

TP1 Algoritmos 2 - DCC207

Lucas Paulo Martins Mariz

- **Objetivos**

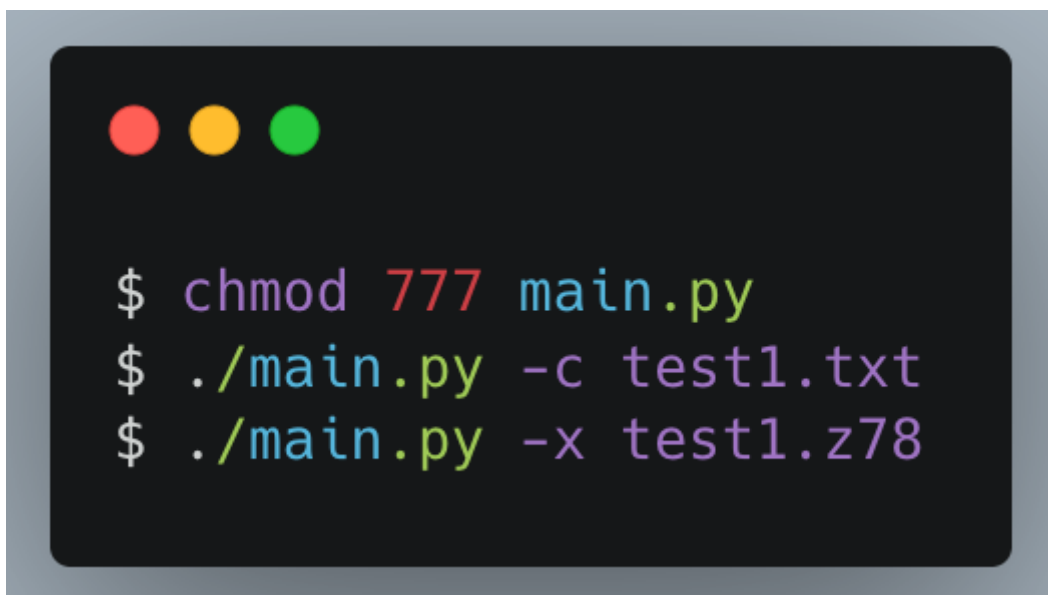
Nesse trabalho foi proposto a exploração, de forma prática, dos conceitos de manipulação de sequências principalmente no que tange a compressão de arquivos. Mais especificamente, o objetivo principal se apresentou na implementação do método LZ78 juntamente à árvores de prefixos.

- **Instruções**

A linguagem python foi escolhida para a realização do trabalho. Entre muitos fatores, a facilidade de utilização, bem como sua versatilidade, foram alguns dos critérios utilizados nessa escolha.

A primeira execução do programa, por sua vez, pode apresentar alguns erros de permissão. Para contorná-los é bem simples, basta seguir o seguinte fluxo de comandos.

OBS: O comando "chmod" dá as permissões necessárias ao script para que o mesmo possa ser executado.

A terminal window with a dark background and a light gray border. At the top left, there are three colored circles: red, yellow, and green. The terminal displays three lines of text, each starting with a dollar sign (\$) and followed by a command. The first command is 'chmod 777 main.py', the second is './main.py -c test1.txt', and the third is './main.py -x test1.z78'. The text is color-coded: '\$' is white, 'chmod' is purple, '777' is red, 'main.py' is green, './main.py' is green, '-c' is purple, 'test1.txt' is green, and '-x' is purple, 'test1.z78' is green.

```
$ chmod 777 main.py
$ ./main.py -c test1.txt
$ ./main.py -x test1.z78
```

- **Implementação**

Primeiramente, foi utilizada uma trie como estrutura principal de manipulação. Sua definição consiste, basicamente, em um construtor, um método para pegar o

prefixo de uma palavra, um método para checar se uma palavra está presente na trie e um método para adicionar elementos à estrutura. Sua implementação pode ser vista no pseudocódigo abaixo:

```
class Trie:
    # Trie constructor
    def __init__(self, value='', numeric_value=None, parent=None, empty=True):
        (...)

    # Get prefix of a word
    def _get_prefix(self, word):
        (...)

    # Returns if the trie contains the word
    def contains(self, word):
        (...)

    # Add one value to the trie
    def add(self, word, value):
        (...)
```

A compressão se baseia na representação do texto como um conjunto de prefixos ordenados, na qual cada sequência é identificada como um prefixo anteriormente cadastrado mais um caractere novo, caso necessário.

A utilização da trie nesse processo é fundamental, pois, com sua estrutura, podemos economizar espaço no armazenamento de prefixos. Em mapeamentos tradicionais, prefixos iguais poderiam ser duplicados ao longo do dicionário, fato que não ocorre na estrutura implementada, na qual a forma de indexação dos prefixos diminui tal ambiguidade.

O fluxo de compressão segue a seguinte lógica: passamos por todo o arquivo e checamos se a cadeia com o caractere já existe nos prefixos. Se sim, adicionamos o caractere a cadeia e continuamos para o próximo caractere do texto. Se não, adicionamos o novo prefixo e escrevemos no arquivo comprimido, o index é atualizado e passamos para o próximo caractere. O código abaixo representa a lógica implantada:

```

import trie as T
# Global variables
import glv

def compress(input_file, output_file):
    with open(input_file, 'r') as input:
        text = input.read()
    with open(output_file, 'wb') as output:
        index = 1
        chain = ''
        trie = T.Trie('', numeric_value=0)

        for c in text:
            contains = trie.contains(chain + c)
            if contains != -1:
                chain += c
                continue

            chain_index = trie.contains(chain)
            trie.add(chain + c, index)
            output.write(chain_index.to_bytes(glv.INT_SIZE, 'big'))
            output.write(ord(c).to_bytes(glv.CHAR_SIZE, 'big'))
            chain = ''
            index += 1

```

A descompressão, por sua vez, ocorre de forma um pouco mais simples. A lógica consiste em passar por todo o arquivo comprimido, identificá-lo na trie e o adicionar na cadeia de descomprimido, a qual será, após chegar no fim do arquivo, escrita no arquivo de saída. O código abaixo representa a lógica implantada:



```

import utils
# Global variables
import glv

def decompress(input_file, output_file):
    with open(input_file, 'rb') as file:
        decoded = ''
        trie = ['']
        while True:
            index = file.read(glv.INT_SIZE)
            if index == b'':
                break

            index = int.from_bytes(index, "big")
            char = file.read(glv.CHAR_SIZE)
            if char == b'':
                char = ''
            else:
                char = chr(int.from_bytes(char, 'big'))

            chain = trie[index]
            new_chain = chain + char
            decoded += new_chain
            trie += [new_chain]
        utils.write_file(output_file, decoded)

```

• Testes

Foram realizados diversos testes em arquivos de tamanhos variados com o intuito de avaliar a taxa de compressão. Tal taxa se apresentou diretamente relacionada ao tamanho do arquivo, bem como seu formato. Além disso, quanto maior o arquivo, maior foi o tempo de execução do script, sendo praticamente instantâneo nos melhores casos ou demorando até 30 segundos nos piores.

Podemos conferir abaixo o resultado da compressão em alguns arquivos de tamanhos variados (1KB até 2MB). A taxa de compressão funciona da seguinte forma: caso positiva significa que o arquivo cresce (no arquivo test1.txt o arquivo

cresceu 78.49%). Por outro lado, se a taxa for negativa o arquivo diminuiu de tamanho (no arquivo test10.txt o arquivo diminuiu 53.80%).



Compression: test1.txt -> test1.z78
Input 1004 bytes
Output 1792 bytes
Compression ratio +78.49%

Compression: test2.txt -> test2.z78
Input 5020 bytes
Output 6484 bytes
Compression ratio +29.16%

Compression: test3.txt -> test3.z78
Input 10040 bytes
Output 11008 bytes
Compression ratio +9.64%

Compression: test4.txt -> test4.z78
Input 50162 bytes
Output 43268 bytes
Compression ratio -13.74%

Compression: test5.txt -> test5.z78
Input 100324 bytes
Output 74156 bytes
Compression ratio -26.08%

Compression: test6.txt -> test6.z78
Input 200648 bytes
Output 129836 bytes
Compression ratio -35.29%

Os testes relacionados à descompressão foram omitidos mas seguem a mesma linha de raciocínio, sendo que os arquivos comprimidos voltam a ter o extamente o tamanho original (antes de serem comprimidos).

● Conclusão

Após a finalização do trabalho e realização de diversos testes é possível retirar algumas conclusões.

O método LZ78 não é ótimo para todos os textos. Em particular, textos pequenos, sem muitas repetições, tendem a ficarem maiores após comprimidos. Por outro lado, sua eficiência aumenta significativamente em determinados casos, em arquivos maiores onde a repetição predomina.

Em relação a complexidade, temos que as operações na trie giram em torno de $O(n)$. Por outro lado, o processo de compressão/descompressão passa por todo o texto, o que levaria a uma complexidade $O(m)$.

Em suma, o tema proposto foi de grande importância para que conceitos importantes da disciplina pudessem ser fixados de forma mais incisiva. Além disso, tal atividade se apresentou como uma forma única de praticar o conteúdo ensinado.