# TP1 Algoritmos 2

João Luiz Cerqueira 2021031858

Maio 2023

## 1 Árvore Trie

A classe **TrieNode** representa um nó da árvore Trie, contendo dois atributos:

- filho: dicionário que mapeia seus filhos na árvore;
- index: um inteiro que indica o índice da palavra armazenada na árvore.

A classe **Trie** representa a própria árvore Trie e possui os seguintes métodos:

- __init__(): construtor queinicializa a raiz da árvore e o índice atual;
- insert(palavra): insere uma palavra na árvore, percorrendo a árvore a partir da raiz e criando novos nós caso necessário;
- search(palavra): busca uma palavra na árvore, percorrendo a árvore a partir da raiz e retornando o índice da palavra caso ela seja encontrada, ou -1 caso contrário. Esse método serve apenas pra saber se a palavra desejada está na árvore;
- search_output(palavra): busca uma palavra na árvore, percorrendo a árvore a partir da raiz e retornando o índice do último nó percorrido. Esse método serve pra retornar o índice que será colocado na tupla (índice,char) do algoritmo LZ78.

## 2 Algoritmo LZ78

Para implementação do algoritmo LZ78 foram usadas as funções:

### 2.1 compress(inputstr)

A função compress recebe uma string de entrada e retorna uma lista de tuplas que representam a compressão da string de entrada. Cada tupla é composta por dois elementos: o índice da palavra encontrada na Trie (iniciando em 1) e o último caractere da palavra.

A compressão é feita da seguinte forma: a string de entrada é percorrida da esquerda para a direita e, em cada posição, é procurada a maior palavra na Trie que começa nessa posição. A palavra é representada pelo índice correspondente na Trie (adicionando 1, já que os índices na Trie começam em 0) e pelo último caractere da palavra.

A cada palavra encontrada, ela é adicionada à Trie para permitir que palavras subsequentes possam ser encontradas. Quando não há mais palavras a serem encontradas, a compressão é interrompida.

## 2.2 decompress(compressed)

A função 'decompress' implementa a descompressão de uma lista de tuplas retornadas pela função compress. Ela itera sobre cada tupla da lista de entrada e chama a função 'decompress_aux' para descomprimir a tupla em uma string. A função 'decompress_aux' é uma função auxiliar que é recursivamente chamada para descomprimir uma tupla.

A função 'decompress_aux' recebe a lista comprimida e um índice como entrada. Ela verifica se o primeiro elemento da tupla no índice passado é diferente de zero. Se for diferente de zero, significa que essa tupla se refere a outra tupla anterior na lista, então a função chama a si mesma recursivamente para descomprimir essa tupla anterior e concatena o resultado com o segundo elemento da tupla atual, que representa o último caractere da palavra atual. Se o primeiro elemento da tupla atual for zero, a função simplesmente concatena o segundo elemento da tupla atual à palavra atual, pois o índice 0 significa que aquela tupla corresponde apenas ao caractere na sua segunda posição.

## 2.3 write_in_binary(filename,compressed)

Essa função recebe o nome do arquivo desejado para armazenar o texto comprimidoem binário e a lista de tuplas retornadas pela função compressed. Cada tupla na lista é dividida em 2 valores, valor1 e valor2. O valor1 corresponde ao índice da tupla convertido pra representação binária em 3 bytes. O valor2 corresponde ao caractere da tupla, primeiramente convertido em seu código ASCII correspondente e em seguida convertido pra representação binária em 2 bytes. Esses 2 valores são armazenados juntos no output.

## 2.4 read_from_binary(filename, lista_convertida)

Essa função recebe o nome do arquivo binário resultante de write_in_binary e a lista a conter as tuplas na forma original não binária. Em seguida ele faz o caminho contrário. Para cada 4 bytes lidos, o primeiro é convertido à um caractere e os 3 últimos convertidos em um inteiro. Em seguida sao inseridos na juntos como uma tupla na lista. Por fim temos as tuplas prontas para serem descomprimidas.

# 3 Taxas de compressão

- don_casmurro.txt: tamanho original 383.5kb, comprimido 272,4kb. Taxa:29%
- os_luisadas.txt: tamanho original 337KB, comprimido 252KB; Taxa: 25%
- constituicao1988.txt: tamanho original 637Kb comprimido 427Kb; Taxa: 32%
- nietzsche.txt: tamanho original 400Kb comprimido 359Kb Taxa:10%
- karamazov.txt: tamanho original 2 MB comprimido 1.4 MB Taxa:30%
- mobydick.txt: tamanho original 539Kb comprimido 478Kb Taxa:11%
- ulisses.txt: tamanho original 243Kb comprimido 234Kb Taxa:3%
- crime.txt: tamanho original 273Kb comprimido 250Kb Taxa:8%
- shakespeare.txt: tamanho original 613Kb comprimido 505Kb Taxa:17%
- leviatan.txt: tamanho original 1226Kb comprimido 900Kb Taxa:26%

# 4 Conclusoes

Minhas conclusões vem principalmente do uso do python. Fiz a escolha errada da linguagem. Consegui fazer uma compressão muito melhor dos arquivos em C++ infelizmente não terei tempo pra concluir o trabalho em C++. Isso se deu pois é possivel utilizar apenas 1 byte pra representar os caracteres, pois mesmo que o caractere precise de 2 bytes no formato utf-8, é possível ler byte por byte da entrada e mesmo que sejam corrompidos se juntam na descompressão. Logo nao consegui comprimir com menos de 5 bytes e para arquivos pequenos isso se torna ruim.

## 4.1 Atenção

A descompressão demorou 2 minutos para arquivos de 2MB no meu computador.

## 4.2 Execução

Como fiz em python a linha de comando pra executar se torna:
python3 main.py -c nomedoarquivo [-o nome da saida] pra compressão
python3 main.py -x nomedoarquivo [-o nome da saida] pra descompressão
o [-o nome da saida] não é obrigatório em nenhum dos casos