Antônio Magno Lima Espeschit

CT-200 Autômatas e Linguagens Formais

Segunda Lista de Exercícios - 10/JUN/2008

Exercício 5 (Sudkamp Pág. 114). *Let G be the grammar:* $S \rightarrow aS \,|\, Sb \,|\, ab \,|\, SS$.

a) Give a regular expression for $L(G)$.

b) Give 2 leftmost derivations for 'aabb'.

c) Build the derivation trees for (b).

d) Construct an unambiguous grammar equivalent to G.

Solução:

a) Aplicando n vezes a regra $S \rightarrow SS$ obtemos uma cadeia na forma $(SS)^n$.

Usando as demais regras, podemos substituir estes S's por um ou mais a's seguidos por um ou mais b's. Ou seja, iremos obter a composição de várias cadeias na forma $\mathbf{aa^*bb^*}$.

Portanto, iremos obter uma cadeia na forma $(\mathbf{aa^*bb^*})^+$.

Logo, a expressão regular para L(G) é dada por: $(\mathbf{a^+b^+})^+$.

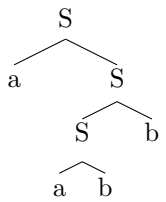
b) Primeira derivação mais à esquerda de 'aabb':

$aS \quad (S \rightarrow aS)$
$aSb \quad (S \rightarrow Sb)$
$aabb \quad (S \rightarrow ab)$

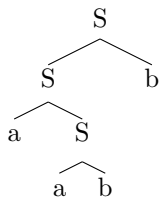
Segunda derivação mais à esquerda de 'aabb':

$Sb \quad (S \rightarrow Sb)$
$aSb \quad (S \rightarrow aS)$
$aabb \quad (S \rightarrow ab)$

c) Árvore da primeira derivação mais à esquerda de 'aabb':



Árvore da segunda derivação mais à esquerda de 'aabb':



d) Temos que criar um grupo com um ou mais a's seguidos de um ou mais b's, repetindo este processo uma ou mais vezes.

Com a produção $S \to aS|aA$ produzimos quantos a's desejarmos e depois vamos para a produção $A \to bA|bS|b$ onde produzimos quantos b's desejarmos podendo voltar para a primeira produção via $S \to bS$ ou encerrar via $S \to b$.

Logo, a gramática G é dada por:

$$S \to aS|aA$$
$$A \to bA|bS|b$$

Exercício 6 (Sudkamp). *Let G be the grammar:*

$$S \to ASB|ab|SS$$
$$A \to aA|\epsilon$$
$$B \to bB|\epsilon$$

a) Give a leftmost derivation for 'aaabb'.

b) Give a rightmost derivation for 'aaabb'.

c) Show that G is ambiguous.

d) Construct an unambiguous grammar equivalent to G.

Solução:

a) Uma derivação mais à esquerda para 'aaabb':

ASB	$(S \to ASB)$
$aASB$	$(A \to aA)$
$aaASB$	$(A \to aA)$
$aaSB$	$(A \to \epsilon)$
$aaabB$	$(S \to ab)$
$aaabbB$	$(B \to bB)$
$aaabb$	$(B \to \epsilon)$

b) Uma derivação mais à direita para 'aaabb':

ASB	$(S \to ASB)$
$ASbB$	$(B \to bB)$
ASb	$(B \to \epsilon)$
$Aabb$	$(S \to ab)$
$aAabb$	$(A \to aA)$
$aaAabb$	$(A \to aA)$
$aaabb$	$(A \to \epsilon)$

c) Para mostrar que G é ambígua, basta mostrar que há pelo menos mais uma derivação à esquerda para 'aaabb'. De fato:

ASB	$(S \to ASB)$
$aASB$	$(A \to aA)$
$aaASB$	$(A \to aA)$
$aaSB$	$(A \to \epsilon)$
$aaabB$	$(S \to ab)$
$aaabbB$	$(B \to bB)$
$aaabb$	$(B \to \epsilon)$

d) A linguagem L(G) é equivalente à expressão regular $\mathbf{a^+b^+}$ a qual pode ser gerada de forma não ambígüa pela gramática:

$$S \rightarrow aS|A$$
$$A \rightarrow bA|\epsilon$$

Exercício 8 (Sudkamp). *Show that $S \rightarrow aaS|aaaaaS|\epsilon$ is ambiguous. Give an unambiguous grammar that generates $L(G)$.*

Solução:

A cadeia $a^{10} \in L(G)$ pode ser gerada de duas formas:

i) Aplicando-se 5 vezes a regra $S \rightarrow aaS$ e, em seguida, a regra $S \rightarrow \epsilon$ ou

ii) Aplicando-se 2 vezes a regra $S \rightarrow aaaaaS$ e, em seguida, a regra $S \rightarrow \epsilon$

Logo, como há mais de uma derivação à esquerda para representar uma única cadeia, a gramática é ambígüa.

Esta gramática gera cadeias $x = a^{(2*k_1+5*k_2)}$ com $\{k_1, k_2\} \subset \mathbb{N}$. A ambigüidade surge porque para gerar uma cadeia de comprimento 10 podemos ter $(k_1 = 5, k_2 = 0)$ ou $(k_1 = 0, k_2 = 2)$. Uma forma de evitar esta redundância é redefinir o comprimento na forma $2 * k_1 + 5 * k_2 = 2 * k_1 + (2 * k_3 + k_4) * 5$ onde k_4 vale 0 ou 1. Deste modo, $2 * k_1 + (2 * k_3 + k_4) * 5 = 2 * (k_1 + 5 * k_3) + 5 * k_4 = 2 * k_5 + 5 * k_4$.

Portanto, podemos ter $L(G) = (\mathbf{aa})^*(\mathbf{aaaaa} + \epsilon)$.

Logo, uma gramática não ambígüa equivalente à G é dada por:

$$S \rightarrow aaS|\epsilon|aaaaa$$

Exercício 25 (Sudkamp). *Let G be:*

$$S \rightarrow aS|AB$$
$$A \rightarrow bAa|a$$
$$B \rightarrow bB|b$$

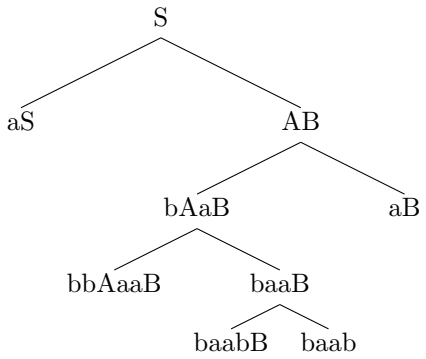
a) Descreva a evolução do conteúdo da fila para um parsing top-down depth-first de 'baab'.

b) Give the tree built by the bread-first bottom-up parse at 'baab'.

c) Descreva a evolução do conteúdo da fila para um parsing bottom-up depth-first de 'baab'.

Solução:

a) A árvore de busca top-down depth-first de 'baab' é dada por:



Portanto, o conteúdo da fila é dado por:

$\emptyset \Rightarrow S \Rightarrow aS|AB \Rightarrow AB \Rightarrow bAaB|aB \Rightarrow bAaB \Rightarrow bbAaaB|baaB \Rightarrow baaB \Rightarrow baabB|baab \Rightarrow baab$

b) Para fazer o processo bottom-up partimos da cadeia 'baab' e fazemos reduções até chegar ao nó S.

Redução	Regra
baab	
baaB	$B \Rightarrow b$
bAaB	$A \Rightarrow a$
AB	$A \Rightarrow bAa$
S	$S \Rightarrow AB$

c) A partir da tabela acima obtemos o conteúdo da fila.

$\emptyset \Rightarrow baab \Rightarrow baaB \Rightarrow bAaB \Rightarrow AB \Rightarrow S$

Exercício 26 (Sudkamp). *Let G be:*

$$S \rightarrow A|AB$$
$$A \rightarrow abA|b$$
$$B \rightarrow baB|a$$

a) Give a regular expression for $L(G)$.

b) Descreva a evolução do conteúdo da fila para um parsing top-down depth-first de 'abbbaa'.

c) Give the tree built by the bread-first bottom-up parse at 'abbbaa'.

d) Descreva a evolução do conteúdo da fila para um parsing bottom-up depth-first de 'abbbaa'.

Solução:

a) As produções $S \rightarrow A$ e $A \rightarrow b$ geram a cadeia $\mathbf{b}$.

As produções $S \rightarrow A$, $A \rightarrow abA$ e $A \rightarrow b$ geram a cadeias da forma $(\mathbf{ab})^+\mathbf{b}$.

Logo, com as produções até agora analisadas, a gramática gera cadeias na forma $(\mathbf{ab})^*\mathbf{b}$.

As produções $B \rightarrow baB$ e $B \rightarrow a$ geram fragmentos de cadeia na forma $(\mathbf{ba})^+\mathbf{a}$. E, pela produção $S \rightarrow AB$ estes fragmentos podem se juntar às cadeias $(\mathbf{ab})^*\mathbf{b}$ acima analisadas.

Logo a expressão regular de $L(G)$ é: $(\mathbf{ab})^*\mathbf{b}(\epsilon + (\mathbf{ba})^+\mathbf{a})$

b) Em cada etapa, expandimos o conteúdo das variáveis S, A e B. Na etapa seguinte, removemos as cadeias cujo prefixo seja incompatível com a cadeia 'abbbaa'. Prosseguimos até não podermos mais expandir variáveis ou até obtermos a cadeia 'abbbaa':

$\emptyset \Rightarrow S \Rightarrow A|AB \Rightarrow abA|b|abAbaB|abAa|bbaB|ba \Rightarrow abA|abAbaB|abAa$

$\Rightarrow ababA|abb|ababAbabaB|abbbaa|ababAbabaB|abbbaa|ababAa|abba \Rightarrow abbbaa$

c) Para fazer o processo bottom-up partimos da cadeia 'abbbaa' e fazemos reduções até chegar ao nó S.

Redução	Regra
abbbaa	
abbbaB	$B \Rightarrow a$
abAbaB	$A \Rightarrow b$
AbaB	$A \Rightarrow abA$
AB	$B \Rightarrow baB$
S	$S \Rightarrow AB$

d) Descreva a evolução do conteúdo da fila para um parsing bottom-up depth-first de 'abbbaa'.

$\emptyset \Rightarrow abbbaa \Rightarrow abbbaB \Rightarrow abAbaB \Rightarrow AbaB \Rightarrow AB \Rightarrow S$

Exercício 2.4 (Sipser Pág. 120). *Give the CFG's that generate the following languages. Consider* $\Sigma = \{0, 1\}$.

a) $\{w|w$ *contains at least three* 1*'s*$\}$.

b) $\{w|w$ *starts and ends with the same symbol*$\}$.

c) $\{w|$*length(w) is odd*$\}$.

d) $\{w|$*length(w) is odd and it's middle symbol is* $0\}$.

e) $\{w|w$ *contain more* 1*'s than* 0*'s*$\}$.

f) $\{w|w = w^R\}$.

g) The empty set.

Solução:

a) $\{w|w$ contém pelo menos três 1's$\}$.

$$S \rightarrow A1A1A1A$$
$$A \rightarrow 0A|1A|\epsilon$$

b) $\{w|w$ começa e termina com o mesmo símbolo$\}$.

$$S \rightarrow 0|1|0A0|1A1$$
$$A \rightarrow 0A|1A|\epsilon$$

c) $\{w|$comprimento de w é ímpar$\}$.

$$S \rightarrow 0A|1A$$
$$A \rightarrow 00A|01A|10A|11A|\epsilon$$

d) $\{w|$comprimento de w é ímpar e o símbolo do meio é 0$\}$.

$$S \rightarrow 0|ASA$$
$$A \rightarrow 0|1$$

e) $\{w|w$ contém mais 1's do que 0's$\}$.

Usando o JFlap para criar um PDA, convertendo o PDA para uma gramática equivalente e simplificando a gramática, obtemos:

$$S \rightarrow SAB|BAB$$
$$A \rightarrow 1A|1$$
$$B \rightarrow B0B1|B1B0|\epsilon$$

f) $\{w|w = w^R\}$.

$$S \rightarrow \epsilon|0|1|0S0|1S1$$

g) O conjunto vazio.

$$S \rightarrow S$$

Esta gramática não tem terminais. Logo, nunca é possível gerar uma cadeia. Portanto, $L(G) = \emptyset$.

Exercício 2.5 (Sipser). *Give the informal descriptions and state diagrams of pushdown automata for the languages in Exercise 2.4.*

Solução:

a) $\{w|w$ contém pelo menos três 1's$\}$.

As cadeias desta linguagem são da forma $A1A1A1A$ onde A pode valer $0, 1$ ou ϵ. Deste modo, garante-se que é possível gerar qualquer número formado por 0's e 1's garantindo-se que haja pelo menos três 1's.

O Autômato de pilha aqui é na verdade um autômato finito comum que, a partir de um estado inicial q_0, nele permanece enquanto estiver lendo 0's e muda para o estado q_1 se ler um dígito 1. Em seguida, permanece em q_1 enquanto estiver lendo 0's e transita para o estado q_{11} se ler um dígito 1. Em seguida, permanece em q_{11} enquanto estiver lendo 0's e transita para o único estado de aceitação q_{111} se ler um dígito 1, permanecendo neste estado enquanto houver caracteres para ler na cadeia de entrada.

b) $\{w|w$ começa e termina com o mesmo símbolo$\}$.

O PDA da Figura 1 não aceita a cadeia vazia. Logo, o estado inicial q_0 não é um estado de aceitação.

Quando lê o primeiro caracter da cadeia, coloca na pilha um valor idêntico ao caracter lido.

A seguir, vai lendo os demais caracteres sem colocar nem retirar nada da pilha. Em cada passo, tenta "adivinhar" se o fim da cadeia foi atingido, retirando um símbolo da pilha e verificando se o mesmo é igual ao símbolo lido na cadeia de entrada no momento em que leu o último caractere.

c) $\{w|$comprimento de w é ímpar$\}$.

d) $\{w|$comprimento de w é ímpar e o símbolo do meio é 0$\}$.

e) $\{w|w$ contém mais 1's do que 0's$\}$.

Usando o JFlap para criar um PDA, convertendo o PDA para uma gramática equivalente e simplificando a gramática, obtemos:

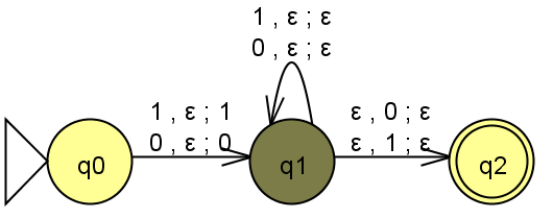


Figura 1 PDA que reconhece $L = \{w|w$ começa e termina com o mesmo símbolo$\}$.

$$S \rightarrow SAB|BAB$$
$$A \rightarrow 1A|1$$
$$B \rightarrow B0B1|B1B0|\epsilon$$

f) $\{w|w = w^R\}$.

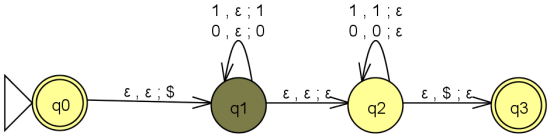


Figura 2 PDA que reconhece $L = \{w|w = w^R\}$.

O PDA da Figura 2 aceita a cadeia vazia. Logo, o estado inicial q_0 é um estado de aceitação.

À medida em que lê a cadeia de entrada, copia os símbolos para a pilha.

Em cada passo, tenta "adivinhar" se o meio da cadeia foi atingido, retirando um símbolo da pilha e verificando se o mesmo é igual ao símbolo lido na cadeia de entrada.

Se a cadeia de entrada for um palíndromo, então a pilha se esvazia no momento em que a leitura da cadeia termina e o PDA transita para o estado final q_3.

g) O conjunto vazio.

É um PDA sem nenhum estado de aceitação. Portanto, não há como reconhecer nenhuma cadeia.

Exercício 2.18 (Sipser 121). *Use the Pumping Lemma to show that the following languages are not context free.*

a) -.

b) $\{0^n\#0^{2n}\#0^{3n}|n \geq 0\}$.

c) $\{w\#x|w$ *is a substring of* x *where* $w, x \in \{a, b\}^*\}$.

d) $\{x_1\#x_2\#\ldots\#x_k|k \geq 2\, x_i \in \{a, b\}^* \wedge \exists i \neq j|x_i = x_j\}$.

Solução:

Se L é regular então existe uma constante p tal que se $z \in L$ e $|z| \geq p$ então, pelo Pumping Lemma para Linguagens Livres de Contexto, podemos escrever $z = uvwxy$ tal que:

i) $|vx| \geq 1$.

ii) $|vwx| \leq p$.

iii) $uv^iwx^iy \in L \forall i \geq 0$.

a) -.

b) $\{0^n\#0^{2n}\#0^{3n} | n \geq 0\}$.

Demonstração. Por contradição.

Vamos supor, por contradição, que L é Livre de Contexto.

Neste caso, nas condições do Pumping Lemma para Linguagens Livres de Contexto, vamos considerar $z = 0^p\#0^{2p}\#0^{3p} \in L$.

Portanto, há duas hipóteses para v e x: ou são formados apenas por caracteres 0 ou por caracteres 0 e um caracter #.

No primeiro caso, a cadeia uv^iwx^iy não pode pertencer a L pois, para que isso fosse possível, cadeia teria que possuir 3 "grupos" de zeros delimitados pelo caracter # e, ao elevarmos v e x a $i > 0$ no máximo 2 destes grupos ganharam novos 0's, causando um desbalanceamento com o terceiro grupo.

No segundo caso, a cadeia uv^iwx^iy também não pode pertencer a L pois possuiria mais do que dois caracteres # e, para pertencer a L, a cadeia teria que ter dois caracteres #.

Devido a esta contradição, temos que L não é Livre de Contexto.

□

c) $\{w\#x | w \text{ is a substring of } x \text{ where } w, x \in \{a, b\}^*\}$.

Demonstração. Por contradição.

Vamos supor, por contradição, que L é Livre de Contexto.

Neste caso, nas condições do Pumping Lemma para Linguagens Livres de Contexto, vamos considerar $z = w\#x = a^pb^p\#a^pb^p \in L$.

Se v e y estão ambos na primeira parte da cadeia, o bombeamento de w com $i > 0$ faz com que o comprimento da cadeia bombeada fique maior que o necessário para que esta seja uma subcadeia de x. Logo, a cadeia bombeada não pertence a L.

Se v e y estão ambos em x (segunda parte da cadeia), o bombeamento com $i = 0$ faz com que a cadeia bombeada fique comprimento muito curto para conter w como subcadeia de x. Logo, a cadeia bombeada não pertence a L.

Pela condição (ii) do Pumping Lemma para Linguagens Livres de Contexto, temos $|vwx| \leq p$. E, conforme já analisamos, nem v e nem y podem conter #. Portanto, v deve possuir b's da primeira parte da cadeia (ou seja, de $w = a^pb^p$) e x deve possuir a's vindos da segunda parte da cadeia (ou seja, de $x = a^pb^p$). Entretanto, quando fizermos o bombeamento com $i > 0$ a primeira parte da cadeia bombeada não pode ser subcadeia da segunda parte pois esta possui mais b's que aquela. Logo, a cadeia bombeada não pertence a L.

Tendo esgotado todas as formas de subdividir $z = w\#x = a^pb^p\#a^pb^p$ sem que seja obtida uma cadeia bombeada que pertença a L, chegamos a uma contradição. Logo, L não é Livre de Contexto.

□

d) $\{x_1\#x_2\#\ldots\#x_k | k \geq 2\, x_i \in \{a,b\}^* \wedge \exists i \neq j | x_i = x_j\}$.

Exercício 2.26 (Sipser). *Let $C = \{x\#y | x, y \in \{0,1\}^* \wedge x \neq y\}$. Show that C is context free.*

Solução:

Demonstração. Por construção.

Se $x \neq y$ então podemos fazer $x = \alpha a \beta$ e $y = \gamma b \delta$ com $\{\alpha, \beta, \gamma, \delta\} \subset \{a,b\}^*$.

Podemos gerar cadeias na forma $x\#y$ com a gramática G:

$$\begin{aligned} S &\rightarrow Ab(a+b)^* \\ A &\rightarrow (a+b)^* A (a+b)^* \\ A &\rightarrow a(a+b)^*\# \end{aligned}$$

Esta gramática é equivalente a:

$$\begin{aligned} S &\rightarrow Ab|Sa|Sb \\ A &\rightarrow aAa|aAb|bAa|bAb|B \\ B &\rightarrow aB|Ba|Bb|\# \end{aligned}$$

Como G é livre de contexto temos que C=L(G) é livre de contexto. □

Exercício 2.27 (Sipser). *Let $D = \{xy | x, y \in \{0,1\}^* \wedge |x| = |y| \, but \, x \neq y\}$. Show that D is context free.*

Solução:

Demonstração. Por construção.

Podemos gerar cadeias na forma xy com $|x| = |y|$ usando a gramática G:

$$\begin{aligned} S &\rightarrow AB|BA \\ A &\rightarrow a|CAC \\ B &\rightarrow b|CBC \\ C &\rightarrow a|b \end{aligned}$$

Como G é livre de contexto, então $D = L(G)$ também é livre de contexto.

□

Exercício 7.1 (Hopcroft Pág. 74). *Design a Turing Machine to recognize the following languages:*

a) $\{0^n 1^n 0^n | n \geq 1\}$.

b) $\{ww^R | w \in \{0,1\}^*\}$.

c) The set of strings of equal number of 0's and 1's.

Solução:

a) $\{0^n1^n0^n | n \geq 1\}$.

Inicialmente, comparo os n primeiros zeros com os n primeiros uns. Para cada zero encontrado, eu o troco por B e depois procuro um 1 e troco por X. Continuo o processo até que todos os 0's tenha se transformado em B e todos os 1's tenham se transformado em X.

Se houver número diferentes de 0's e 1's sobrarão 0's antes dos 1's ou 1's após os X's. Neste caso, a máquina rejeita.

Porém, se o número inicial de zeros for igual ao número de 1's haverá n dígitos $X's$ imediatamente antes dos números zeros finais.

Portanto, eu comparo o número de X's com o número de zeros no fim da fita. Para cada X lido eu o troco por um B e procuro o próximo B. Ao encontrar este B, eu retorno uma casa e tenho que encontrar um zero. Se achar um X, travo (tem mais X's que zeros). Se acho um zero, troco por B e volto até encontrar um X. Se não achar o X, certifico-me de que também não há nenhum 0. Neste caso, termino num estado de aceitação. Se, não havendo nenhum X eu ainda achar um 0, há mais 0's que X e eu travo. Se, houver X, continuo indo para a esquerda até achar B. Quando encontro B, ando uma casa para a direita, encontrando um X e reiniciando o processo.

b) $\{ww^R | w \in \{0,1\}^*\}$.

Considero uma MT de duas fitas.

Leio o primeiro caracter e escrevo na segunda fita. Em seguida, substituo este caracter por B e avanço para a direita até achar B. Achando B, volto um caracter.

Neste ponto, estou no caracter mais à direita da fita. Comparo este caracter com o caracter escrito na fita 2. Se forem diferentes, rejeito. Se forem iguais, substituo o caracter da fita 1 e o caracter da fita 2 por B e retorno para a esquerda até achar um caracter B. Em seguida, ando um caracter para a direita e renicio o ciclo até que não existam mais caracteres na fita.

c) Cadeias com mesmo número de 0's e 1's.

Considero uma MT de três fitas.

Na primeira fita, está a cadeia de 0's e 1's a ser testada.

Na fita 2 guardo o número de 0's e na fita 3 guardo o número de 1's.

Etapa I: Cálculo do número de 0's e de 1's.

Vou lendo a fita 1 da esquerda para a direita. Se acho um 0, escrevo um 0 na fita 2 e ando para a direita nas fitas 1 e 2. Se acho um 1, escrevo um 1 na fita 3 e ando para a direita nas fitas 1 e 3. Se acho um B, terminei a leitura e vou para a próxima etapa, caso contrário, continuo lendo e repetindo o algoritmo de escrita nas fitas 2 (quando acho 0) e 1 (quando acho 1).

Etapa II: Verificação se o número de 0's é igual ao número de 1's.

Quando terminei a leitura da fita 1, volto a cabeça das fitas 2 e 3 para o início. Em seguida, vou lendo as fitas 2 e 3. Para cada zero lido na fita 2 tem que haver um 1 lido na fita 3. Caso contrário, rejeita. Se for lido B simultaneamente nas fitas 2 e 3 então o número de 0's é igual ao número de 1's e eu aceito a cadeia de entrada.

Exercício 7.2 (Hopcroft). *Design a Turing Machine to compute the following functions:*

a) $\lceil \log_2 n \rceil$.

b) $n!$.

c) n^2.

Solução:

a) $\lceil \log_2 n \rceil$.

A idéia é dividirmos n por 2 até obter 1 ou 0 como resultado. Se fizermos m divisões então $m = \lceil \log_2 n \rceil$. Por exemplo, se $n = 8$ então, na primera divisão temo $8/2 = 4$, na segunda divisão temos $4/2 = 2$ e na terceira divisão temos $2/2 = 1$. Como fizemos 3 divisões, temos que $m = \lceil \log_2 8 \rceil = 3$.

Para efetuar as m divisões, consideremos uma MT com duas fitas, sendo que o número n é representado como n zeros na primeira fita enquanto que o número m é representado pelo número de zeros na segunda fita, inicialmente vazia.

Durante a divisão a MT deve ir lendo os caracteres 0's da primeira fita, da esquerda para a direita, mantendo um zero e apagando o próximo até encontrar um caracter em branco. Quando este caracter em branco for encontrado, deverá ser escrito um zero na segunda fita. Neste momento, há $\lceil \frac{n}{2} \rceil$ zeros na primeira fita. Uma subrotina "aglutina" estes zeros eliminando os espaços em branco entre os mesmos e preparando a MT para uma nova iteração.

O processo se repete até que reste apenas um (ou nenhum) zero na primeira fita. Neste momento, o número de zeros na segunda fita representa $m = \lceil \log_2 n \rceil$.

b) $n!$.

Inicialmente, desenvolvemos uma subrotina que multiplica dois números m e n seguindo este raciocínio:

i) Inicialmente, números m e n são representados em "unário" na forma 0^m10^n1.

ii) Para cada um dos m zeros lidos, substituimos este zero por um branco (B) e escrevemos n zeros após o segundo 1. Quando todos os m zeros tiverem sido lidos haverá mn zeros após o segundo 1.

iii) Trocamos os caracteres 10^n por B.

iv) Movemos os mn zeros para o início da fita.

Para calcular $n!$ usamos a rotina de multiplicação acima para multiplicar n por $(n-1)$ e o resultado por $(n-2)$ repetindo o processo k vezes até que $n - k = 1$.

De forma um pouco mais detalhada:

i) Inicialmente, temos o número n representado em "unário" na forma 0^n1.

ii) Copiamos o número n para a direita obtendo 0^n10^n1.

iii) Apagamos o último zero do segundo número obtendo $0^n10^(n-1)1$.

iv) Finalmente, escrevemos 0^n1 no fim da fita. Ou seja, temos os números n, $(n-1)$ e $p = n$ na fita.

v) Se considerarmos estes números como a, b e c podemos ter um loop no qual, em cada iteração: $c = c * b$ e $b = b - 1$. Este loop continua até que $b = 1$.

vi) Ao término do processo, teremos $c = n!$.

c) n^2.

Este algoritmo também usa a rotina de multiplicação acima discutida.

i) Inicialmente, temos o número n representado em "unário" na forma 0^n1.

ii) Copiamos o número n para a direita obtendo 0^n10^n1.

iii) Finalmente, usamos o algoritmo de multiplicação acima descrito para multiplicar $n * n$ obtendo n^2.