# Aula 06


Universidade Federal de Santa Maria
SEDES SAPIENTIAE
1960

# Compressão por entropia

**Código de Huffman**

**Variantes**

**Huffman adaptativo/dinâmico**

Código construído durante transmissão dos dados

**Huffman n-ário**

Árvore de código n-ária ao invés de binária

**Huffman de variância mínima/comprimento limitado**

Código com tamanho máximo pré-estabelecido

Outros

# Métodos de compressão por dicionário

Abraham Lempel e Jacob Ziv

Comprimir strings pode ser mais eficiente do que comprimir símbolos isolados

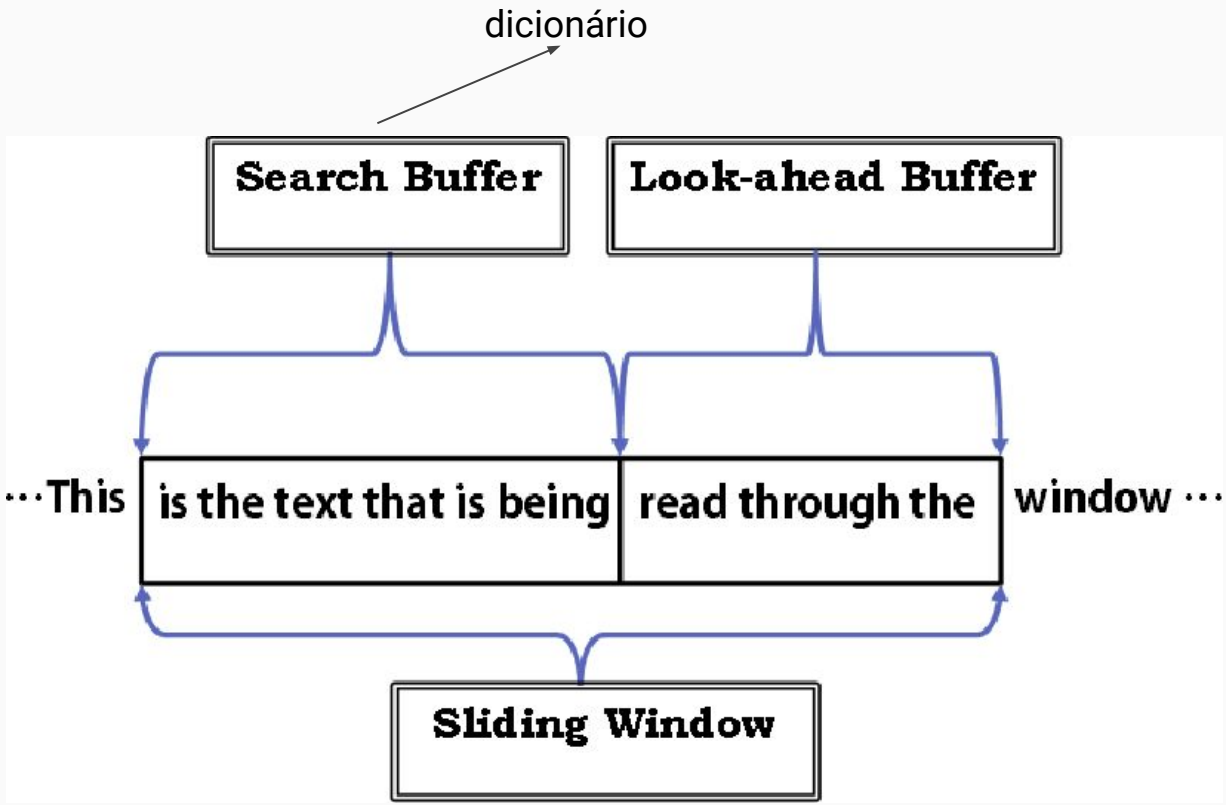
Lempel-Ziv

- LZ77 - janela deslizante (*sliding window*)
- LZ78 - dicionário dinâmico

# LZ77

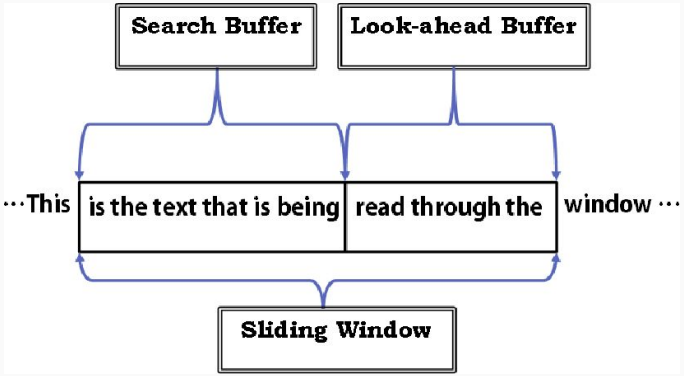
- Baseado no uso de janela deslizante (*sliding window*)
- Algoritmo de uma passada (*1-pass*)
- Naturalmente adaptativo
  - Tamanho da janela (tamanho influi diretamente no tamanho das tuplas)

# LZ77



# LZ77 - algoritmo

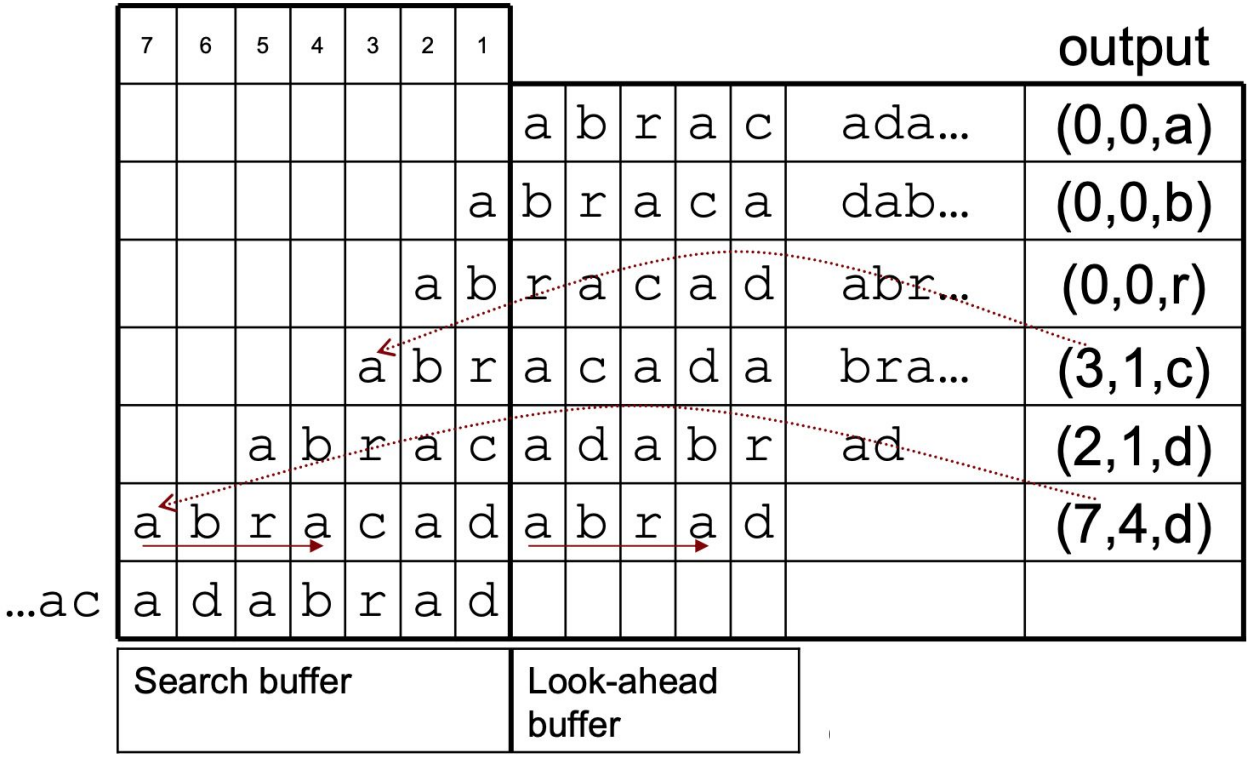
- Enquanto *look-ahead buffer* != NULL
  - Encontrar maior *match* em *search* para a string no início do *look-ahead buffer*
  - Escrever tupla na saída (*offset, length,* símbolo)
  - janela++



# LZ77

Encoding of the string:
abracadabrad

Output Tuple:(offset, length, symbol)




Descompr = 12*8b = 96 bits

**Com alfabeto mínimo:**
Comp = 6*(4+3+3) = 60 bits

Tc = 96b/60b = 1,6
Fc = 60b/96b = 0,625
**OBS**: somar alfabeto depois

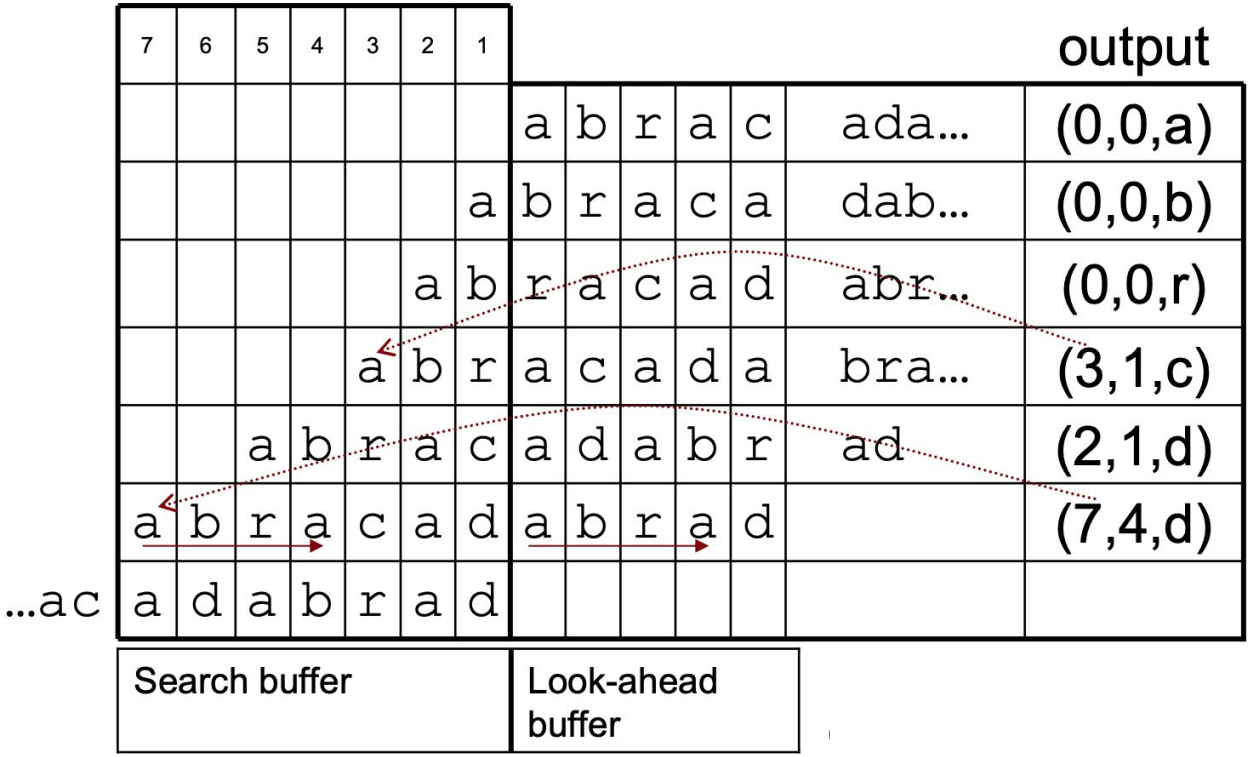
**Com alfabeto ASCII:**
Comp = 6*(4+3+8) = 90 bits

Tc = 96b/90b = 1,06
Fc = 90b/96b = 0,9375

# LZ77

Encoding of the string:
abracadabrad

Output Tuple:(offset, length, symbol)




Descompr = 12*8b = 96 bits

**LZSS (bit de indicação de tupla). Ex. 7bit ASCII:**
Comp = (3*8)+(3*(4+3+7)) = 66b

Tc = 1,45
Fc = 0,6875

# LZ77 - outras variantes

LZS - Símbolo/offset-comprimento codificado com Huffman Ex.: DEFLATE (PKzip, Winzip)

LZ4, LZO, LZRW, Zstandard - velocidade de compressão/descompressão

LZMA - (*Markov chain*) usa dicionários muito grandes e modelo probabilístico

LZX - (*Microsoft cabinet*)

Muitos outros

# LZ78

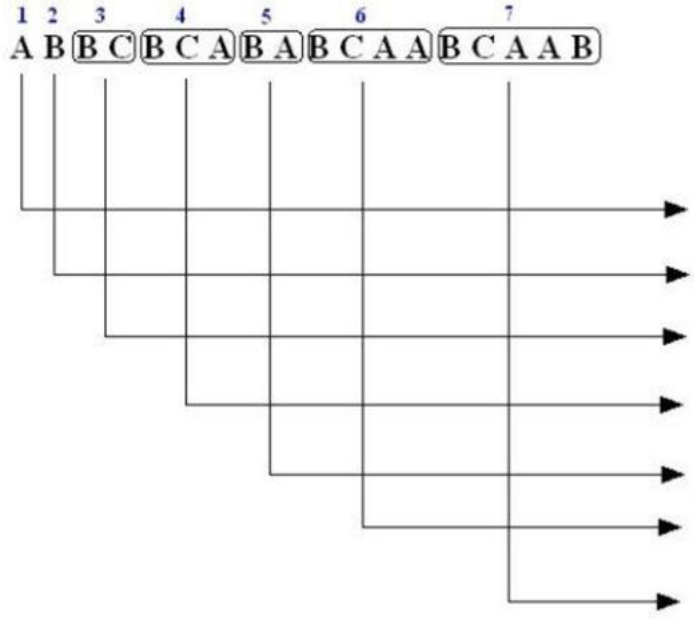
- Criação de dicionário dinâmico incremental
- Algoritmo de uma passada (*1-pass*)
- Naturalmente adaptativo

# LZ78 - Algoritmo

- Ler próximo símbolo na entrada, procurar se está no dicionário
  - Se estiver, atualizar índice do “último match” para a entrada encontrada
  - Se não estiver, escrever (último match, símbolo) na saída, criar nova entrada (se possível) e zera “último match”

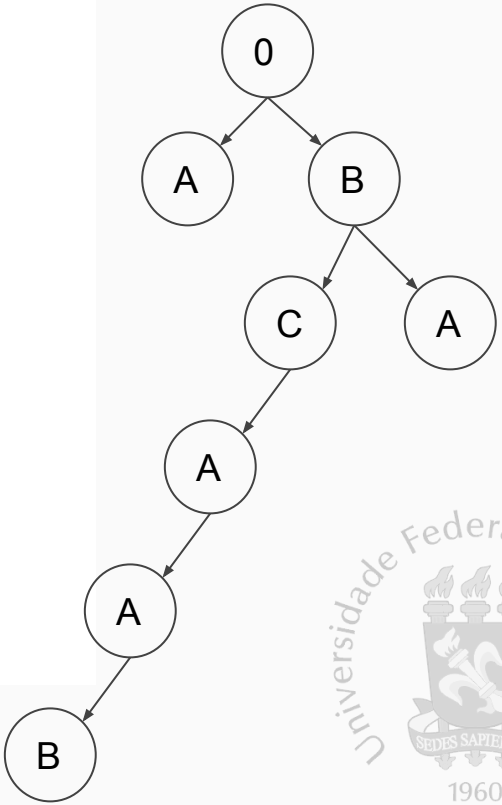
# LZ78

Encode (i.e., compress) the string **ABBCBCABABCAABCAAB** using the LZ78 algorithm.




| | Dictionary | |
|---|---|---|
| output | index | string |
| (0, A) | 1 | A |
| (0, B) | 2 | B |
| (2, C) | 3 | BC |
| (3, A) | 4 | BCA |
| (2, A) | 5 | BA |
| (4, A) | 6 | BCAA |
| (6, B) | 7 | BCAAB |

The compressed message is: **(0,A)(0,B)(2,C)(3,A)(2,A)(4,A)(6,B)**



# LZ78 - uso de memória

Dicionário pode ocupar toda a memória livre

Soluções:

- Congelar o dicionário
- Deletar o dicionário e recomeçá-lo

Ex. : compress (unix) → congela o dicionário e monitora a taxa de compressão.
Se ficar menor que um dado *threshold*, deleta o dicionário e recomeça-o

# LZ78 - Variantes

LZW - Dicionário pré-inicializado

LZWL - variante de LZW baseada em sílabas

BTLZ - modems (v.42bis)

LZT - LZW com dicionário armazenado em lista LRU (*Least Recently Used*)

LZC/LZJ

Outras

# LZ77 x LZ78

| LZ77 | LZ78 |
| --- | --- |
| Usa dados anteriores | Tenta usar dados futuros |
| Mais lento que LZ78 | Requerimentos de espaço para o dicionário em memória |
| Open source. Usado em ZIP, PNG, TIFF, PDF, LHA, gzip, entre outros. | Usado em compress, GIF, ARC, CCITT (modems), PAK, entre outros. |