Complexidade de Tempo

Complexidade de Tempo

- Quando um problema é decidível, ele pode não ser solúvel na prática se a solução requer uma quantidade excessiva de tempo ou memória

Medindo a complexidade

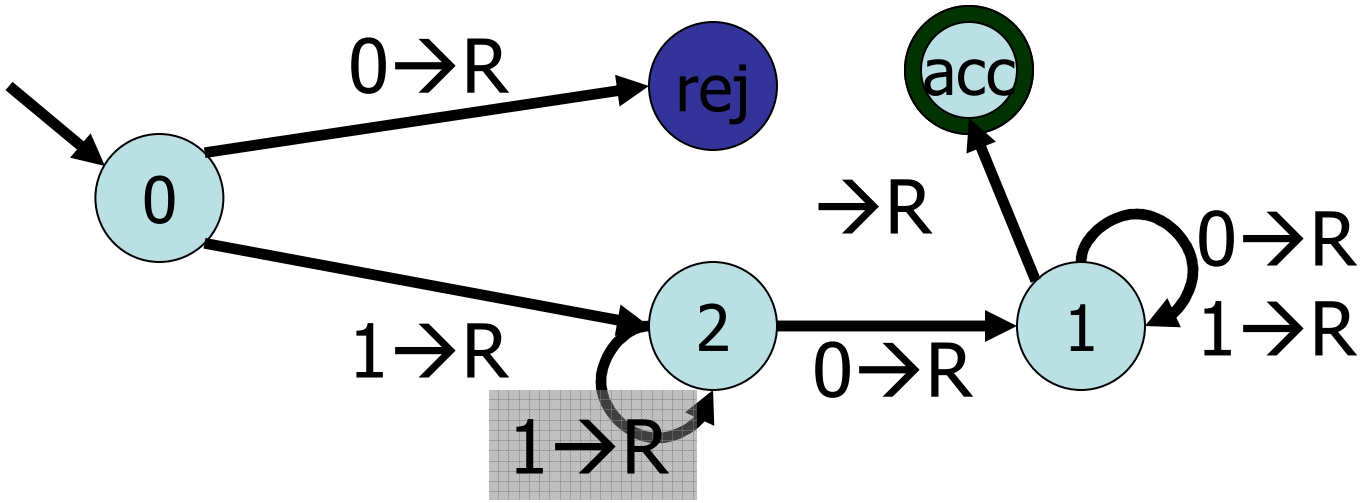
- Ex.: Seja a linguagem $A=\{0^k1^k \mid k \geq 0\}$.
- Quanto tempo uma TM de uma única fita precisa para decidir A?

Tempo Polinomial vs. Tempo Exponencial

A experiência mostra que a linha divisória entre problemas solúveis em *tempo polinomial* vs. problemas que requerem *tempo exponencial* é fundamental. Vamos definir o tempo de execução de TM's:

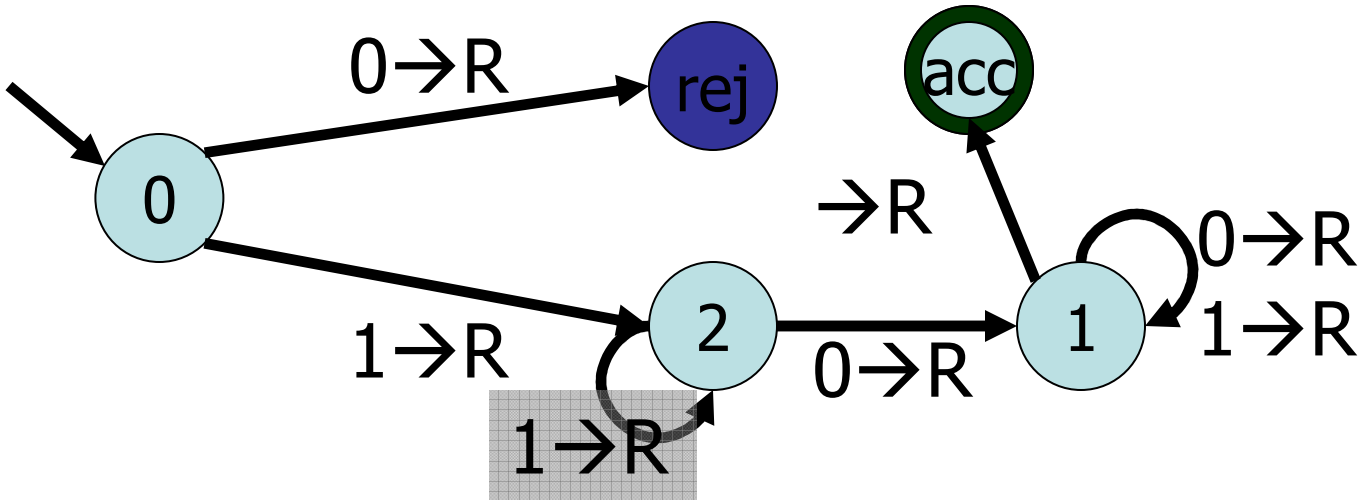
DEF: O ***tempo de execução*** de uma TM *M* é a função $f(n)$ = o máximo número de transições tomadas por *M* até parar, dada uma entrada arbitrária de comprimento *n*.

Tempo de Execução



Q: Qual é o tempo de execução da máquina acima?

Tempo de Execução



R: Supondo $\Sigma = \{0,1\}$, o tempo de execução é $T(n)=n+1$, como se pode ver para qq string começando em 1. Mesmo que a TM execute mais rápido sobre 0{0,1}*, o tempo de execução é definido em termos do pior caso. Falhas não afetam isso.

Complexidade Polinomial

DEF: *M* tem ***complexidade de tempo polinomial*** se existe um polinômio $p(n)$ tal que o tempo de execução de *M*, $f(n)$, satisfaz a:

$$f(n) \leq p(n) \ .$$

Notação Big-*O*

DEF: Sejam *f* e *g* funções de domínio $\mathbf{R}_{\geq 0}$ ou **N** e codomínio **R**. Se existem constantes *C* e *k* tais que

$$\forall\ x > k,\ |f(x)| \leq C \cdot |g(x)|$$

i.e., depois de *k*, *f* é menor ou igual a um múltiplo de *g*, então escrevemos:

$$f(x) = O(g(x))$$

Big-Ω e Big-Θ

Big-Ω é apenas o reverso de big-*O*. I.e.

$$f(x) = \Omega(g(x)) \leftrightarrow g(x) = O(f(x))$$

Portanto big-Ω diz que $f(x)$ *domina* $g(x)$ assintoticamente.

Big-Θ diz que ambas as funções dominam uma à outra e portanto são assintoticamente equivalentes. I.e.

$$f(x) = \Theta(g(x))$$

$$\leftrightarrow$$

$$f(x) = O(g(x)) \wedge f(x) = \Omega(g(x))$$

Sinônimo para $f = \Theta(g)$: "*f* é ***da ordem de*** *g*"

Fatos úteis

- Qualquer polinômio é big-Θ de seu maior termo
  - EX: $x^4/100000 + 3x^3 + 5x^2 - 9 = \Theta(x^4)$
- A soma de duas funções é big-*O* da maior delas
  - EX: $x^4 \ln(x) + x^5 = O(x^5)$
- Constantes diferentes de zero são irrelevantes:
  - EX: $17x^4 \ln(x) = O(x^4 \ln(x))$

Big-*O*, Big-Ω, Big-Θ. Exemplos

Q: Ordene os seguintes do menor para o maior assintoticamente. Agrupe juntas todas as funções que são big-Θ equivalentes:

$$x+\sin x, \ln x, x+\sqrt{x}, \frac{1}{x}, 13+\frac{1}{x}, 13+x, e^x, x^e, x^x$$

$$(x+\sin x)(x^{20}-102), x\ln x, x(\ln x)^2, \lg_2 x$$

Big-*O*, Big-Ω, Big-Θ. Exemplos

R:

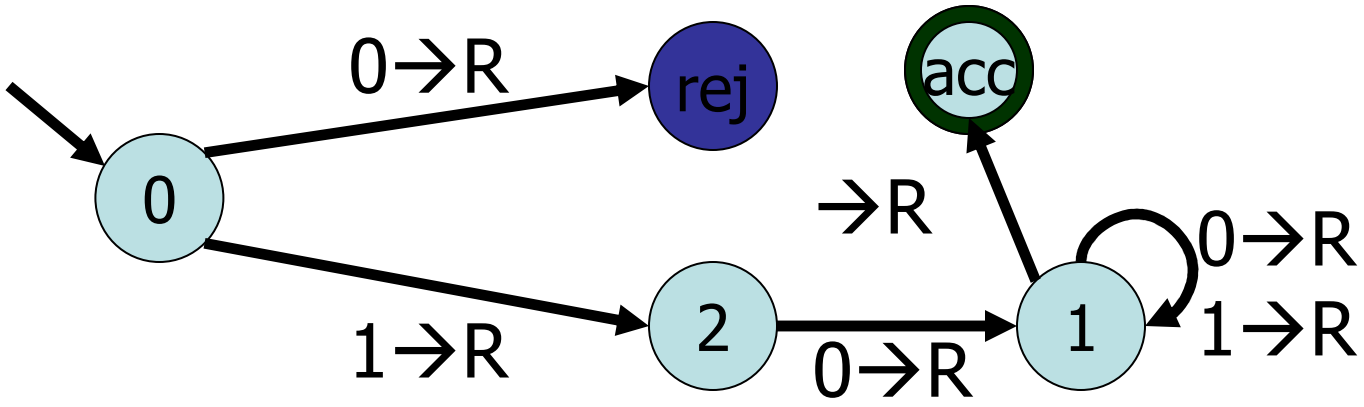
1. $1/x$
2. $13+1/x$
3. $\ln x,\ \lg_2 x$ (mudança de base)
4. $x+\sin x, x+\sqrt{x}, 13+x$
5. $x\ln x$
6. $x(\ln x)^2$
7. x^e
8. $(x+\sin x)(x^{20}-102)$
9. e^x
10. x^x

Classes de Complexidade

Classes de complexidade de tempo são as próximas classes de linguagens que vamos estudar:

DEF: Suponha que $g(n)$ é uma função sobre reais. A ***classe de complexidade de tempo*** TIME($g(n)$) consiste de todas as linguagens que podem ser decididas por uma TM em tempo de execução $O(g(n))$. Qualquer dessas linguagens é dita ter ***complexidade de tempo*** $g(n)$.

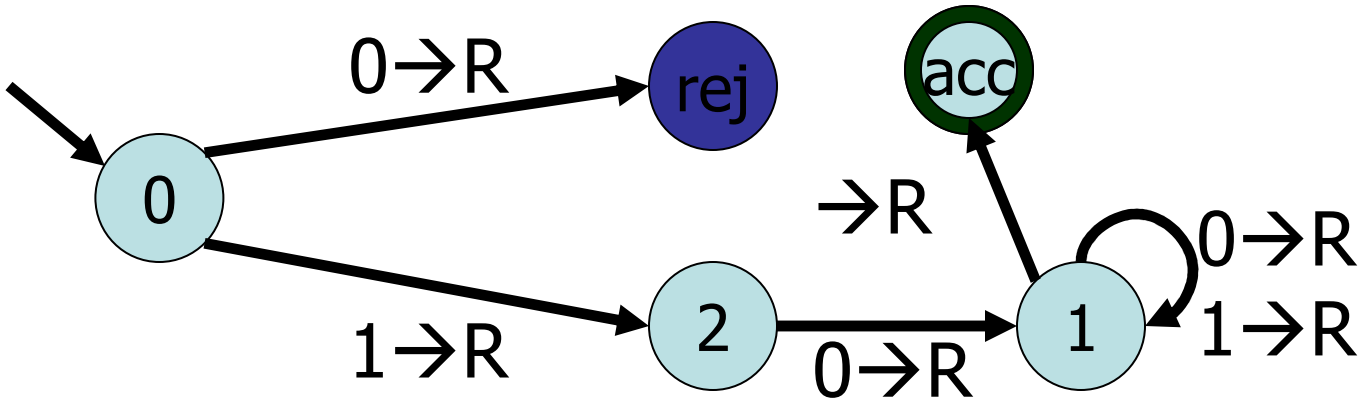
Complexidade de Tempo



Q: A linguagem aceita pela TM acima tem que complexidade de tempo O($g(n)$) ?

1. $g(n) = n + 2$
2. $g(n) = n/2$
3. $g(n) = n^2$
4. $g(n) = 1$

Complexidade de Tempo

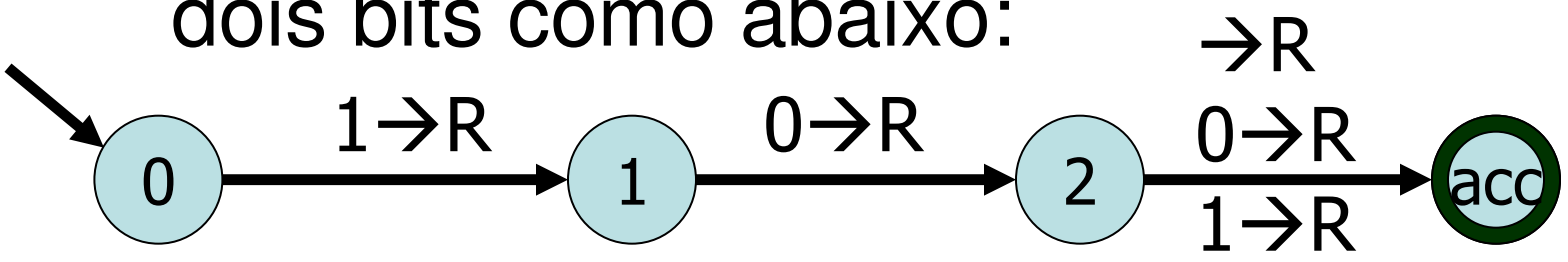


A: TODAS! Este decisor particular para a linguagem tem tempo de execução $f(n) = n+2$, o qual é big-O dos no.'s 1, 2 e 3. Mas a definição não diz que devemos nos pautar por uma dada implementação:

Q: Como se pode modificar a TM para que a linguagem aceita esteja em TIME(1) ?

Complexidade de Tempo

R: A linguagem aceita é 10{0,1}*.
Portanto, basta verificar os primeiros dois bits como abaixo:



Agora vemos que a linguagem é aceita por uma TM cujo tempo de execução é *constante*.

A Classe **P**

P é a classe das linguagens que podem ser decididas por uma TM em tempo de execução de complexidade polinomial. I.e.,

DEF: $$\mathbf{P} = \bigcup_{k=0}^{\infty} \mathrm{TIME}(n^k)$$

De RAM para TM em Tempo Polinomial

THM 1: Qualquer programa RAM que executa em tempo $O(f(n))$ pode ser simulado por um TM multi-fita que executa em tempo $O(f(n)^3)$.

THM 2: Qualquer TM multi-fita que executa em tempo $O(g(n))$ pode ser simulada por uma TM 1-fita que executa em tempo $O(g(n)^2)$.

COR: Qualquer programa RAM que executa em tempo $O(f(n))$ pode ser simulado por uma TM 1-fita que executa em tempo $O(f(n)^6)$.

Consequentemente, qualquer algoritmo que executa em tempo polinomial pode ser simulado no seu computador pode ser executado em uma TM em tempo polinomial.

De RAM para TM em Tempo Polinomial

Não vamos provar, mas apenas explicar:

- Uma RAM é uma máquina com programas estilo "goto" e memória constituída de um número arbitrário de inteiros, denominados registradores, cada um tendo um endereço inteiro. Cada passo de execução da RAM age sobre um dos registradores[1]. Depois de k passos, a RAM pode ter no máximo $O(k)$ registradores, cada um contendo um inteiro de tamanho $O(k)$. Se mantemos os conteúdos desses registradores em uma das fitas da TM, mudar o estado da RAM simulada requer ler todos os $O(k)$ registradores de tamanho $O(k)$, portanto $O(k^2)$ células da fita. Como $k = O(f(n))$, e o número de passos a serem simulados é $f(n)$, $g(n) = O(f(n)^3)$.

De RAM para TM em Tempo Polinomial

- Lembre-se da conversão de uma TM multi-fita em uma TM 1-fita. Convertemos primeiro p/ uma TM multi-trilha. A TM multi-trilha precisa andar p/ frente e p/ trás ao longo de toda a fita para ler as células ativas. Portanto, no passo k da TM multi-fita, o conteúdo é $O(k)$ e andar p/ frente e p/ trás consome $O(k)$. Como $k = O(g(n))$, e são $g(n)$ passos a simular, $h(n) = O(g(n)^2)$.
- Agora vamos da TM multi-trilha para a TM 1-fita. Lembre-se que tudo o que fizemos foi renomear o alfabeto e as transições. Portanto o tempo de execução no simulador de 1-fita é idêntico ao tempo de execução da TM de k-trilhas simulada!

RAM's são Polinomialmente Equivalentes a TM's

A outra direção também é válida: Podemos simular qualquer TM em Java, com a mesma complexidade de tempo.

Conclusão: A pertinência de uma linguagem na classe **P** não depende do dispositivo de computação escolhido.

Portanto, a análise de complexidade pode ser feita sobre pseudocódigo.

Exemplo de problema em **P**

- Seja *G* um grafo direcionado que contém os nodos *s* e *t.* O problema CAM={<*G,s,t*>|*G* é um grafo direcionado que tem um caminho de *s* para *t*}

Um algoritmo de tempo polinomial para CAM

M="Sobre a entrada <*G,s,t*>, onde G é um grafo direcionado com nodos *s* e *t*:

1. Ponha uma marca sobre o nó *s.*
2. Repita o seguinte até que nenhum nó adicional seja marcado:
   1. Faça uma varredura em todas as arestas de G. Se uma aresta *(a,b)* for encontrada indo de um nó marcado *a* para um nó não marcado *b*, marque o nó *b.*
3. Se *t* estiver marcado, aceite. Caso contrário, rejeite."

Exemplo de problema em P

- Primos entre si. Dois números são primos entre si se 1 é o maior inteiro que divide ambos.
- PRIM-ES={<*x*,*y*> | x e y são primos entre si}
- Resolvemos usando o algoritmo euclidiano.

PRIM-ES $\in$ P

E = “Sobre a entrada <x,y>, onde x e y são número naturais em binário:

1. Repita até y=0:
    1. Atribua $x \leftarrow x$ mod y
    2. Intercambie x e y.
2. Dê como saída x.”

R = “Sobre a entrada $<x,y>$, onde x e y são números naturais em binário.

1. Rode E sobre $<x,y>$.
2. Se o resultado for 1, aceite. Caso contrário, rejeite.”

CFL $\subseteq$ **P**

Vamos mostrar que toda linguagem livre de contexto admite uma solução polinomial para seu problema de aceitação.

NOTA: Como a complexidade do algoritmo é medida em termos dos strings testados, e não da representação da linguagem livre de contexto, podemos supor qualquer forma de representação que nos seja conveniente! Portanto, vamos supor que a linguagem é representada por uma *gramática na forma normal de Chomsky.*

Algoritmo CYK: Decisão em Linguagens Livres de Contexto

IDÉIA: Para cada substring de uma dada entrada *x*, encontre todas as variáveis que podem derivar esse substring. Uma vez que essas tenham sido encontradas, dizer que variáveis podem gerar *x* é uma tarefa simples, já que a gramática está na forma normal de Chomsky.

Algoritmo CYK: Decisão em Linguagens Livres de Contexto

Q: Considere a gramática *G* dada por

$S \rightarrow \varepsilon \mid AB \mid XB$

$T \rightarrow AB \mid XB$

$X \rightarrow AT$

$A \rightarrow a$

$B \rightarrow b$

1. $x = aaabb$ está em $L(G)$?
2. $x = aaabbb$ está em $L(G)$?

Algoritmo CYK: Decisão em Linguagens Livres de Contexto

O algoritmo é “bottom-up”: começamos das folhas para a raiz da árvore de derivação.

a	*a*	*a*	*b*	*b*

$S \rightarrow \varepsilon \mid AB \mid XB$

$T \rightarrow AB \mid XB$

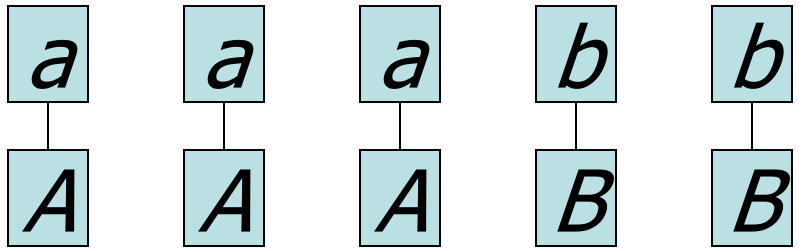
$X \rightarrow AT$

$A \rightarrow a$

$B \rightarrow b$

Algoritmo CYK: Decisão em Linguagens Livres de Contexto

1) Escreva as variáveis para todos os substrings de comprimento 1



$S \rightarrow \varepsilon \mid AB \mid XB$

$T \rightarrow AB \mid XB$

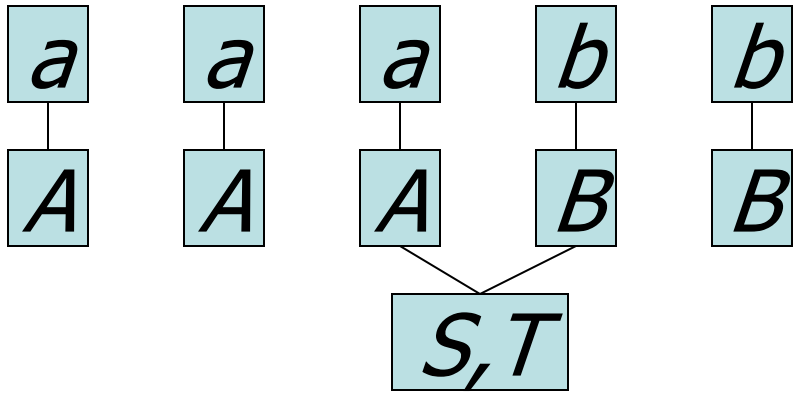
$X \rightarrow AT$

$A \rightarrow a$

$B \rightarrow b$

Algoritmo CYK: Decisão em Linguagens Livres de Contexto

2) Escreva as variáveis para todos os substrings de comprimento 2



$S \rightarrow \varepsilon \mid AB \mid XB$

$T \rightarrow AB \mid XB$

$X \rightarrow AT$

$A \rightarrow a$

$B \rightarrow b$

Algoritmo CYK: Decisão em Linguagens Livres de Contexto

3) Escreva as variáveis para todos os substrings de comprimento 3

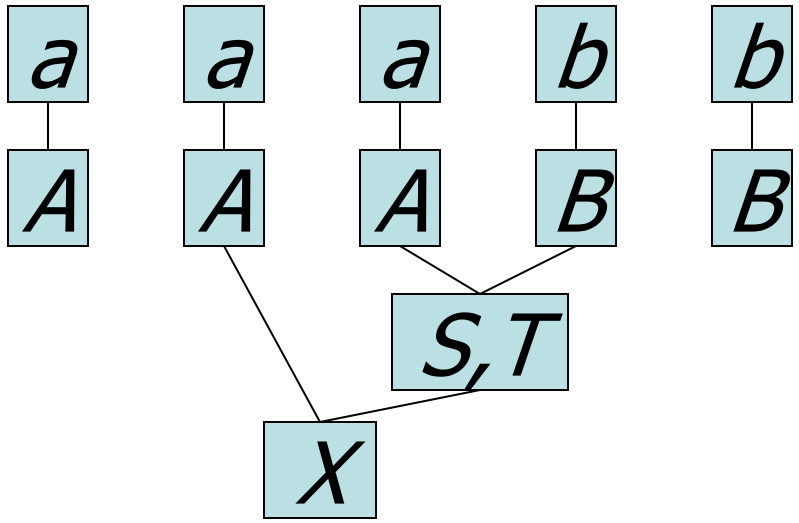
$S \rightarrow \varepsilon \mid AB \mid XB$

$T \rightarrow AB \mid XB$

$X \rightarrow AT$

$A \rightarrow a$

$B \rightarrow b$



Algoritmo CYK: Decisão em Linguagens Livres de Contexto

4) Escreva as variáveis para todos os substrings de comprimento 4

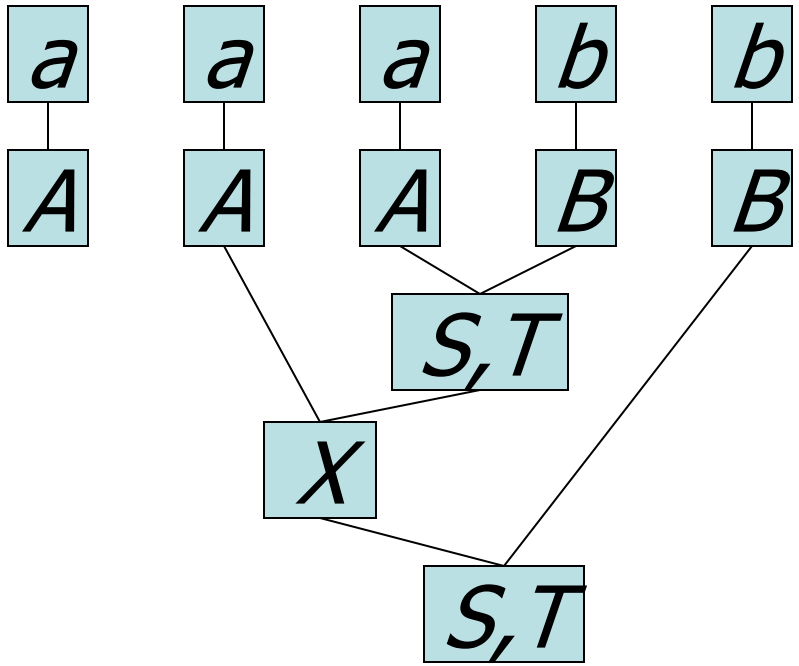
$S \rightarrow \varepsilon \mid AB \mid XB$

$T \rightarrow AB \mid XB$

$X \rightarrow AT$

$A \rightarrow a$

$B \rightarrow b$



Algoritmo CYK: Decisão em Linguagens Livres de Contexto

5) Escreva as variáveis para todos os substrings de comprimento 4. Apenas *X* !

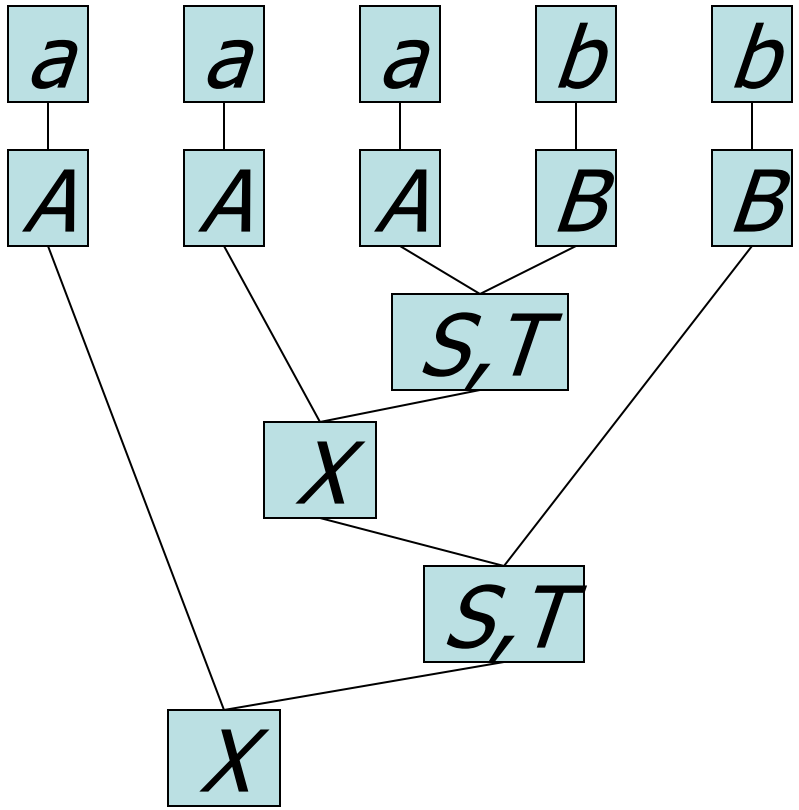
$S \rightarrow \varepsilon \mid AB \mid XB$

$T \rightarrow AB \mid XB$

$X \rightarrow AT$

$A \rightarrow a$

$B \rightarrow b$



REJEITA!

Algoritmo CYK: Decisão em Linguagens Livres de Contexto

Vejamos agora *aaabbb* :

$S \rightarrow \varepsilon \mid AB \mid XB$

$T \rightarrow AB \mid XB$

$X \rightarrow AT$

$A \rightarrow a$

$B \rightarrow b$

a	*a*	*a*	*b*	*b*	*b*

Algoritmo CYK: Decisão em Linguagens Livres de Contexto

1) Escreva as variáveis para todos os substrings de comprimento 1

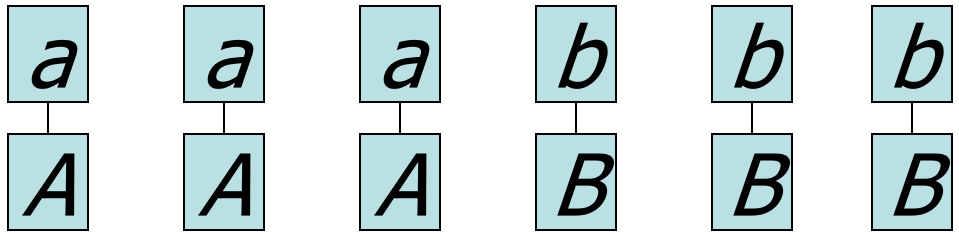
$S \rightarrow \varepsilon \mid AB \mid XB$

$T \rightarrow AB \mid XB$

$X \rightarrow AT$

$A \rightarrow a$

$B \rightarrow b$



Algoritmo CYK: Decisão em Linguagens Livres de Contexto

2) Escreva as variáveis para todos os substrings de comprimento 2

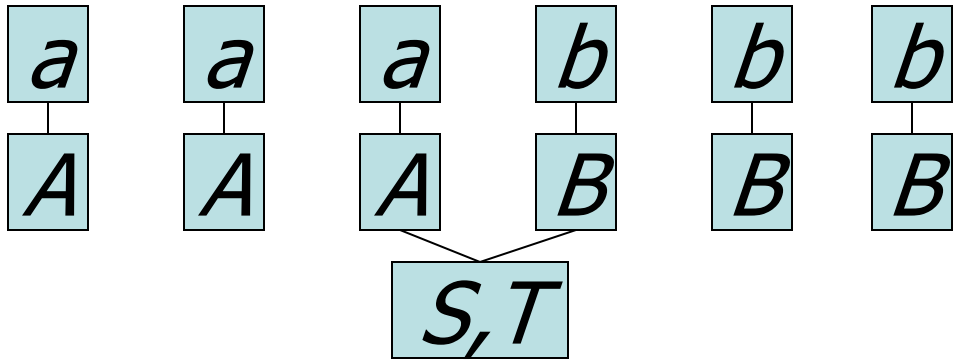
$S \rightarrow \varepsilon \mid AB \mid XB$

$T \rightarrow AB \mid XB$

$X \rightarrow AT$

$A \rightarrow a$

$B \rightarrow b$



Algoritmo CYK: Decisão em Linguagens Livres de Contexto

3) Escreva as variáveis para todos os substrings de comprimento 3

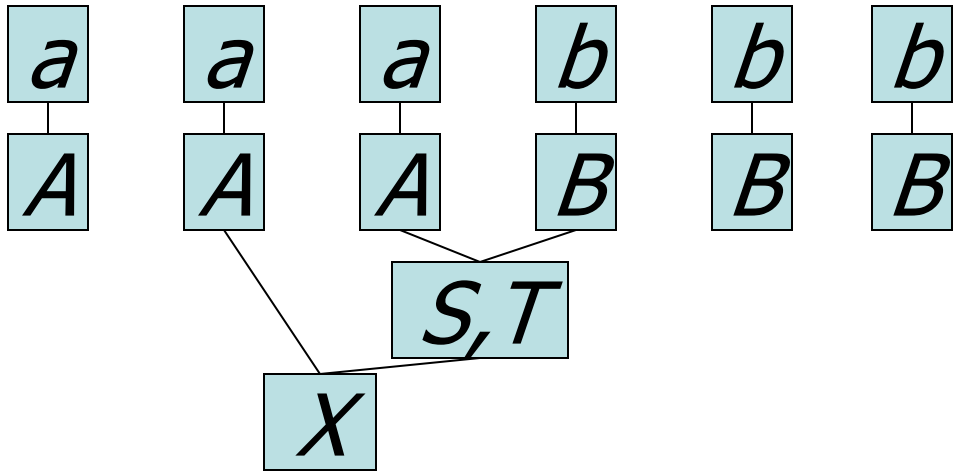
$S \rightarrow \varepsilon \mid AB \mid XB$

$T \rightarrow AB \mid XB$

$X \rightarrow AT$

$A \rightarrow a$

$B \rightarrow b$



Algoritmo CYK: Decisão em Linguagens Livres de Contexto

4) Escreva as variáveis para todos os substrings de comprimento 4

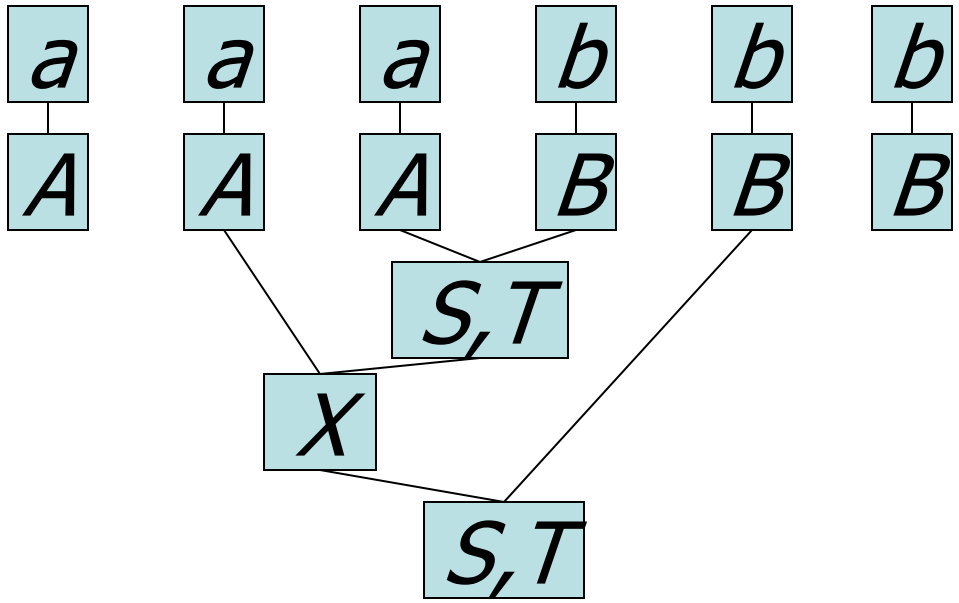
$S \rightarrow \varepsilon \mid AB \mid XB$

$T \rightarrow AB \mid XB$

$X \rightarrow AT$

$A \rightarrow a$

$B \rightarrow b$



Algoritmo CYK: Decisão em Linguagens Livres de Contexto

5) Escreva as variáveis para todos os substrings de comprimento 5

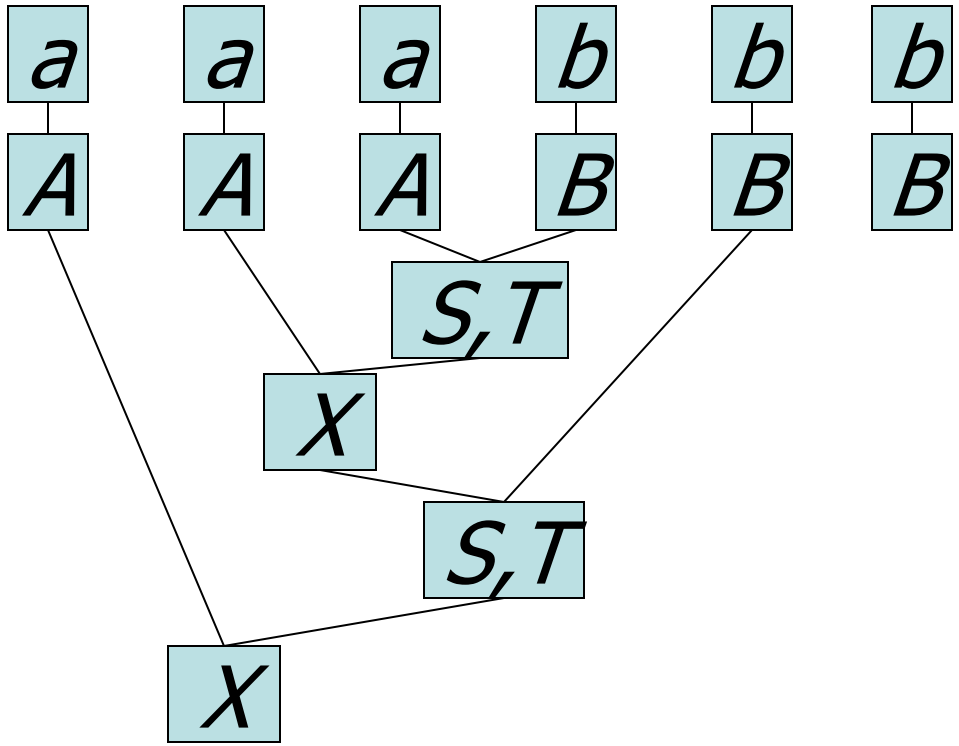
$S \rightarrow \varepsilon \mid AB \mid XB$

$T \rightarrow AB \mid XB$

$X \rightarrow AT$

$A \rightarrow a$

$B \rightarrow b$



Algoritmo CYK: Decisão em Linguagens Livres de Contexto

6) Escreva as variáveis para todos os substrings de comprimento 6

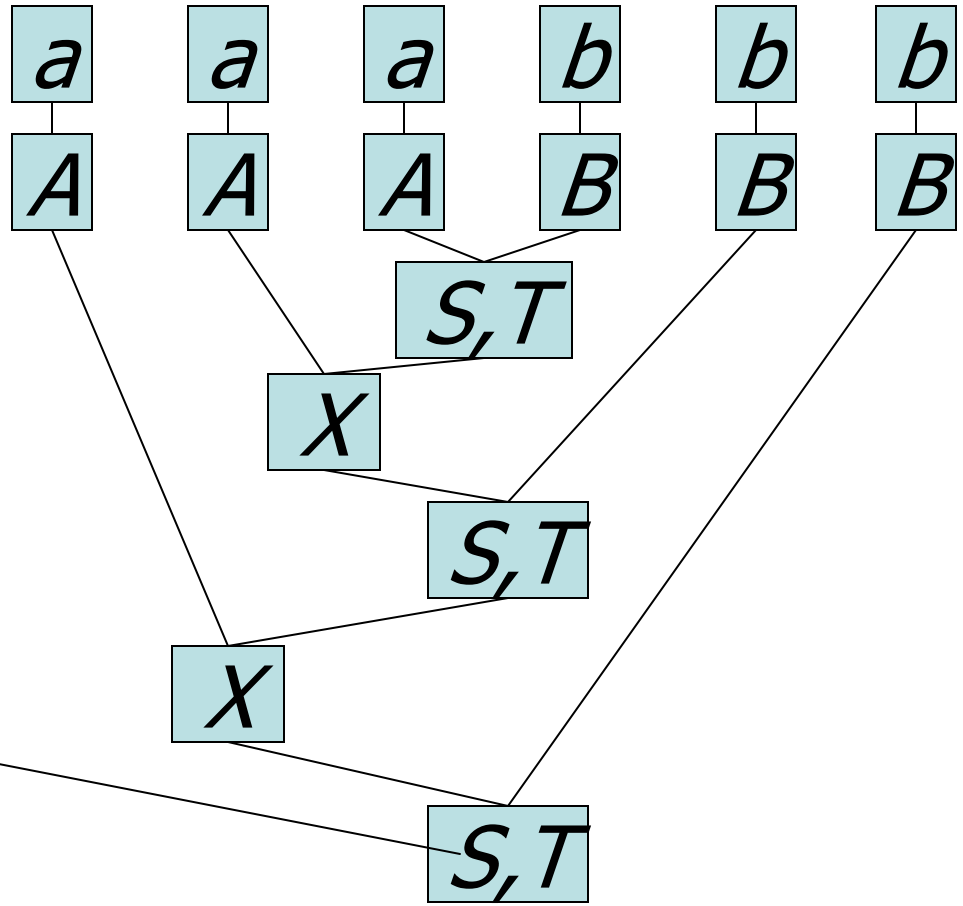
$S \rightarrow \varepsilon \mid AB \mid XB$

$T \rightarrow AB \mid XB$

$X \rightarrow AT$

$A \rightarrow a$

$B \rightarrow b$



S está incluído,

aaabbb aceito!

Algoritmo CYK: Decisão em Linguagens Livres de Contexto

Podemos usar uma tabela para isso.

end at / start at	1: aaabbb	2: aaabbb	3: aaabbb	4: aaabbb	5: aaabbb	6: aaabbb
0:aaabbb						
1:aaabbb						
2:aaabbb						
3:aaabbb						
4:aaabbb						
5:aaabbb						

Algoritmo CYK: Decisão em Linguagens Livres de Contexto

1. Variáveis para substrings de comp. 1.

end at / start at	1: aaabbb	2: aaabbb	3: aaabbb	4: aaabbb	5: aaabbb	6: aaabbb
0:aaabbb	*A*					
1:aaabbb		*A*				
2:aaabbb			*A*			
3:aaabbb				*B*		
4:aaabbb					*B*	
5:aaabbb						*B*

Algoritmo CYK: Decisão em Linguagens Livres de Contexto

2. Variáveis para substrings de comp. 2.

end at / start at	1: aaabbb	2: aaabbb	3: aaabbb	4: aaabbb	5: aaabbb	6: aaabbb
0:aaabbb	*A*	-				
1:aaabbb		*A*	-			
2:aaabbb			*A*	*S,T*		
3:aaabbb				*B*	-	
4:aaabbb					*B*	-
5:aaabbb						*B*

Algoritmo CYK: Decisão em Linguagens Livres de Contexto

3. Variáveis para substrings de comp. 3

end at / start at	1: aaabbb	2: aaabbb	3: aaabbb	4: aaabbb	5: aaabbb	6: aaabbb
0:aaabbb	*A*	-	-			
1:aaabbb		*A*	-	*X*		
2:aaabbb			*A*	*S,T*	-	
3:aaabbb				*B*	-	-
4:aaabbb					*B*	-
5:aaabbb						*B*

Algoritmo CYK: Decisão em Linguagens Livres de Contexto

4. Variáveis para substrings de comp. 4

end at / start at	1: aaabbb	2: aaabbb	3: aaabbb	4: aaabbb	5: aaabbb	6: aaabbb
0:aaabbb	*A*	-	-	-		
1:aaabbb		*A*	-	*X*	*S,T*	
2:aaabbb			*A*	*S,T*	-	-
3:aaabbb				*B*	-	-
4:aaabbb					*B*	-
5:aaabbb						*B*

Algoritmo CYK: Decisão em Linguagens Livres de Contexto

5. Variáveis para substrings de comp. 5

end at / start at	1: aaabbb	2: aaabbb	3: aaabbb	4: aaabbb	5: aaabbb	6: aaabbb
0:aaabbb	*A*	-	-	-	*X*	
1:aaabbb		*A*	-	*X*	*S,T*	-
2:aaabbb			*A*	*S,T*	-	-
3:aaabbb				*B*	-	-
4:aaabbb					*B*	-
5:aaabbb						*B*

Algoritmo CYK: Decisão em Linguagens Livres de Contexto

6. Variáveis para *aaabbb*. ACEITO!

end at / start at	1: aaabbb	2: aaabbb	3: aaabbb	4: aaabbb	5: aaabbb	6: aaabbb
0:aaabbb	*A*	-	-	-	*X*	*S,T*
1:aaabbb		*A*	-	*X*	*S,T*	-
2:aaabbb			*A*	*S,T*	-	-
3:aaabbb				*B*	-	-
4:aaabbb					*B*	-
5:aaabbb						*B*

CYK Pseudocódigo

D="Sobre a entrada $w=w_1w_2...w_n$:

1. Se $w=\varepsilon$ e $S\rightarrow\varepsilon$ for uma regra, aceite.
2. Para *i=1* até *n*:
    1. Para cada variável *A*:
        1. Teste se $A\rightarrow b$ é uma regra, onde $b=w_i$.
        2. Se for, coloque *A* em *tabela(i,i)*.
3. Para *l=2* até *n*:
    1. Para *i=1* até *n-l+1*:
        1. Faça *j=i+l+1.*
        2. Para *k=i* até *j-1*
            1. Para cada regra $A\rightarrow BC$;
                1. Se *tabela(i,k)* contém *B* e *tabela(k+1,j)* contém *C*, ponha *A* em *tabela(i,j)*.
4. Se *S* estive em tabela(1,n), *aceite*. Caso contrário, *rejeite.*