TP01 - Compressão LZ78

Leandro Freitas - 2021037902

1 Introdução

Esse trabalho se trata de um problema de compressão de arquivos: um arquivo é fornecido como entrada e o programa deve retornar um outro arquivo comprimido. Além disso, o código também deve descomprimir um arquivo devidamente. O algorítmo foi executado na linguagem Python 3.10.6

2 LZ78

O algorítmo de compressão LZ78 foi o algorítmo utilizado para a compressão de arquivos nesse trabalho. Tal algorítmo, inventado em 1978, se baseia no aproveitamento de strings já vistas anteriormente no arquivo para comprimir os dados.

2.1 Algorítmo

Para reaproveitar strings já usadas, o LZ78 utiliza um dicionário D, que nesse trabalho foi representado por uma trie, e códigos para referenciar ocorrências passadas de padrões.

Ao receber um texto de entrada, o algorítmo navega caractere por caractere e verifica se ele já foi inserido no dicionário. Caso ele já tenha sido inserido, o caractere é concatenado com o próximo, que por sua vez será analisado como uma string. Assim, esse passo é repetido indutivamente até que seja encontrada uma string que não foi inserida no dicionário. Nesse caso, o código insere a string no dicionário e retorna uma tupla (D_{seq}, c), em que D_{seq} é o código da última sequência encotrada e c o ultimo caractere da sequência inserida.

2.2 Trie

O dicionário do algorítmo LZ78 implementado nesse trabalho foi representado por uma Trie. Desse modo, é possível realizar as consultas e inserções em tempo $\mathcal{O}(m)$, sendo m o tamanho da string inserida.

Para a implementação da Trie, foi criada a classe "Node", que armazena os nós da árvore, de forma que cada nó armazene um mapa com os seus filhos e o código correspondente à string representada por cada nó. Não é necessário guardar se um nó é final ou não, uma vez que todos os nós serão finais na trie, já que toda string inserida terá apenas um caractere que "quebrará"a sequência.

A classe "Trie"armazena apenas a raíz da árvore, que é representada por um nó de código 0, e o tamanho da árvore. Além disso, a classe possui duas funções: Insert e Search, que realizam as operações de inserção e busca na árvore em tempo $\mathcal{O}(m)$, e retornam o código do prefixo correspondente.

c	cadeia	saída	D
A	''	(0, 'A')	[0:'', 1:'A']
_	''	(0, '_')	[0:'', 1:'A', 2:'_']
A	''	-	-
S	'A'	(1, 'S')	[0:'', 1:'A', 2:'_', 3:'AS']
A	''	-	-
_	'A'	(1, '_')	[0:'', 1:'A', 2:'_', 3:'AS', 4:'A_']
D	''	(0, 'D')	[0:'', 1:'A', 2:'_', 3:'AS', 4:'A_', 5:'D']
A	''	-	-
_	'A'	-	-
C	'A_'	(4, 'C')	[0:'', 1:'A', 2:'_', 3:'AS', 4:'A_', 5:'D', 6:'A_C']
A	''	-	-
S	'A'	-	-
A	'AS'	(3, 'A')	[0:'', 1:'A', 2:'_', 3:'AS', 4:'A_', 5:'D', 6:'A_C', 7:'ASA']
EOF	''	(0, '')	-

Figura 1: Exemplo do funcionamento da compressão LZ78 para a string "a asa da casa"

2.3 Compressão

Para a compressão, é necessário armazenar, para cada caractere, uma tupla (D_{seq}, c). Para economizar mais espaço, a tupla foi armazenada na seguinte sequência de bytes: o primeiro byte representa um inteiro que indica a menor quantidade de bytes que pode ser usada para armazenar D_{seq}, os próximos bytes, correspondentes ao número lido no byte anterior, correspondem a D_{seq}, e, por fim, a forma de c em bytes é armazenada, podendo precisar de 1 a 4 bytes, por ser codificado em UTF-8.

2.4 Descompressão

Para descomprimir o arquivo, basta fazer a leitura byte a byte dele. Sabemos que seu primeiro byte representa a quantidade de bytes usado pelo primeiro código, e assim por diante. Assim, basta recriar o dicionário, em forma de lista, a partir dos códigos, de maneira similiar à demonstrada na figura 1. Uma vez obtido o dicionário, ao concatenarmos todas as suas entradas, obteremos o texto descomprimido.

3 Exemplos

Para testar a efetividade do algorítmo, ele foi testado em 10 arquivos diferentes:

Nome do Arquivo	Tamanho Descomprimido(Kb)	Tamanho Comprimido(Kb)	Taxa de Compressão(%)
bible_ru.txt	6207	2762	44.50
bible.txt	3961	2341	59.10
constituicao1988.txt	637	349	54.79
crime_punishment.txt	1150	746	64.87
dom_casmurro.txt	401	287	71.57
dracula.txt	855	567	66.32
dune.txt	1171	756	64.56
grande_sertao.txt	1201	765	63.70
o_cortiço.txt	470	323	68.72
os_lusiadas.txt	337	251	74.48

Fazendo a média das taxas de compressão, obtemos que o código obteve uma taxa média de 63.26% de compressão.

4 Conclusão

Por fim, o algorítmo implementado conseguiu com sucesso reduzir o tamanho de todos os arquivos testados, de forma a atingir o objetivo requerido. No entanto, uma dificuldade enfrentada na implementação do código foi tornar o arquivo descomprimido idêntico ao arquivo de entrada. Em alguns casos, eles possuem uma quantidade diferente de bytes, embora possuam o mesmo conteúdo.