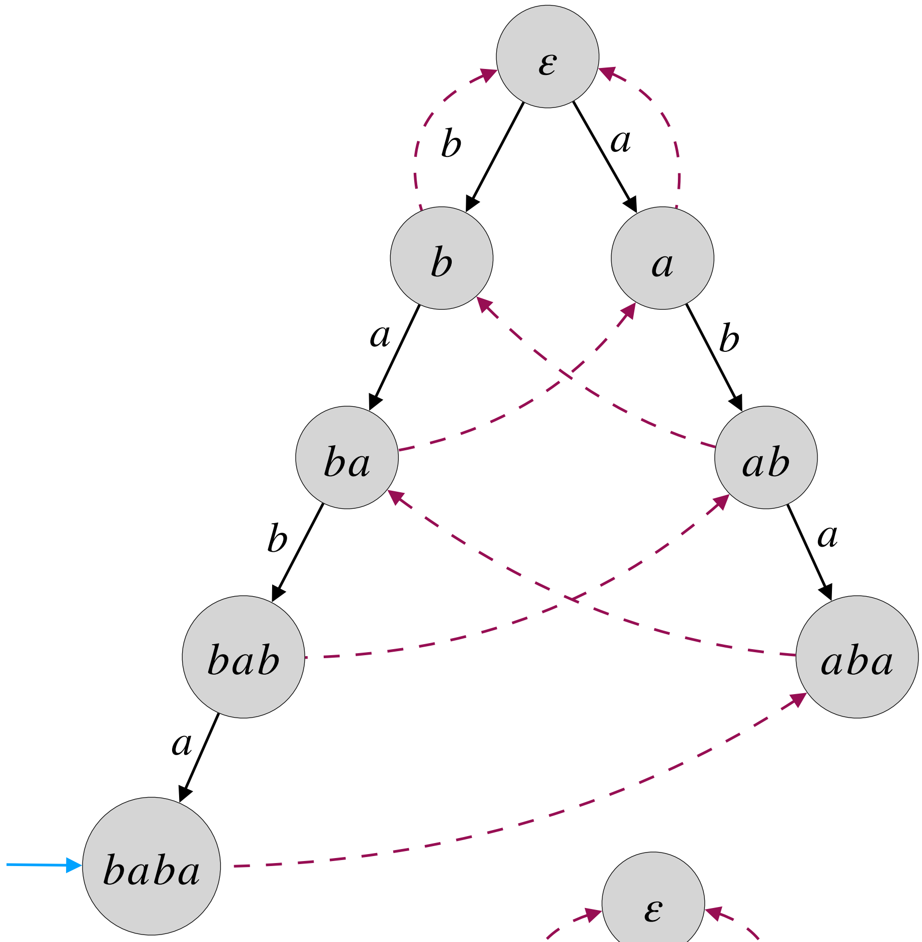
$w = ba$



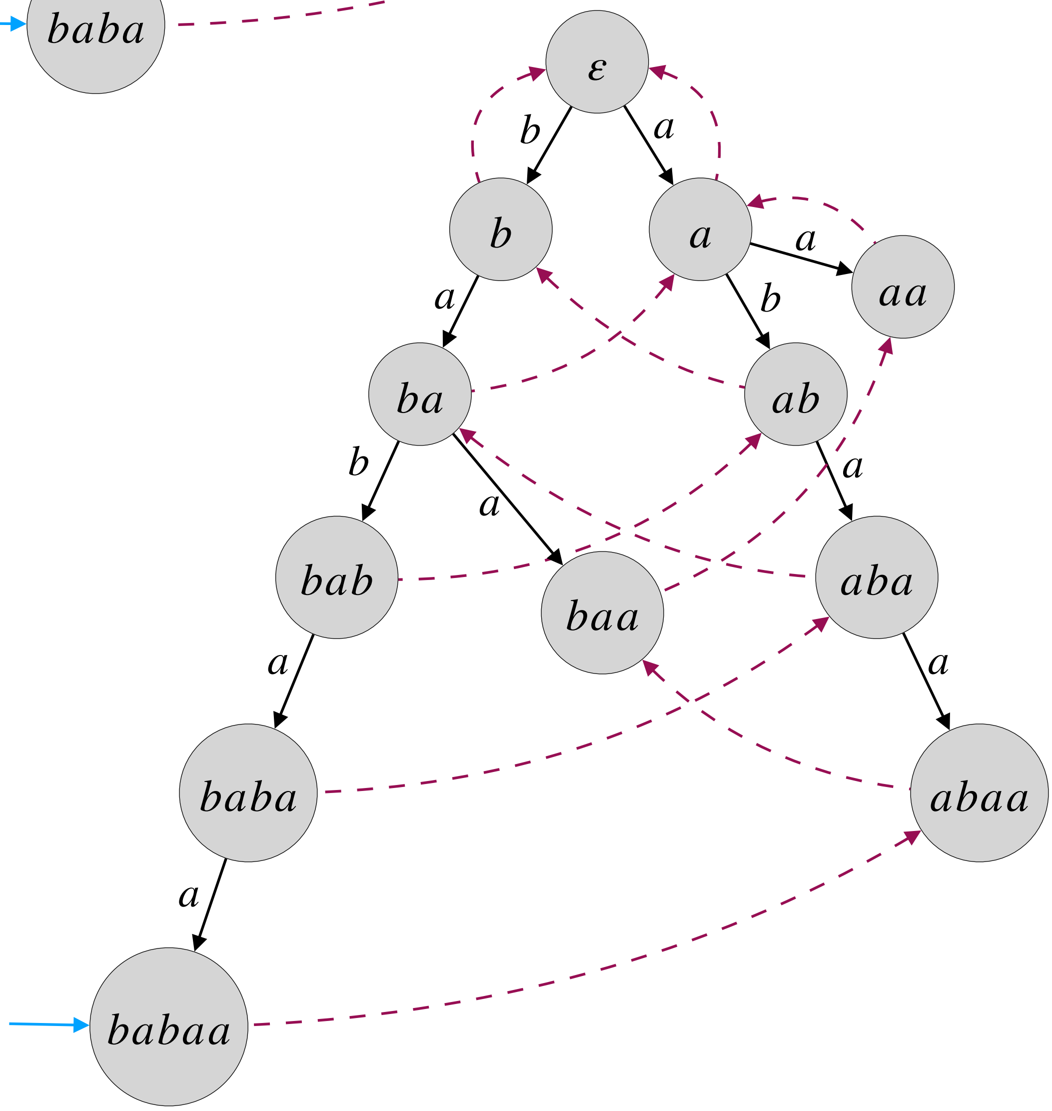
$w = bab$



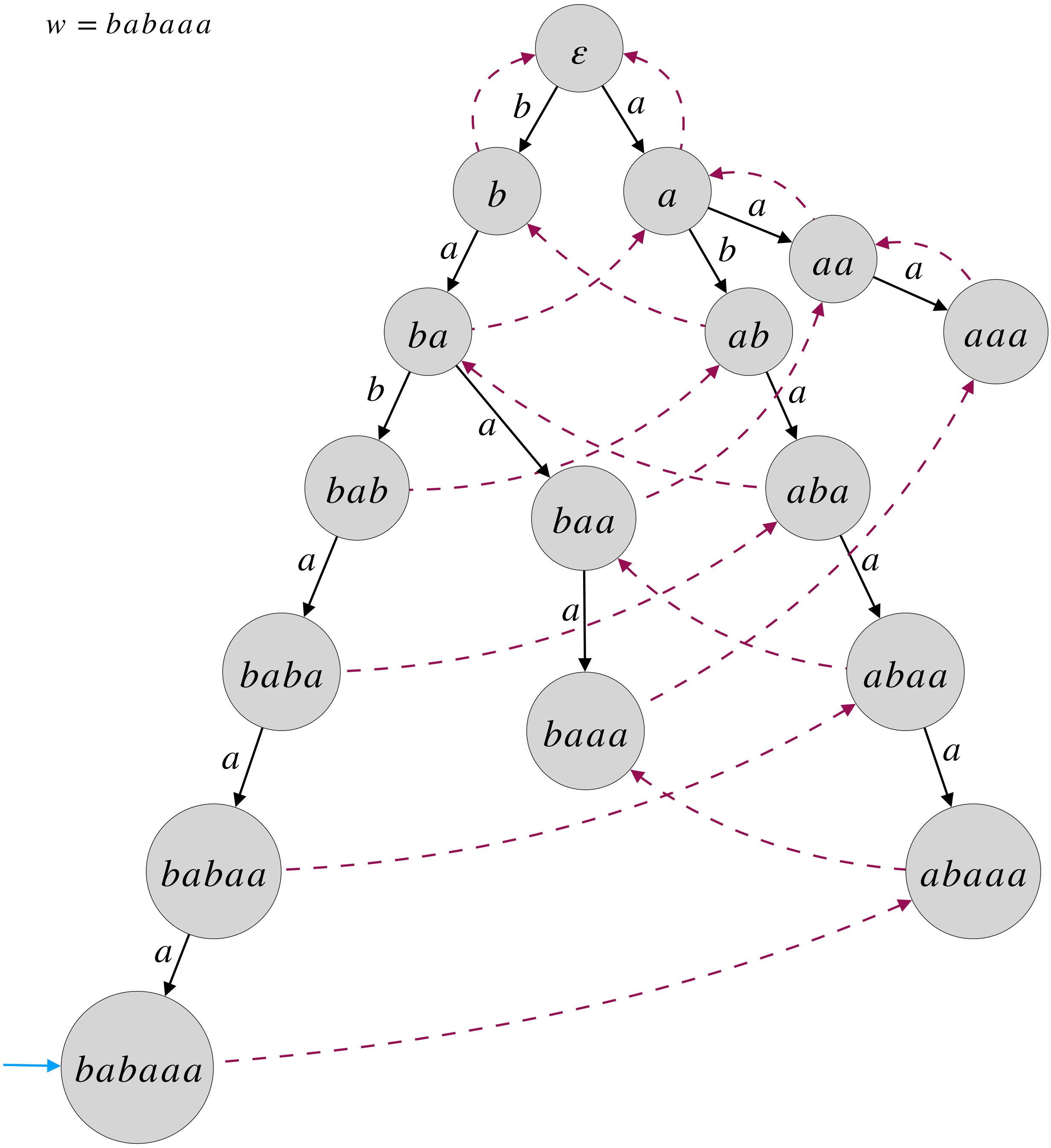
$w = baba$



$w = babaa$

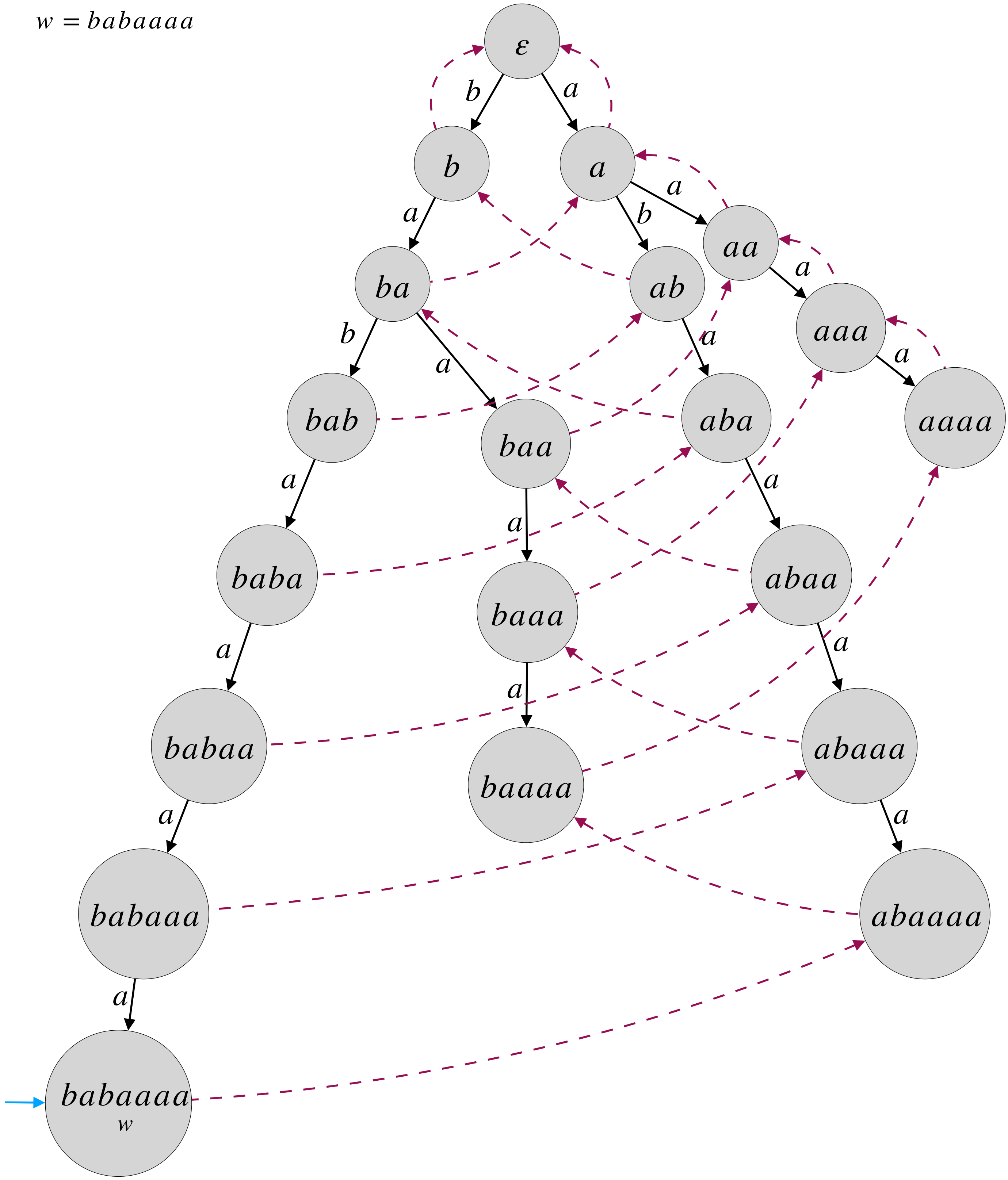


$w = babaaa$



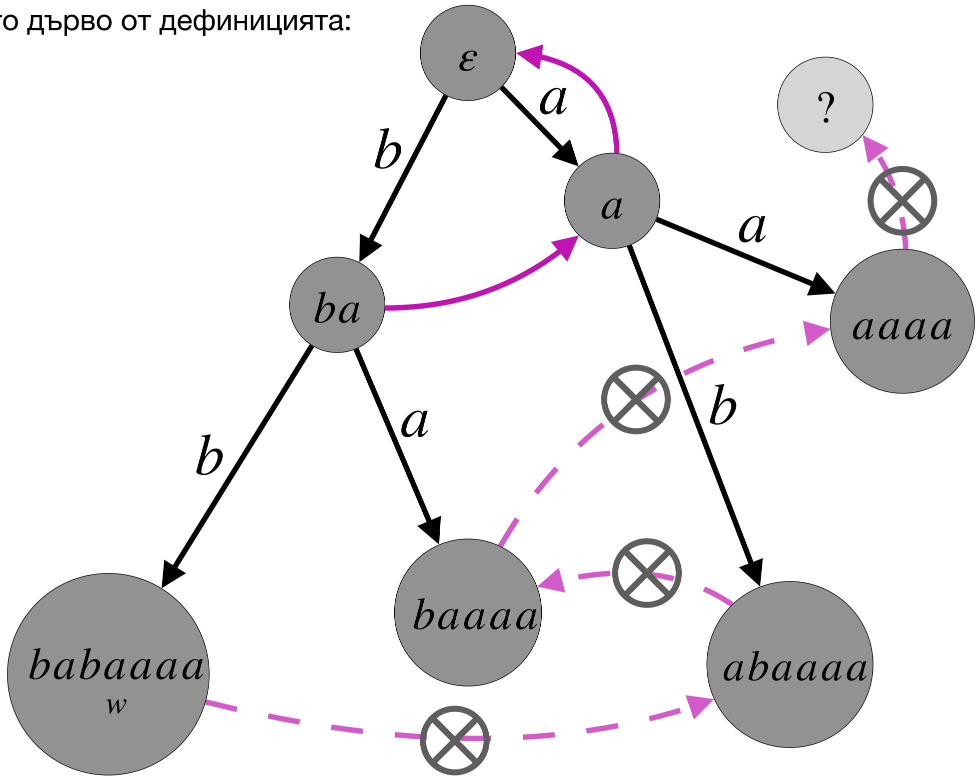
ε
b
a
b
a
a
aa
ba
b
ab
a
aaa
b
a
a
bab
baa
aba
a
a
a
baba
baaa
abaa
a
a
babaa
abaaa
a
babaaa

$w = babaaaa$



ε
b
a
ba
ab
aa
bab
baa
aba
aaa
aaaa
baba
baaa
abaa
babaa
baaaa
abaaa
babaaa
abaaaa
babaaaa
w

Суфиксното дърво от дефиницията:



Вътрешните върхове в суфиксното дърво са тези върхове от суфиксния $Trie$, които имат поне две деца, а листата на суфиксното дърво са тези върхове от суфиксния $Trie$, които нямат деца.

<u>Дефиниция</u>: $s : Inf(w) \longrightarrow Inf(w)$; $s(av) = v$, за $av \in Inf(w)$, $a \in \Sigma$.
<u>Свойство</u>: Нека $w \in V_w$ и $v \neq \varepsilon$ и v НЕ е листо. Тогава $s(v) \in V_w$.

<u>Доказателство</u>: От условието $\Rightarrow \exists b, c\ (b \neq c;\ b, c \in \Sigma)$, за които $vb,\ vc \in Inf(w)$, освен това $v \neq \varepsilon$ и следователно $v = av'$, за някоя буква $a \in \Sigma$. По дефиниция $s(v) = v'$. Тъй като vb и vc са инфикси на w, то $v'b$ и $v'c$ също са инфикси на $w \Rightarrow v' \in V_w$.

<u>Свойство</u>: $|V_w| \leq 2|w| + 1$
<u>Доказателство</u>:

$$V_w^{\geq 2} = \{u \in Inf(w) \,|\, \exists a \neq b\ (a, b \in \Sigma\ \&\ ua, ub \in Inf(w)\}$$

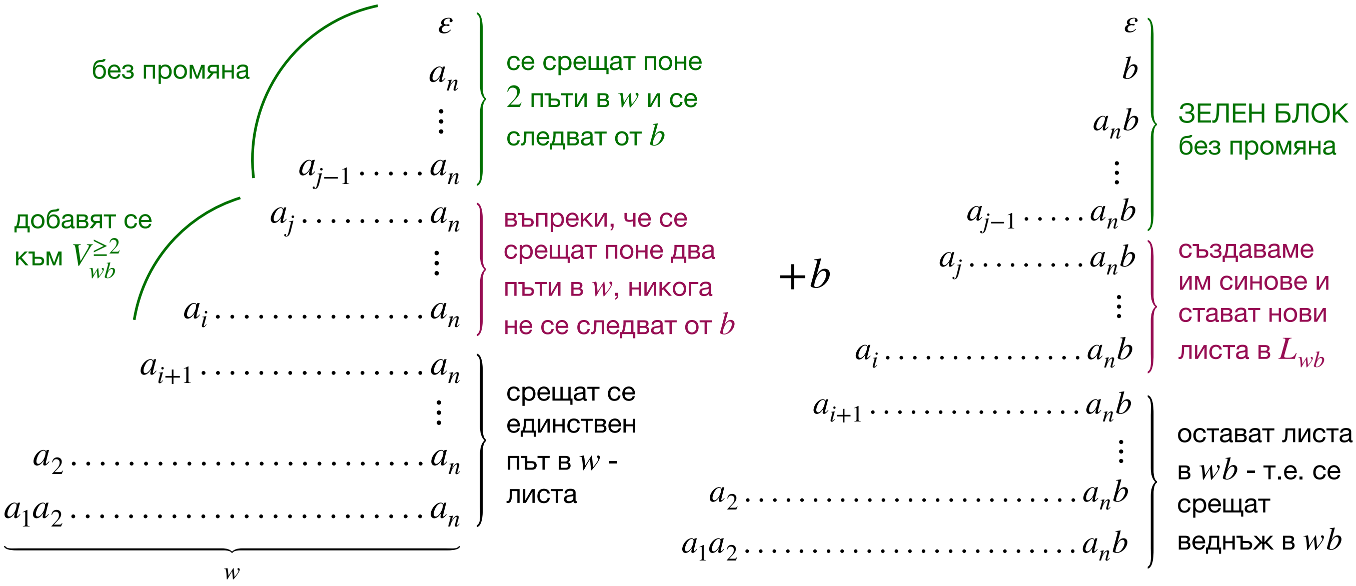
$L_w = \{u \in Suff(w) \,|\, u$ е листо и не е празната дума $\varepsilon\}$

$|V_w| = |L_w \cup \{\varepsilon\}| + |V_w^{\geq 2}| \leq |Suff(w)| + |V_w^{\geq 2}| = |w| + 1 + |V_w^{\geq 2}|$, но
$|V_w| - 1 = \underbrace{|E_w|}_{edges} \geq 2|V_w^{\geq 2}|$ и следователно

$|V_w| \leq |w| + 1 + |V_w^{\geq 2}| \leq |w| + 1 + \dfrac{|V_w| - 1}{2}$ и следователно $\underbrace{|V_w| \leq 2|w| + 1}$.

<u>Построяване</u>: ще поддържаме V_W, $\tilde{\delta}_w$, s_w за $V_w^{\geq 2} \setminus \{\varepsilon\}$

<u>Означение</u>: L_W - листата в S_w.



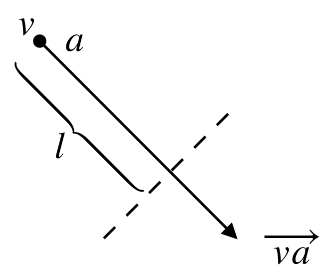
Черния блок винаги ще го има, но един от останалите два блока (розовия и зеления) може да бъде изроден (т.е. да не съществува). В случая когато не съществува розовия блок, ще имаме дума, която се среща поне два пъти и се следва от b. Това обаче не означава, че тази дума ще бъде връх в V_w и съответно не е ясно трябва ли да бъде добавена.

От друга страна, ако зеления блок е изроден (включително и ε не се следва от b), тогава всичко нагоре от черния блок ще е розов и ще трябва да свършим работата по целия розов блок. На следващата итерация, това което ще се случи е, че новото разпределение на розов и зелен блок ще бъде само в областта от зеления блок в предишната конфигурация. Поради тази причина, работата която ще се върши е само в розовите блокове и може да съобразим, че броя на елементите в зеления блок ще се увеличи най-много с един. Тоест заленените думи, които се добавят са най-много една във всяка итерация и съответно розовите блокове, върху които ще трябва да работим - общо ще бъдат най-много n на брой. Т.е. общата работа която трябва да свършим ще бъде свързана с тези розови думи, а те в крайна сметка ще бъдат толкова, колкото е дължината на цялата дума w.

Това е в основата на алгоритъма на Уконен.

Представяне на инфикси от w: инфикс $\alpha \longmapsto (v, a, l)$ - наредена тройка, където

1. $\alpha = v$(е експлицитен връх), $a = \perp$, $l = 0$
2. $\alpha \in Pref\big(\delta_w(v,a)\big) \setminus \{\overrightarrow{va}\}$, $l = |\alpha| - |v|$



Вярно ли е, че b може да следва α?

В 1.сл. отговора на въпроса е ясен - просто поглеждаме $\tilde{\delta}(v, b)$, но във втория случай отговора не е тривиален.

1. Имаме масив за w: $w[0...n-1]$ (тъй като търсим онлайн алгоритъм , нека масива е динамичен)

2. $V_w^{\geq 2}$: $\begin{cases} \textbf{int } start \text{ // } v = w[start \dots start + len - 1] \\ \textbf{int } len \\ list\ \tilde{\delta}_w(v) \\ \textbf{int } s_w(v) \text{ - суфикс линка} \end{cases}$

3. L_w: $\textbf{int}$ $start$ (могат да бъдат представени само с началната позиция)

Връщаме се на нетривиалния въпрос:

$\textbf{procedure bool } followedBy(\underset{\text{състояние}}{v}, \underset{\text{буква}}{a}, \underset{\text{дължина}}{l}, \underset{\text{буква}}{b})\ \{$

// $\alpha b \overset{?}{\in} Inf(w)$, където $\alpha = (v, a, l)$

if $l = 0$ **then**

  return $\tilde{\delta}(v)(b)$ $is\ defined$

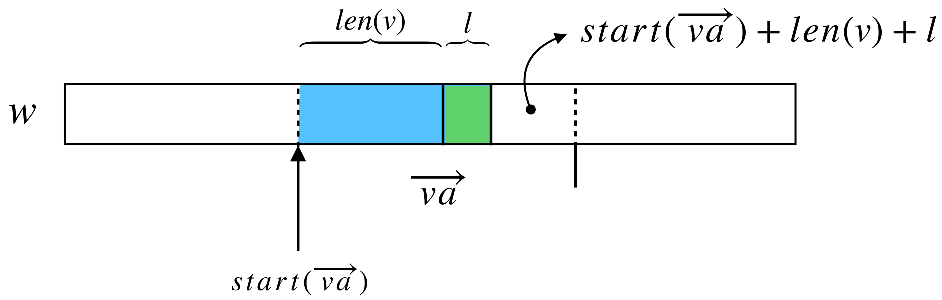
else

  $v' \leftarrow \tilde{\delta}(v)(a)$

  $pos \leftarrow start(v') + len(v) + l$

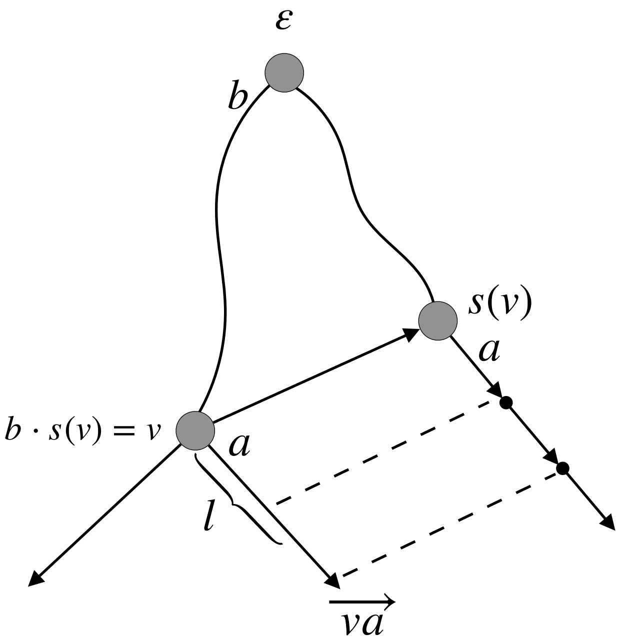
  return $w[pos] = b$

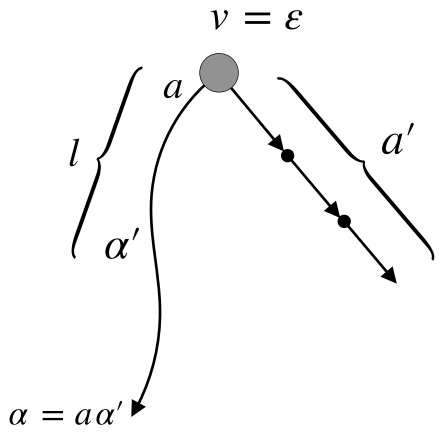
$\}$



$\alpha \longmapsto (v, a, l)$

Търсим представянето на $s(\alpha)$, $s(\alpha) \cancel{\longmapsto (s(v), a, l)}$





$l = |\alpha| - |v|$

procedure $followInfix(\underset{V_w^{\geq 2}}{\underset{\uparrow}{v}}, start, l)$ {

// Връщаме представяне на инфикса $v \cdot w[start \dots start + l - 1]$

if $l = 0$ **then**

return $(v, \perp, 0)$

$v' \leftarrow \tilde{\delta}(v)(w[start])$

$l' \leftarrow len(v') - len(v)$

if $l' > l$ **then**

return $(v, w[start], l)$

else

// $l \geq l'$

return $followInfix(v', start + l', l - l')$

}

Ако резултатът е $\alpha = (\bar{v}, a, \bar{l})$, то времето за извършване на гореописаната рекурсия е най-много $O(l - \bar{l})$

procedure $findSuffixLink(v, a, l)$ {

// $\alpha = (v, a, l)$, $\alpha \in Inf(w)$

// Търсим $s(\alpha)$ (представяне)

if $v = \varepsilon$ **then**

if $l = 0$ **then** \\ $\alpha = \varepsilon$ и $s(\alpha)$ не е дефинирано

return $\perp$

else

$v' \leftarrow \tilde{\delta}(v)(a)$

$start \leftarrow start(v') + 1$

return $followInfix(v, start, l - 1)$

else

if $l = 0$ **then**

return $\big(s(v), \perp, 0\big)$

else

$v' \leftarrow \tilde{\delta}(v)(a)$

$start \leftarrow start(v') + len(v)$

return $followInfix\big(s(v), start, l\big)$

}