ALGORITMOS EM SEQUÊNCIAS

Yan Soares Couto

Orientadora: Cristina Gomes Fernandes

2016

Instituto de Matemática e Estatística

DEFINIÇÃO

String $S[1..|S|]$: vetor em que cada elemento é de um alfabeto Σ finito.

Em geral, $\Sigma = \{a, b, \ldots, z\}$.

DEFINIÇÃO

String $S[1..|S|]$: vetor em que cada elemento é de um alfabeto Σ finito.

Em geral, $\Sigma = \{a, b, \ldots, z\}$.

"abracadabra"

Substring de S: subvetor de S, por exemplo, "cadab".

PALÍNDROMOS

PALÍNDROMOS

Palíndromo: string que coincide com seu reflexo (de trás pra frente)

"reviver"

Problema: Qual a maior substring de S que é um palíndromo?

PALÍNDROMOS

Palíndromo: string que coincide com seu reflexo (de trás pra frente)

"reviver"

Problema: Qual a maior substring de S que é um palíndromo?

Solução trivial: Para cada posição i de S, determinar o maior palíndromo par e ímpar com centro i

Complexidade: $\mathcal{O}(|S|^2)$.

PALÍNDROMOS ÍMPARES

Consideremos apenas palíndromos de tamanho ímpar.

Centro de palíndromo (ímpar): posição do meio

Ex: “reviver” tem centro na letra i.

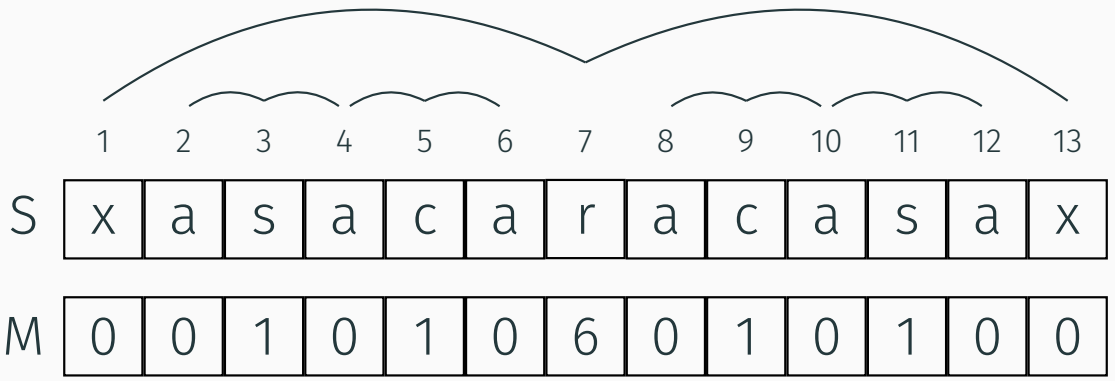
PALÍNDROMOS ÍMPARES

Consideremos apenas palíndromos de tamanho ímpar.

Centro de palíndromo (ímpar): posição do meio

Ex: “reviver” tem centro na letra i.

Seja M[i] é o maior inteiro tal que S[i − M[i] . . i + M[i]] é palíndromo.



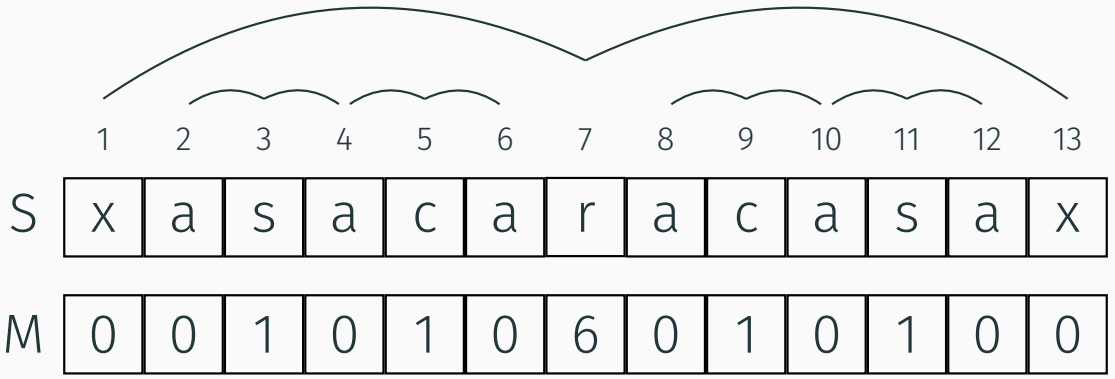
PALÍNDROMOS ÍMPARES

Consideremos apenas palíndromos de tamanho ímpar.

Centro de palíndromo (ímpar): posição do meio

Ex: “reviver” tem centro na letra i.

Seja M[i] é o maior inteiro tal que S[i − M[i] . . i + M[i]] é palíndromo.

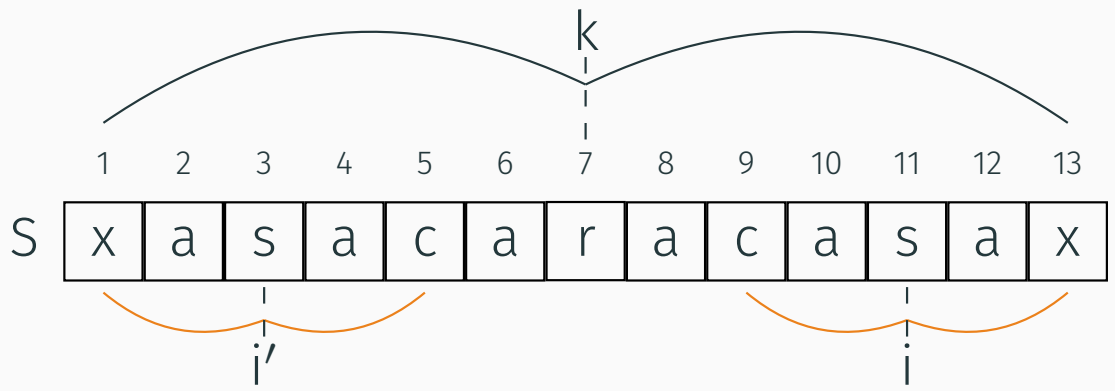


Como calcular M de forma rápida?

USANDO RESULTADOS ANTERIORES

Seja k tal que $k < i$ e $k + M[k] > i$.

Existe uma “cópia” da string em volta de i espelhada em k, centrada na posição $i' = 2k - i$.

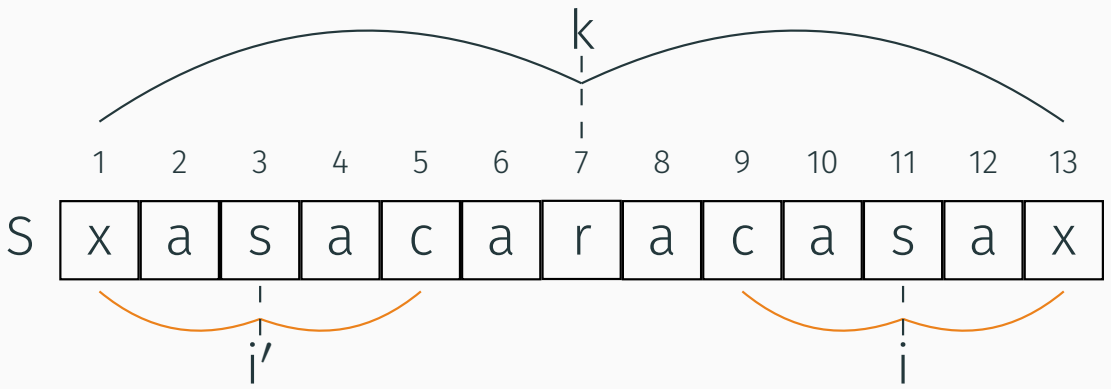


A string destacada centrada em i' é o reflexo da string destacada centrada em i.

USANDO RESULTADOS ANTERIORES - CASO 1

String espelhada: $S[i - (k + M[k] - i) \,..\, i + (k + M[k] - i)]$.

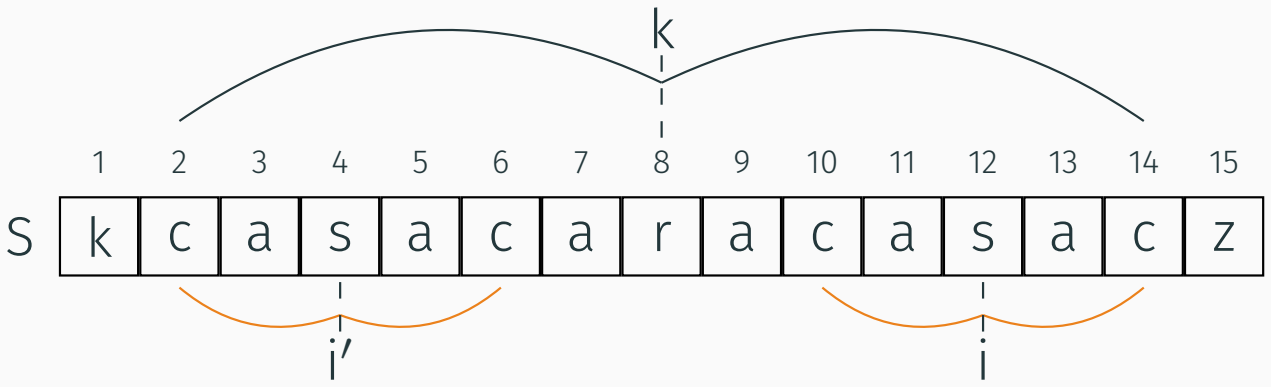
Se essa string não é palíndromo ($M[i'] < k + M[k] - i$), então $M[i] = M[i']$.



Nesse caso $M[i] = M[i'] = 1$.

USANDO RESULTADOS ANTERIORES - CASO 2

Se essa string é palíndromo ($M[i'] \geq k + M[k] - i$), temos que $M[i] \geq k + M[k] - i$.



Nesse caso $M[i] = 2$ e $M[i'] = 2$.

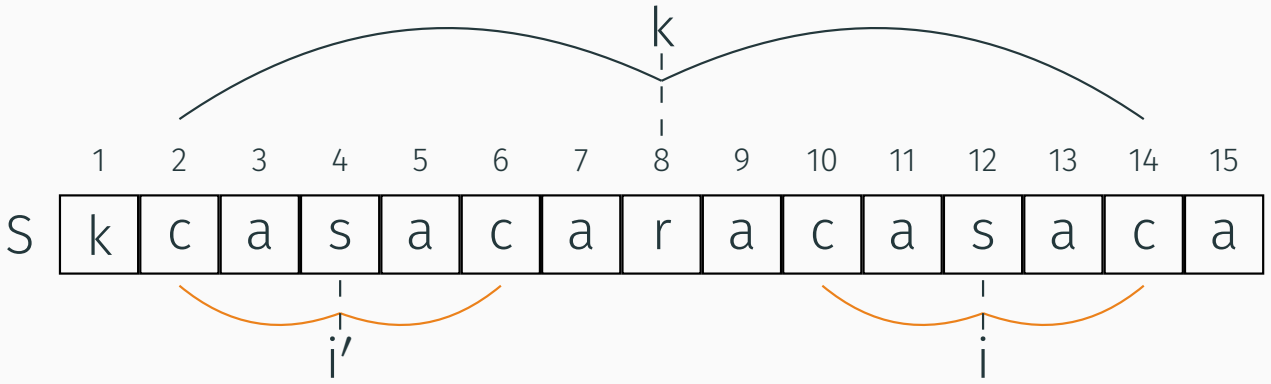
Pode ocorrer de

$$M[i] = k + M[k] - i = M[i']$$

ou...

USANDO RESULTADOS ANTERIORES - CASO 2

Se essa string é palíndromo ($M[i'] \geq k + M[k] - i$), temos que $M[i] \geq k + M[k] - i$.



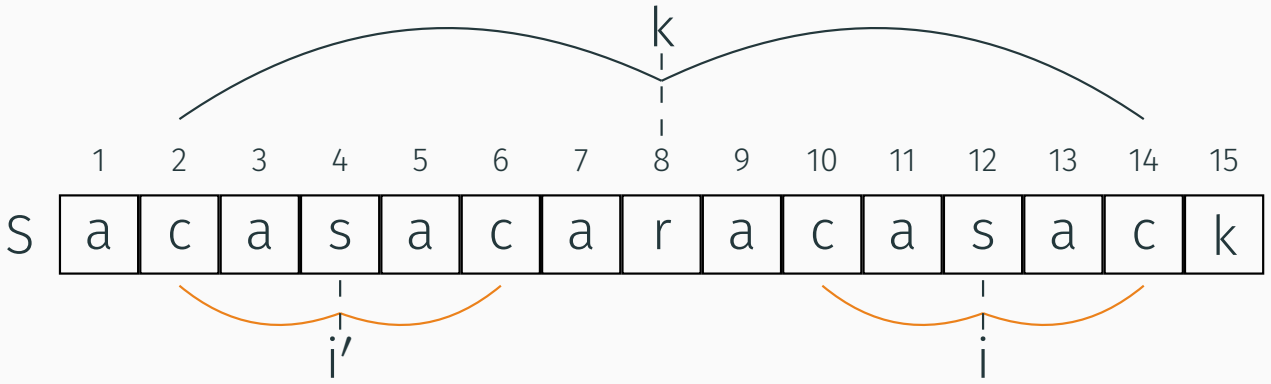
Nesse caso $M[i] = 3$ e $M[i'] = 2$.

ou...

$$M[i] > k + M[k] - i = M[i']$$

USANDO RESULTADOS ANTERIORES - CASO 2

Se essa string é palíndromo ($M[i'] \geq k + M[k] - i$), temos que $M[i] \geq k + M[k] - i$.



Nesse caso $M[i] = 2$ e $M[i'] = 3$.

ou...

$$M[i] = k + M[k] - i < M[i']$$

ALGORITMO

Ideia: Usar os valores M[i′] para i′ $<$ i para facilitar o cálculo de M[i].

Usar o k que maximiza k $+$ M[k] para partir da melhor cota inferior para M[i].

É possível manter o k ótimo enquanto calculamos os valores de M, e obter uma implementação que consome tempo $\mathcal{O}(|S|)$.

FINALIZAÇÃO, E PALÍNDROMOS PARES

Podemos assim descobrir a maior substring que é um palíndromo de tamanho **ímpar**.

É possível adaptar o algoritmo para lidar com palíndromos pares.

Por exemplo, modifique a string S para $\textsc{Fill}(S) = S[1]\$S[2]\$\ldots\$S[|S|]$. Os palíndromos de S se transformam em palíndromos ímpares em $\textsc{Fill}(S)$.

$$\texttt{abba} \rightarrow \texttt{a\$b\$b\$a}$$

TRIES

DEFINIÇÃO

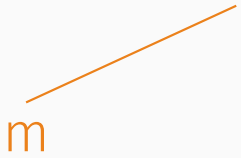
Trie: árvore enraizada que armazena um conjunto de strings.

Strings são representadas como caminhos a partir da raiz.

CONSTRUÇÃO

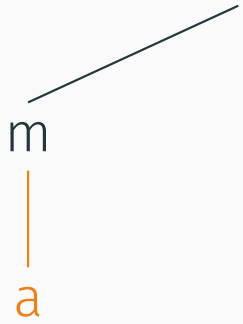
Adicionando “mata”.

CONSTRUÇÃO



Adicionando “mata”.

CONSTRUÇÃO

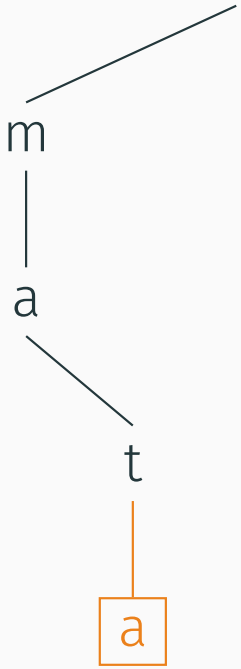


Adicionando “mata”.

CONSTRUÇÃO

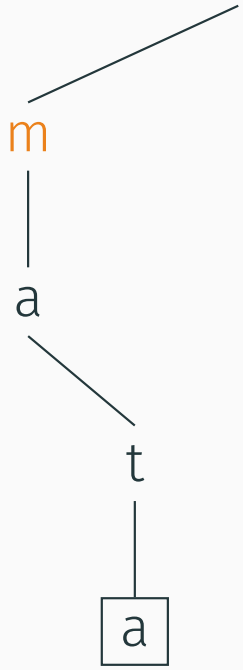
Adicionando “mata”.

CONSTRUÇÃO



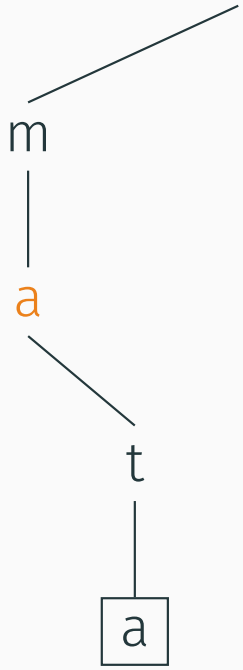
Adicionando “mata”.

CONSTRUÇÃO



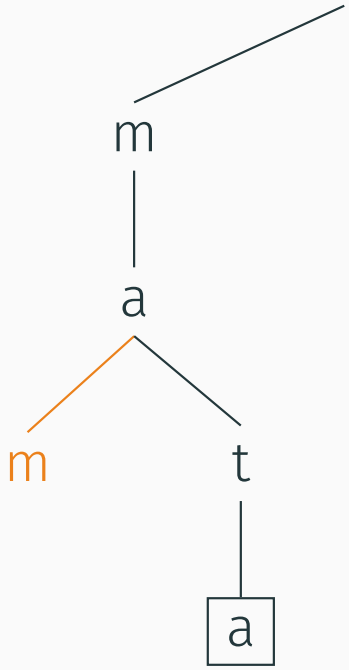
Adicionando “mamata”.

CONSTRUÇÃO



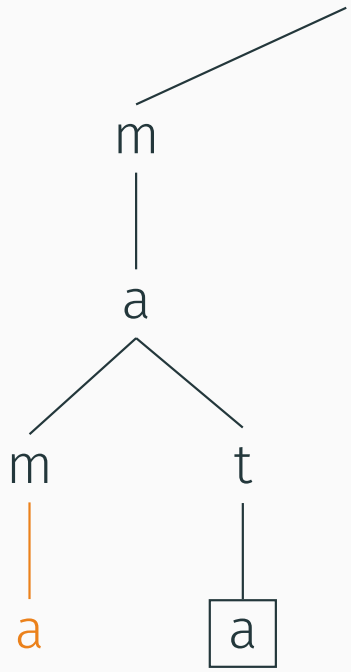
Adicionando “mamata”.

CONSTRUÇÃO



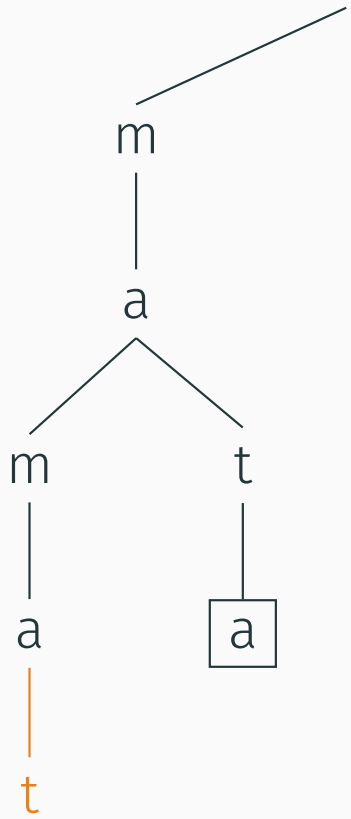
Adicionando “mamata”.

CONSTRUÇÃO



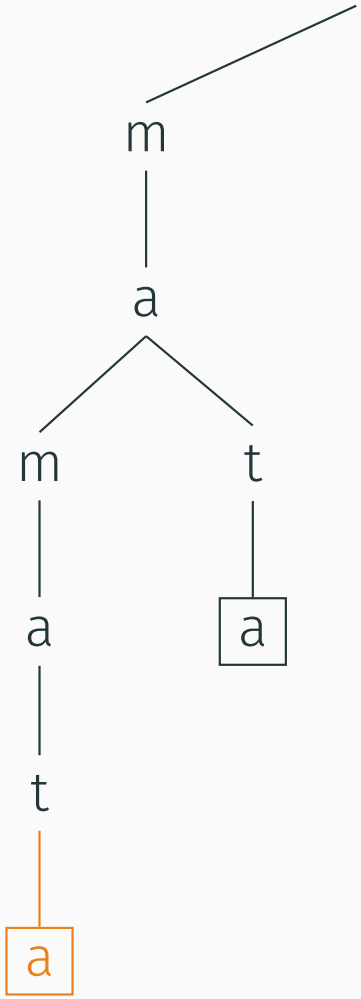
Adicionando “mamata”.

CONSTRUÇÃO



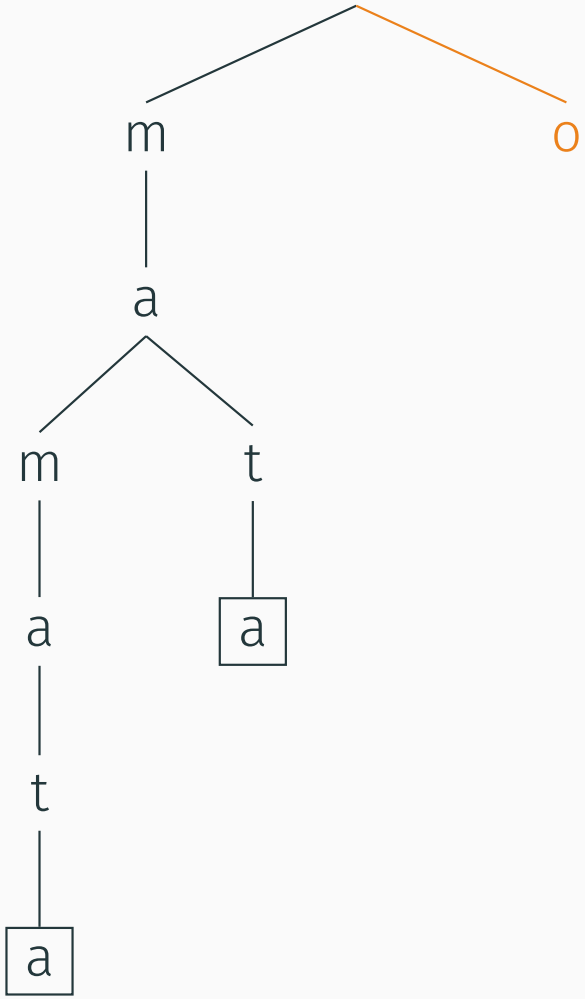
Adicionando “mamata”.

CONSTRUÇÃO



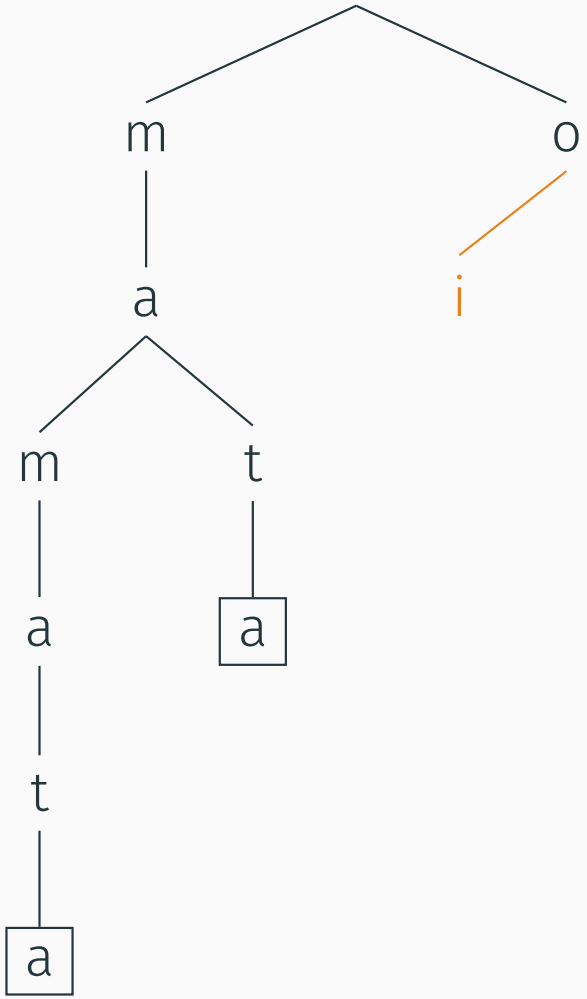
Adicionando “mamata”.

CONSTRUÇÃO



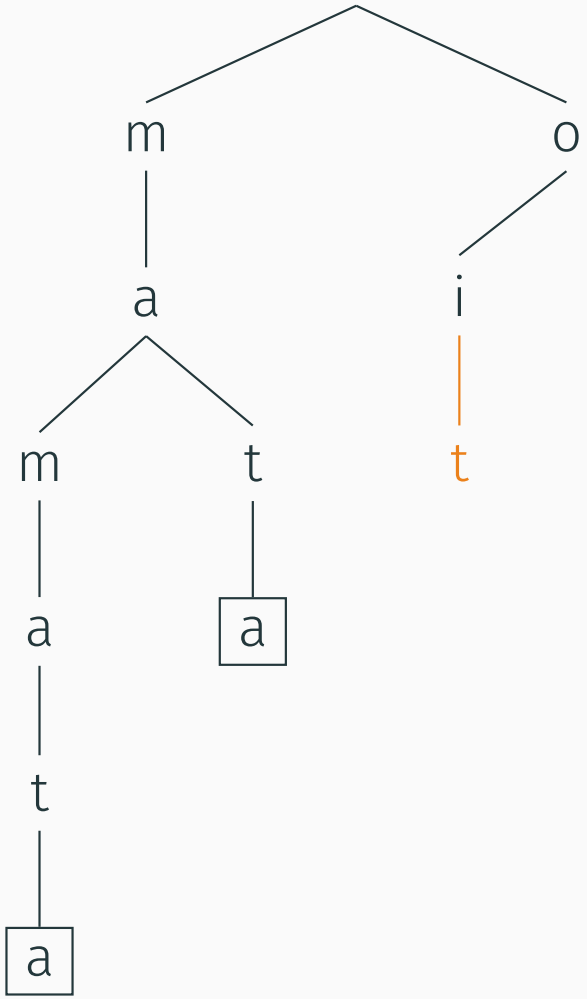
Adicionando “oito”.

CONSTRUÇÃO



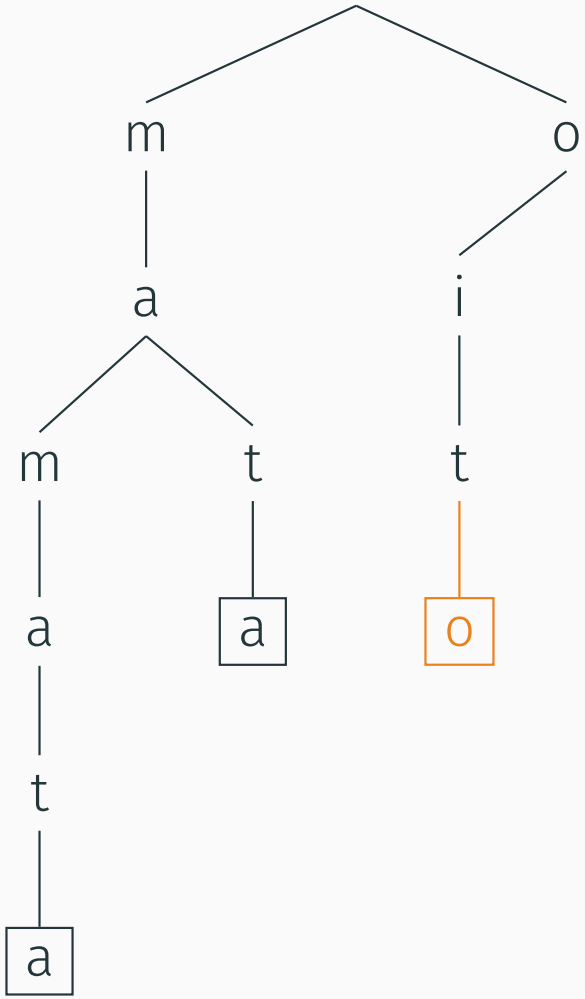
Adicionando “oito”.

CONSTRUÇÃO



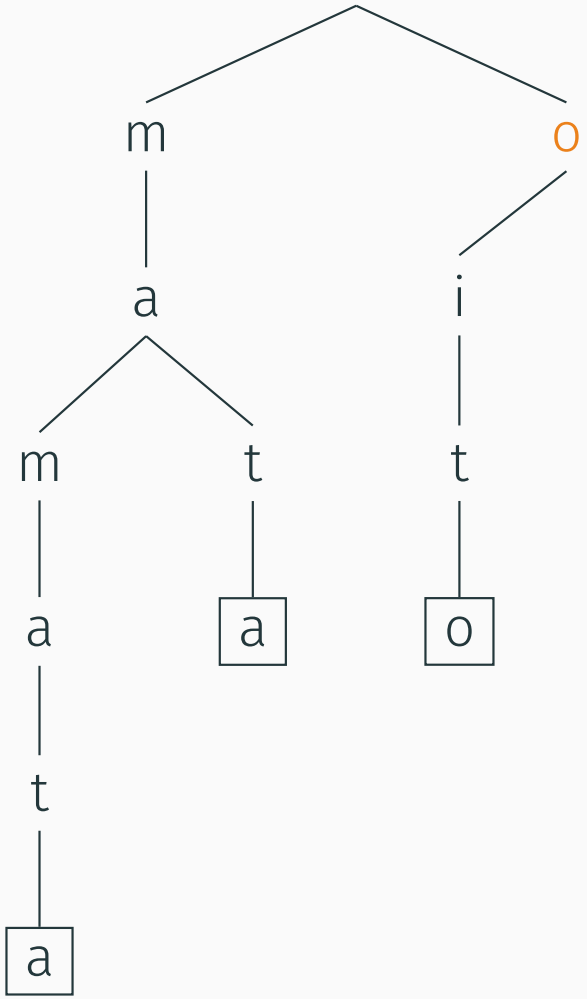
Adicionando “oito”.

CONSTRUÇÃO



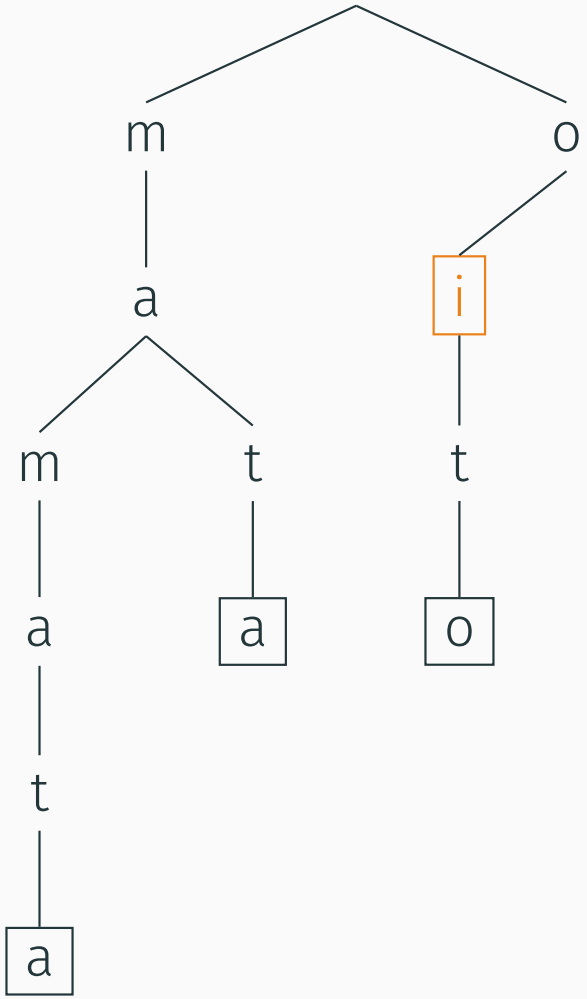
Adicionando “oito”.

CONSTRUÇÃO



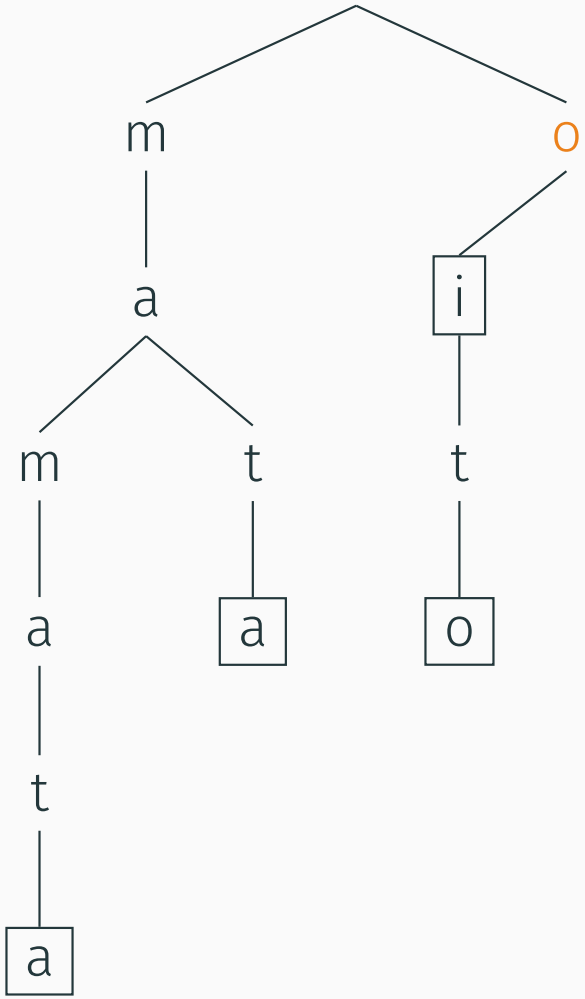
Adicionando “oi”.

CONSTRUÇÃO



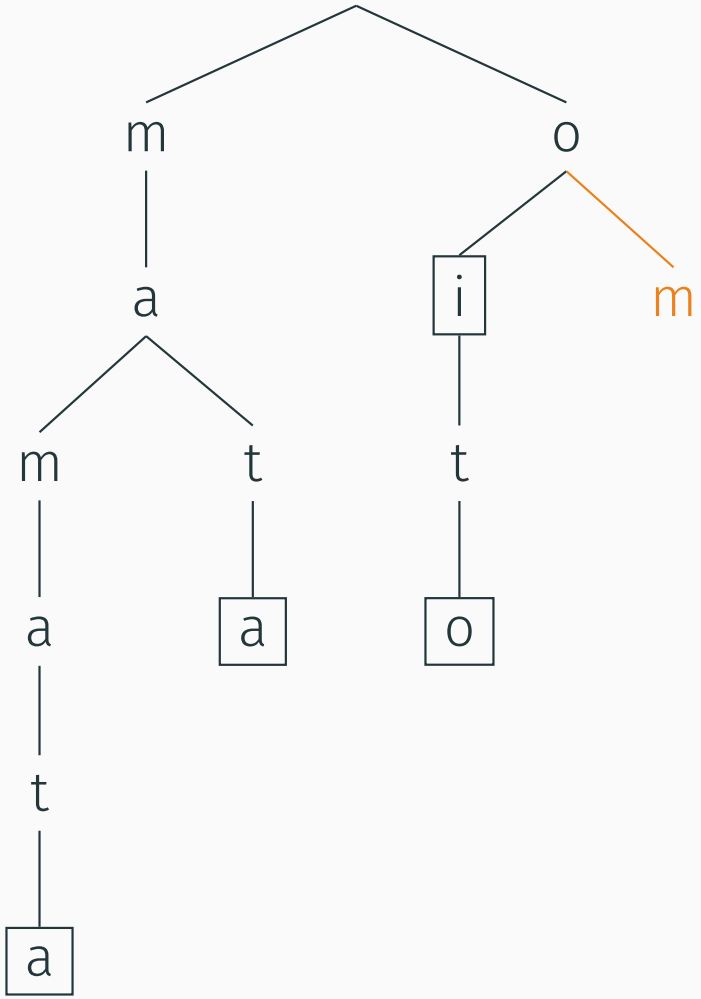
Adicionando “oi”.

CONSTRUÇÃO



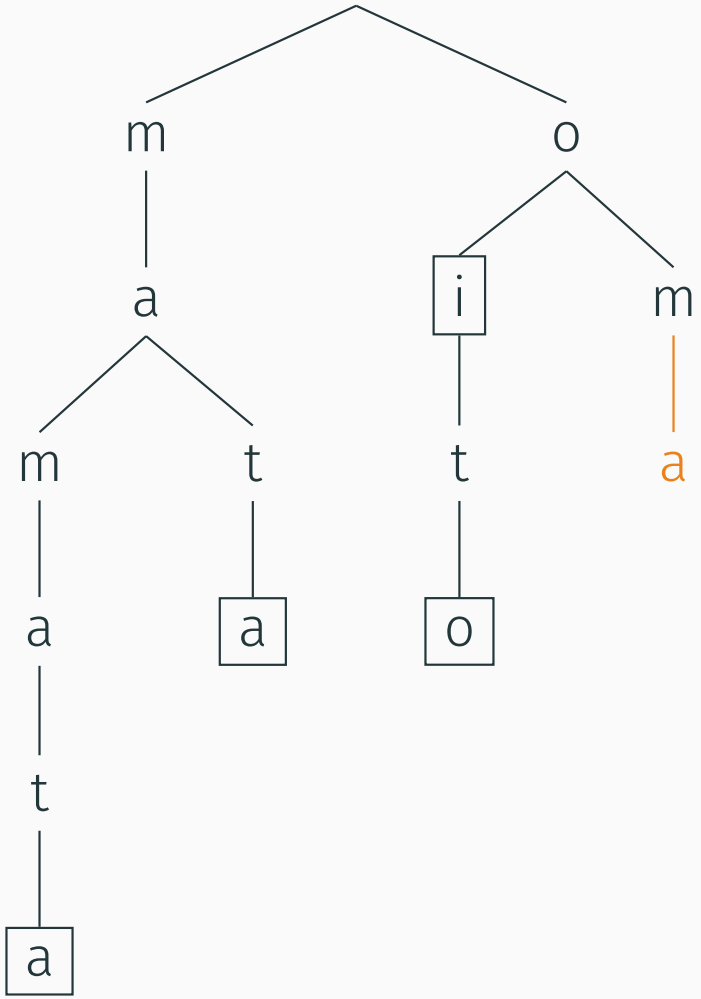
Adicionando “omar”.

CONSTRUÇÃO



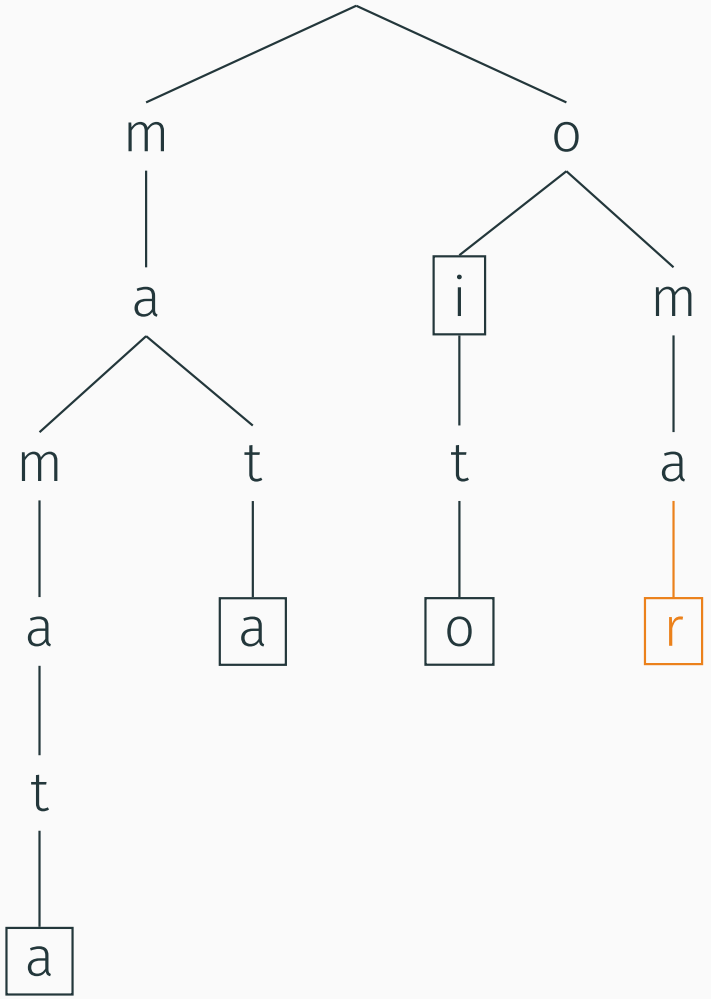
Adicionando “omar”.

CONSTRUÇÃO



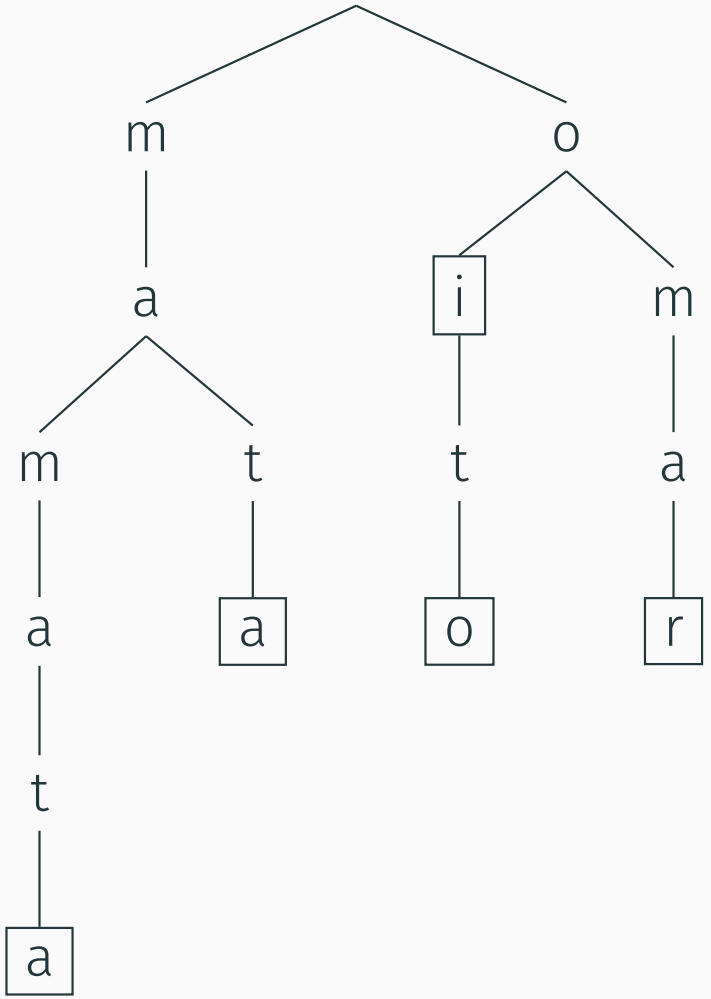
Adicionando “omar”.

CONSTRUÇÃO



Adicionando “omar”.

CONSTRUÇÃO



Adicionando “omar”.

USOS

Construir uma trie para $\mathcal{S} = \{S_1, \ldots, S_k\}$ consome tempo $\mathcal{O}(\sum_{i=1}^{k} |S_i|)$.

Com esta trie, podemos realizar:

CONTAINS(S) Determina se $S \in \mathcal{S}$.

LCP(S) Determina o maior prefixo comum de S com alguma string de $\mathcal{S}$.

Consumo de tempo: $\mathcal{O}(|S|)$.

AHO-CORASICK

Problema: Determine todas as ocorrências de todas as strings de $\mathcal{S} = \{S_1, \ldots, S_k\}$ em T.

Problema: Determine todas as ocorrências de todas as strings de $\mathcal{S} = \{S_1, \ldots, S_k\}$ em T.

Para cada sufixo $T[i \ldots |T|]$, usando uma trie, determinamos as strings de $\mathcal{S}$ que ocorrem **no início** de $T[i \ldots |T|]$.

Isso leva tempo $\mathcal{O}(|T|^2 + \sum_{i=1}^{k} |S_i||\Sigma|)$.

LINKS DE FALHA

No KMP: função prefixo guarda, para cada i, o comprimento do maior prefixo de T que é sufixo próprio de T[1 . . i].

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
T	a	c	a	s	a	c	a	r	a	c	a	s	a	c
		0	1	0	1	2	3	0	1	2	3	4	5	6

No KMP: função prefixo guarda, para cada i, o comprimento do maior prefixo de T que é sufixo próprio de T[1 . . i].

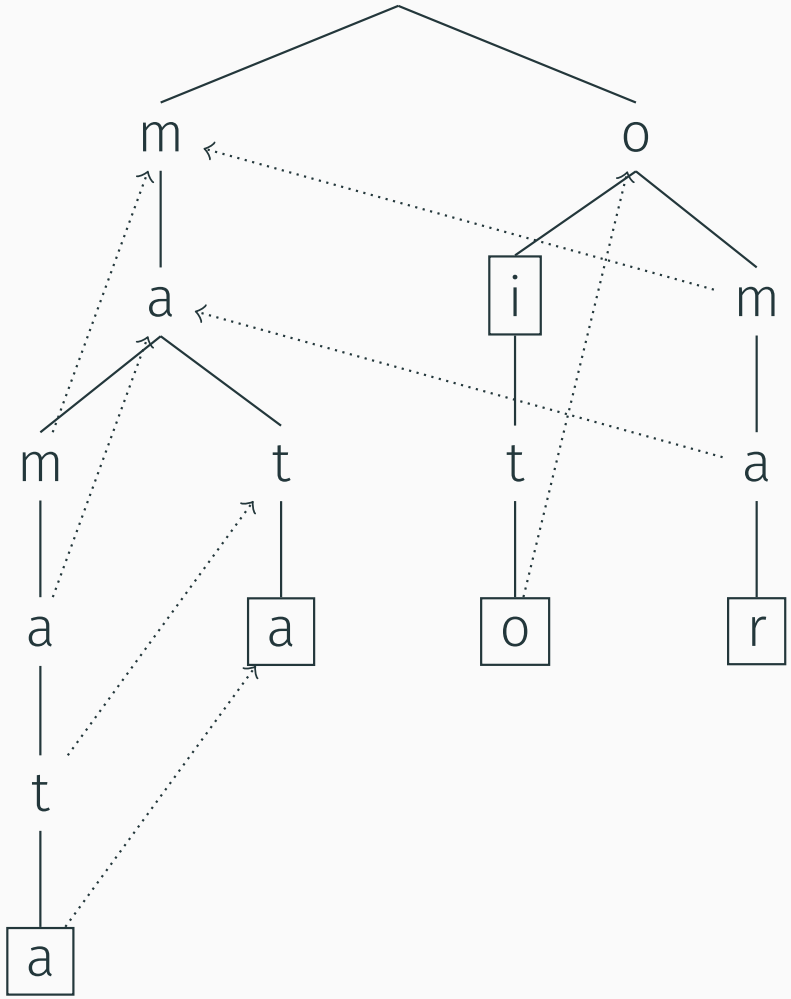
	1	2	3	4	5	6	7	8	9	10	11	12	13	14
T	a	c	a	s	a	c	a	r	a	c	a	s	a	c
		0	1	0	1	2	3	0	1	2	3	4	5	6

Em Tries: para cada nó v, guarda o nó mais profundo cuja string seja um sufixo próprio da string de v.

link de falha

LINKS DE FALHA

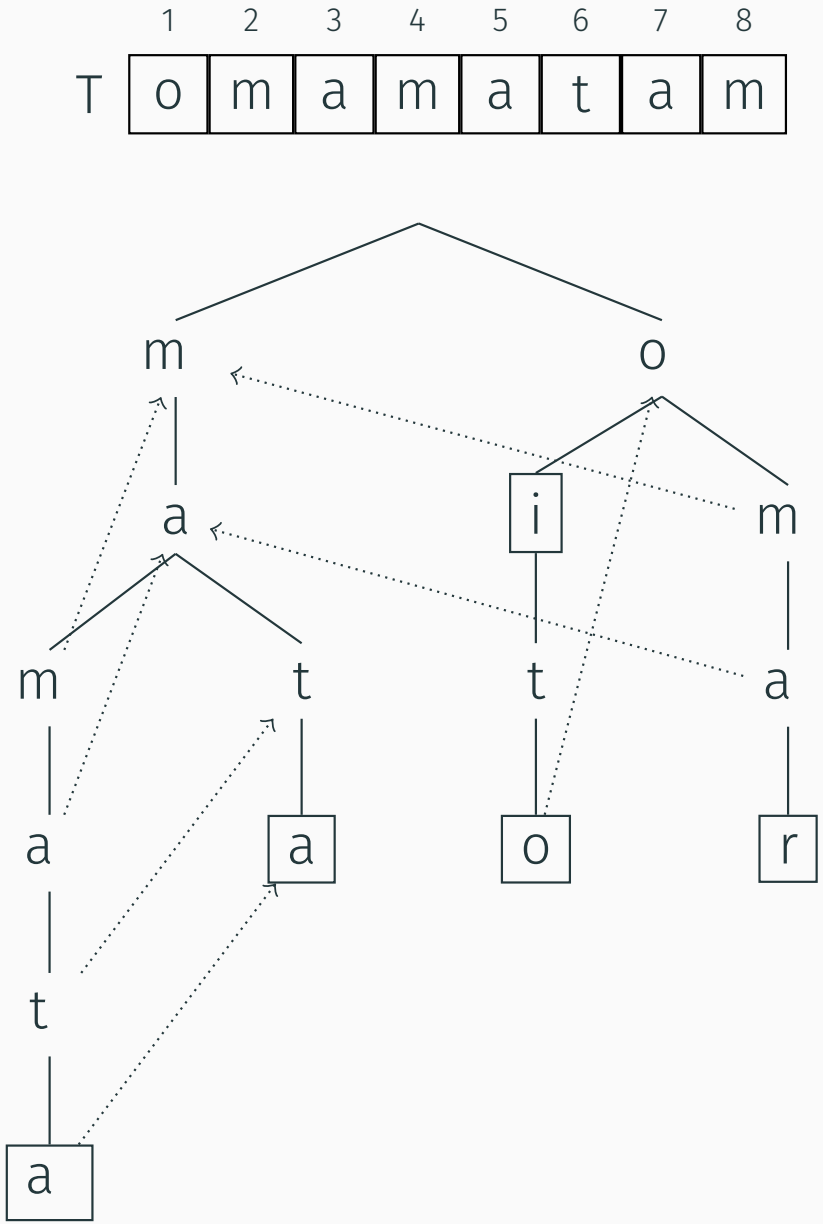
Para cada nó v, guardar o nó mais profundo cuja string seja um sufixo próprio da string de v.



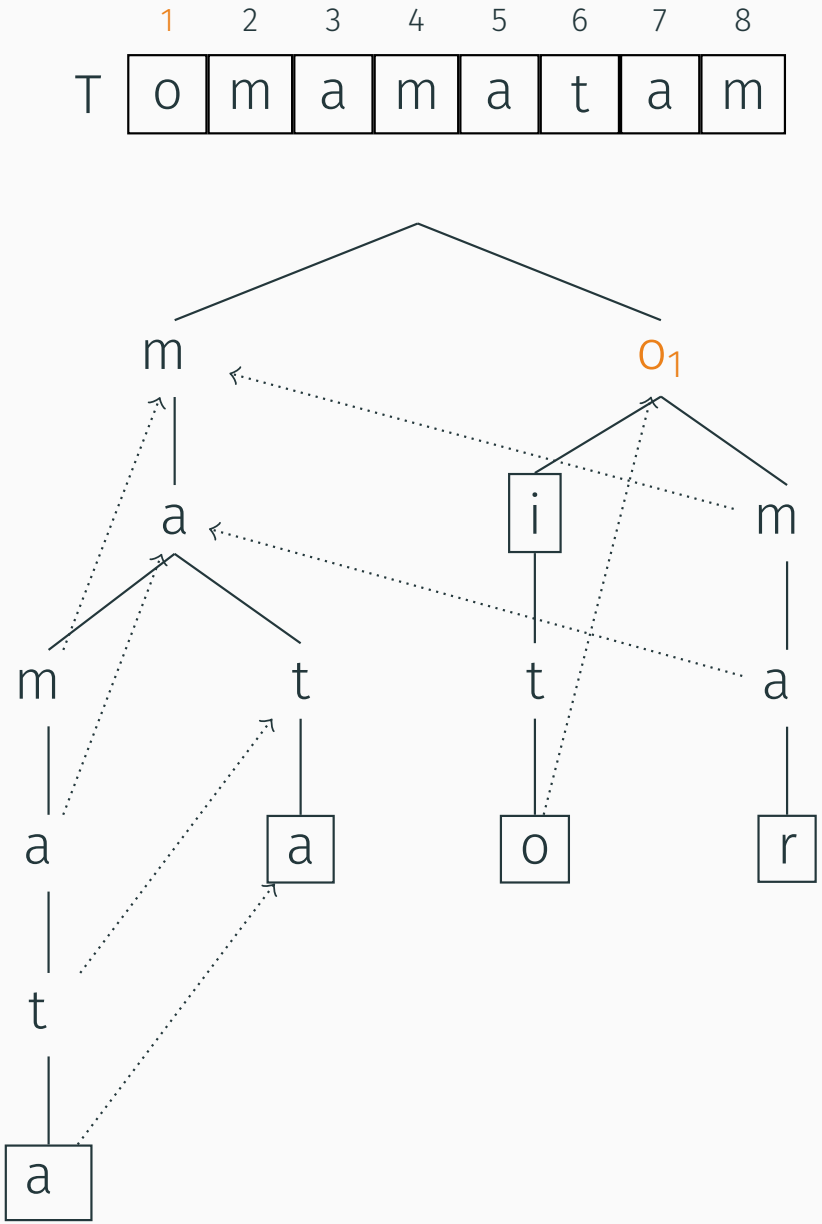
ALGORITMO

Para cada prefixo T[1 . . i], determinar o maior prefixo S[1 . . j] de alguma string de $\mathcal{S}$ que é sufixo de T[1 . . i].

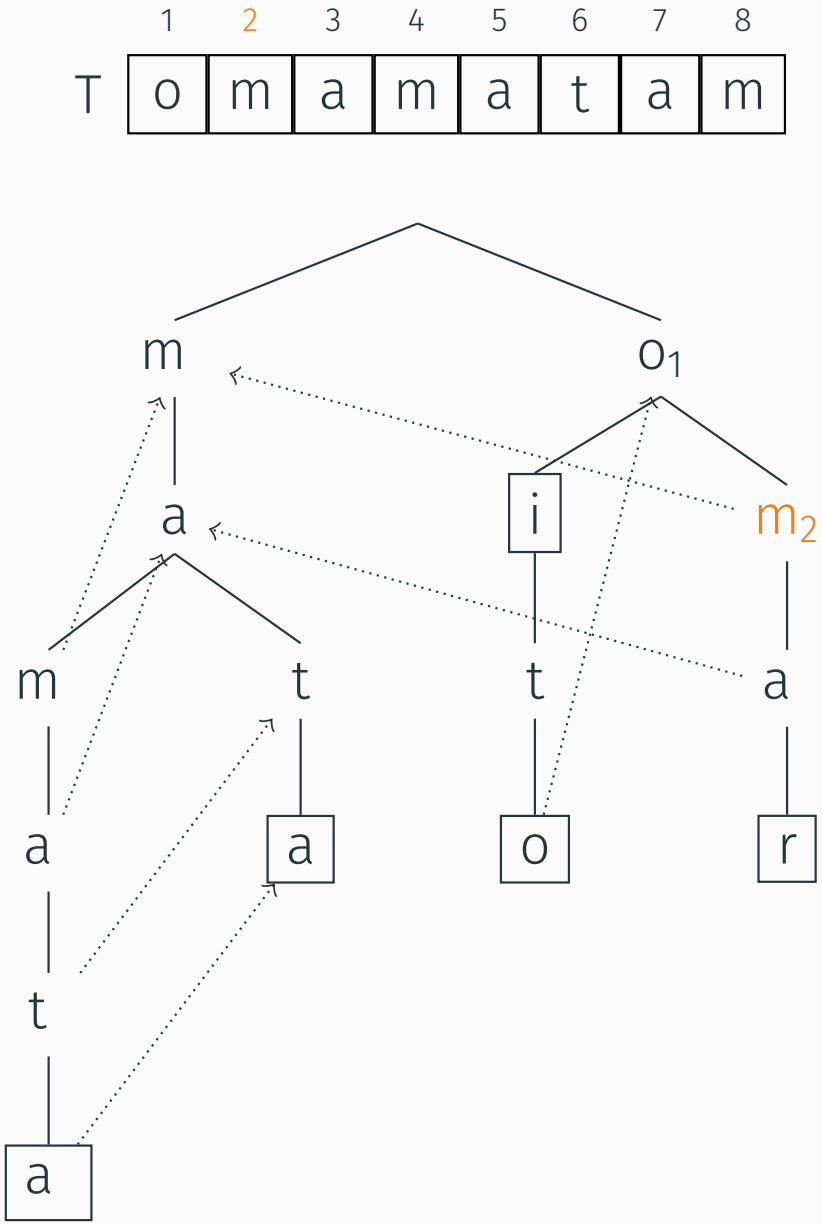
ALGORITMO



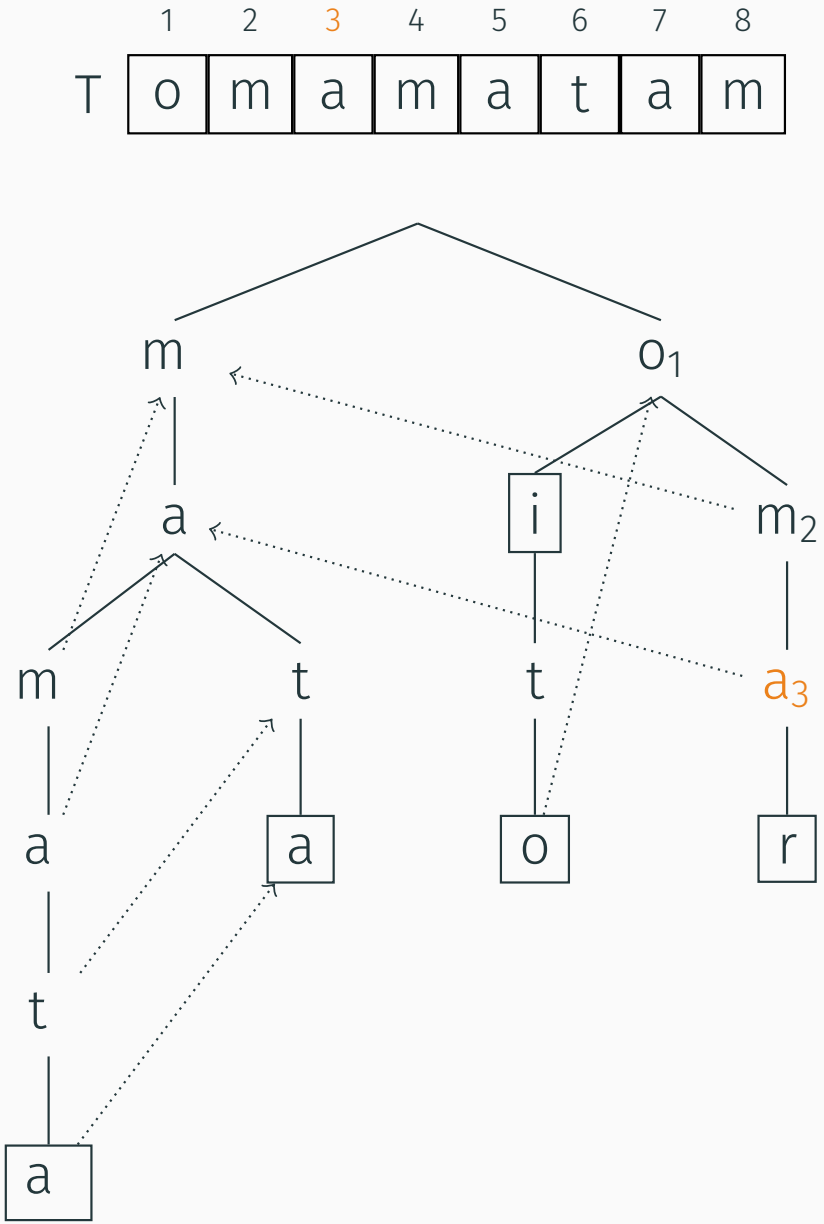
ALGORITMO



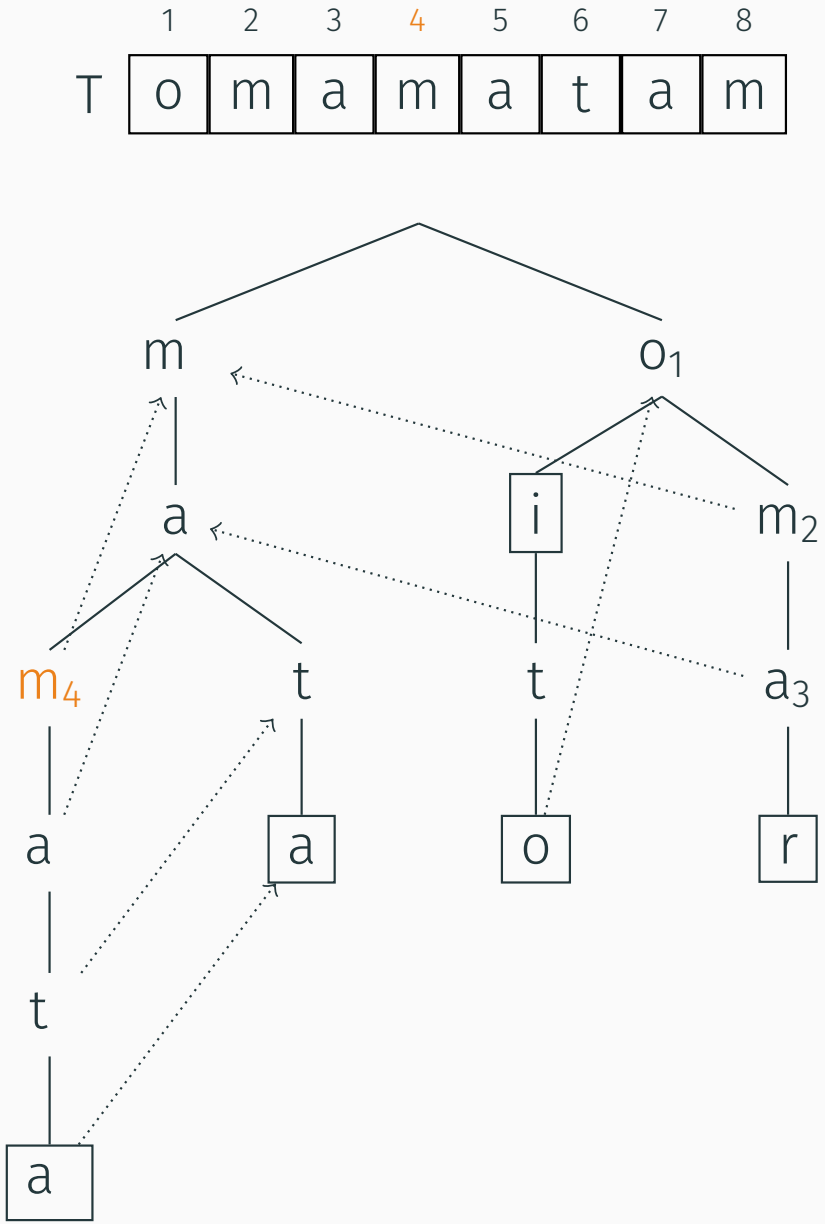
ALGORITMO



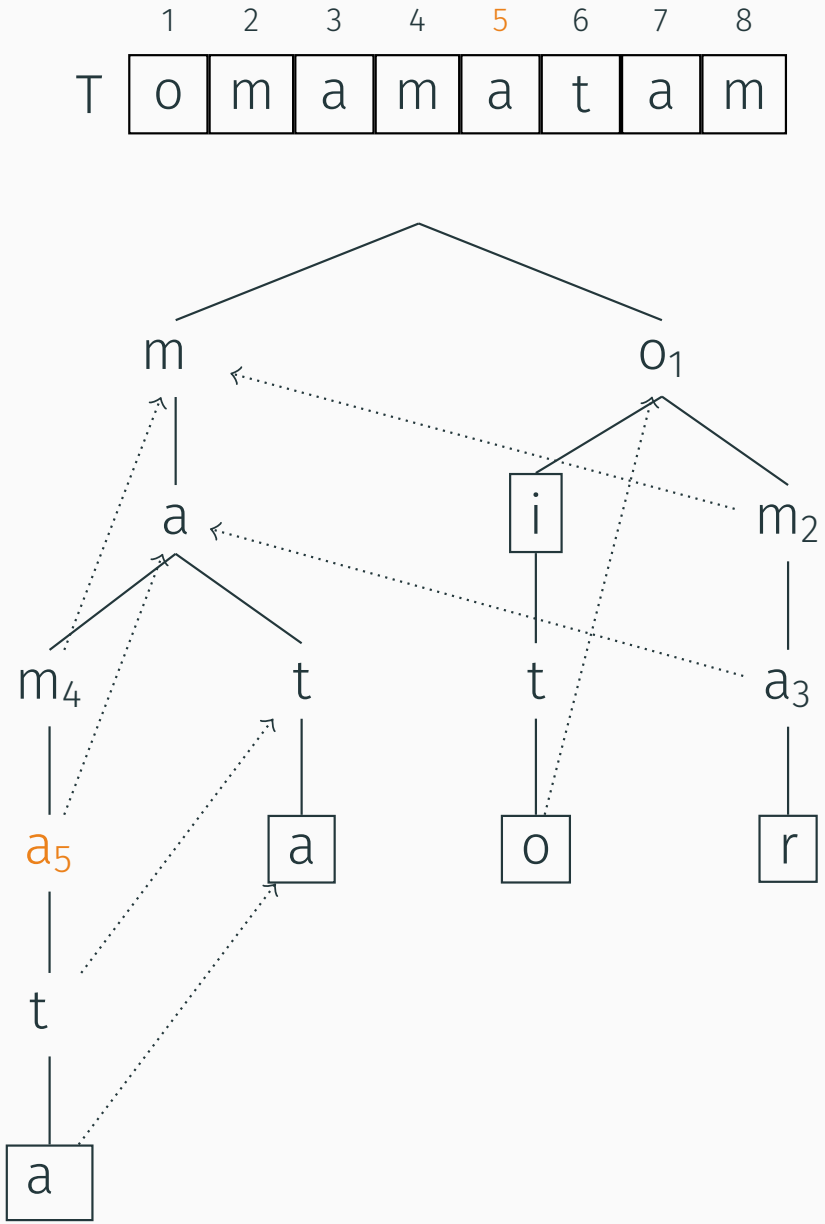
ALGORITMO



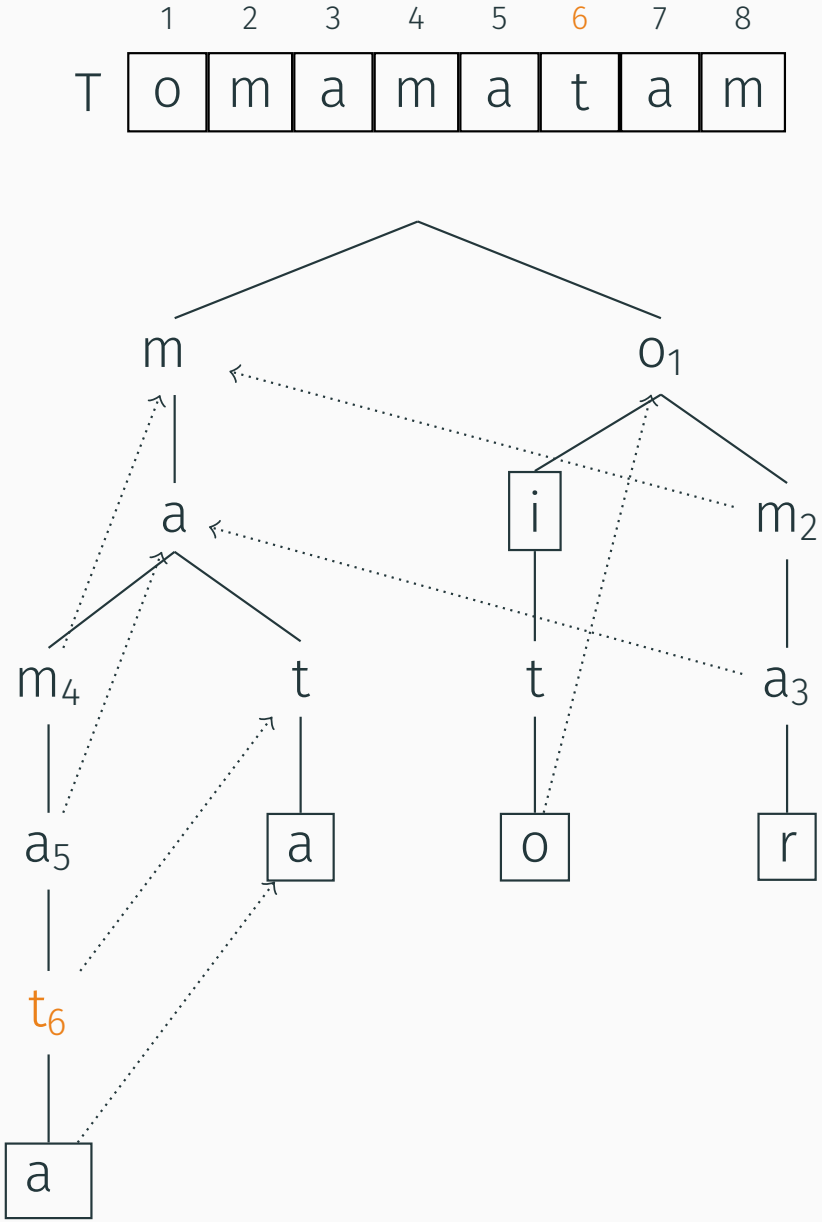
ALGORITMO



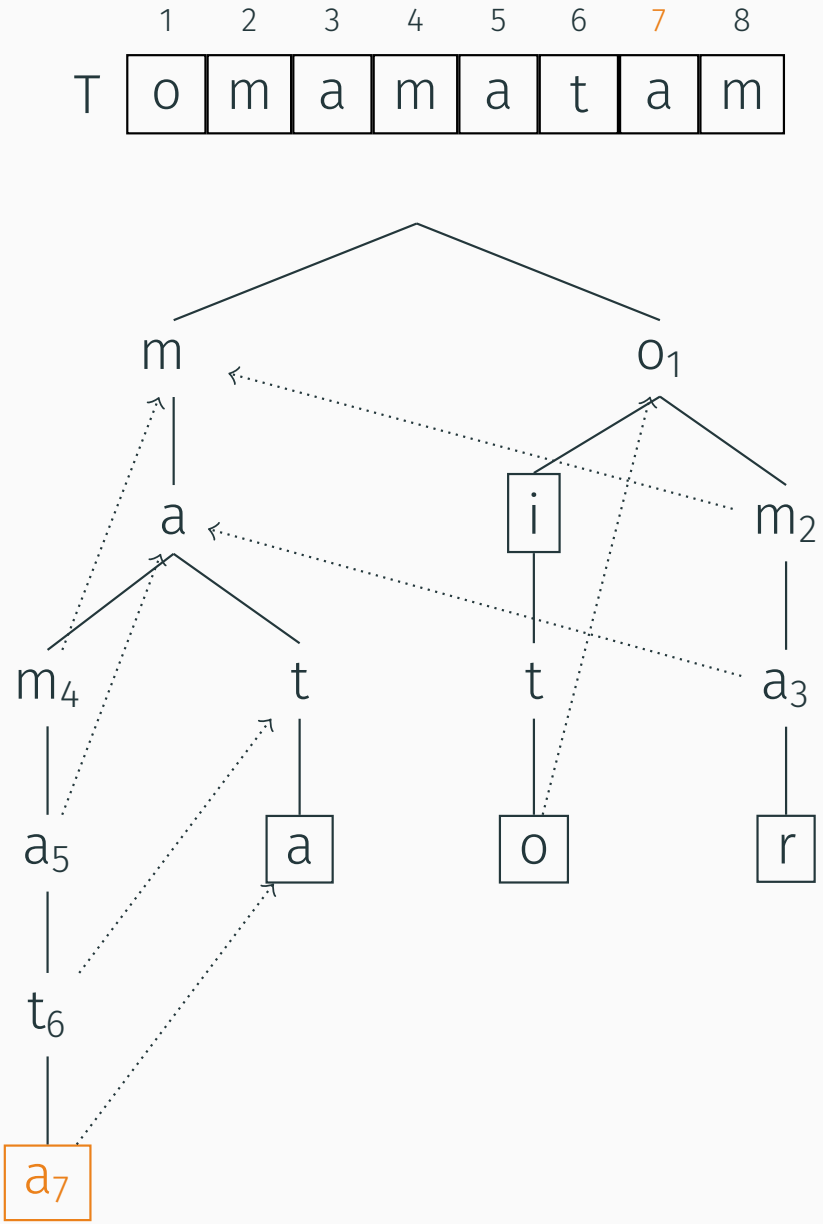
ALGORITMO



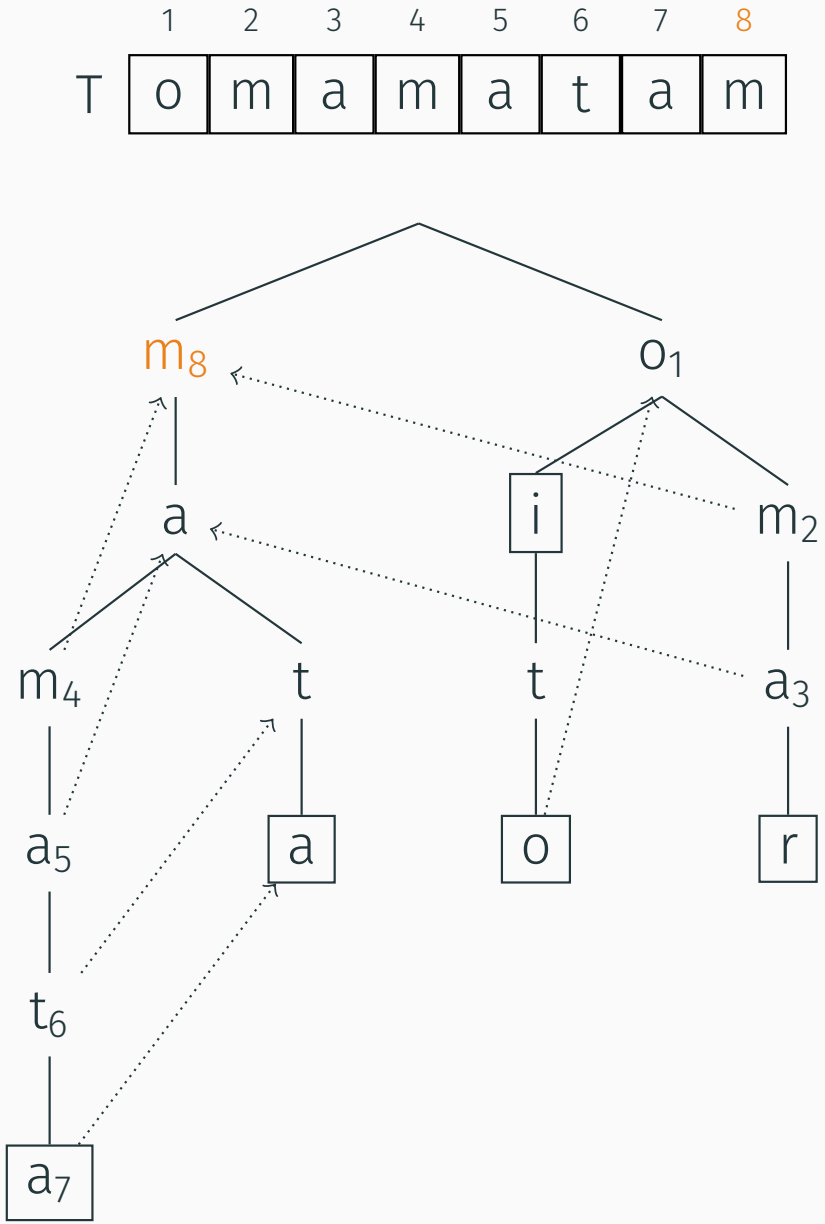
ALGORITMO



ALGORITMO



ALGORITMO



CONSIDERAÇÕES FINAIS

Uma string de $\mathcal{S}$ pode ser sufixo próprio de S[1 . . j]!

link de ocorrência de v: vértice que representa a maior string de $\mathcal{S}$ que é sufixo próprio da string de v.

CONSIDERAÇÕES FINAIS

Uma string de $\mathcal{S}$ pode ser sufixo próprio de $S[1..j]$!

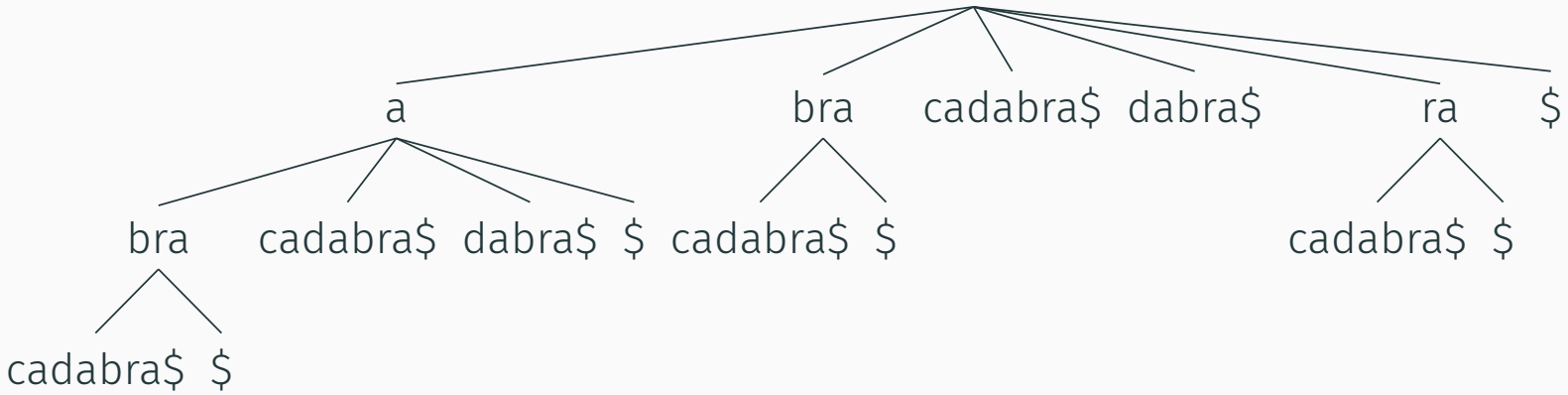
link de ocorrência de v: vértice que representa a maior string de $\mathcal{S}$ que é sufixo próprio da string de v.

O algoritmo pode ser implementado em tempo $\mathcal{O}(\sum_{i=1}^{k} |S_i||\Sigma| + |T| + x)$, onde x é o número de ocorrências.

OUTROS TÓPICOS

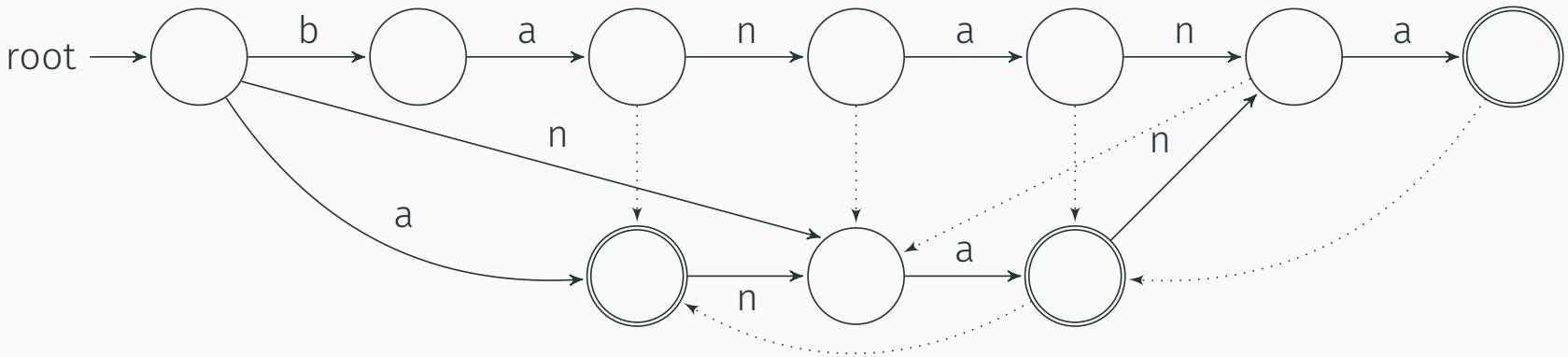
ÁRVORE DE SUFIXOS

Trie comprimida para todos os sufixos de uma string.



AUTÔMATO DE SUFIXOS

Autômato que aceita todos os sufixos de uma string.



Perguntas?