Trabalho Prático 1 - Algoritmo LZ78
Universidade Federal de Minas Gerais

Igor Rahzel Colares Galdino
Matricula 2020096255

Maio 2023

1 Introdução

O Trabalho Prático consiste na implementação do algoritmo LZ78, desenvolvido por Lempel e Ziv em 1978, para compressão e descompressão de arquivos de texto. A ideia desse algoritmo é substituir sequências repetidas de caracteres por códigos que as representem, reduzindo assim o tamanho do texto. As sequncias de caracteres do texto são armazenadas em um dicionário, porém para este trabalho prático foi solicitado a implementação de uma árvore Trie para tal função.

2 Árvore Trie

Inicialmente vamos explicar o funcionamento e a implementação da Árvore Trie, na seção subsequente entraremos em detalhes de como ela foi utilizado no algoritmo LZ78.

2.1 Funcionamento da Árvore Trie

Conceitualmente a Trie é uma árvore M-ária, onde para cada um de seus nós há M possíveis ligações para cada um dos símbolos do alfabeto. A ideia é quebrar o texto em palavras e inseri-las na Trie, de modo que cada nó armazene um única carácter, dessa forma as palavras que estão em uma mesma subárvore devem compartilhar o mesmo prefixo

2.2 Implementação

A árvore Trie foi implementada no arquivo *TrieTree.py*, ela tem como atributos *value* que representa o número referente àquele nó da árvore, *label* que representa o carácter respectivo àquele nó e por fim temos o atributo *children* que é uma lista contendo os filhos daquele nó.

Como o algoritmo LZ78 requer somente inserções na árvore Trie, foi implementada apenas a função **insert()**. Esse método essencialmente checa se a palavra a ser inserida na árvore já foi inserida anteriormente ou não, caso a palavra já tenha sido inserida o método retorna *None* caso contrário é retornado o último carácter da palavra. Para verificar se uma palavra já foi inserida começamos verificando a partir da raíz se algum de seus nós filhos é igual ao primeiro carácter da palavra, se isso acontece então realizamos esse mesmo procedimento, mas agora para os filhos do nó que era igual ao primeiro símbolo da palavra, esse procedimento é realizado até que hajam nós na árvore que correspondam ao i-ésimo caracter da string ou até chegarmos ao último carácter da mesma, nesse caso temos que a palavra já está na árvore, caso contrário, teremos encontrado o maior prefixo da palavra que foi armazenado na árvore e então a partir dele inserimos o restante da palavra criando um novo nó para cada carácter restante e fazendo com que o nó que representa carácter da posição *s[i+1]* seja filho do nó que representa o símbolo *s[i]*, onde *s* é a string que se deseja inserir na árvore Trie.

3 ALgoritmo LZ78

Nessa seção serão detalhadas as etapas de compressão e descompressão do algoritmo.

3.1 Compressão

Para realizar a compressão do arquivo texto utilizando o algoritmo LZ78, percorremos cada símbolo do texto e tentamos realizar a sua inserção na árvore trie, caso o carácter atual já tenha sido inserido devemos então formar uma palavra que consiste da concatenação do carácter atual com o próximo carácter do texto e então devemos tentar inserir essa palavra, repetimos esse procedimento até que a palavra seja inserida na árvore ou o final do texto seja atingido. É importante mencionar que toda vez que um símbolo ou palavra é inserido na Trie, é dado ao nó referente ao último carácter um valor, dessa forma é possível saber o número de inserções realizadas na árvore.

Quando uma inserção é realizada, é impresso o seguinte código no arquivo de saída(arquivo comprimido), **(index,símbolo)**, onde index é o valor do nó pai do nó onde o último símbolo da string foi inserido e o símbolo representa o último carácter da palavra.

Para que o arquivo realmente apresente um compressão significativo é necessário converter o códigos **(index,símbolo)** para binário, pois dessa forma economizamos o número de bits utilizados para a representação do código. Para atingir tal objetivo realiza-se uma primeira passada no arquivo e constrói-se a Árvore Trie somente com o intuito de saber o número de inserções realizadas nela. Através do número de inserções realizadas podemos então calcular o número máximo de bits necessário para representar cada **index**, esse valor é dado por $\lceil \log_2(N) \rceil$, sendo N o número de inserções. Já o **símbolo** do código é convertido para sua representação em bits do *UTF-8*. É necessário mencionar que o primeiro byte do arquivo comprimido contém a informação do número de bytes necessários para representação dos **indexes** utilizados no código, isso será importante na hora de realizar a descompressão do arquivo.

3.2 Descompressão

Inicialmente na descompressão lê-se o primeiro byte do arquivo, pois dessa forma será possível determinar quantos bytes devem ser lidos no arquivo comprimido para se obter o **index** do código,os próximos bytes são referentes a conversão do **símbolo** do código para a sua representação em *UTF-8*, isso implica que após a leitura dos bytes referentes ao **index**, deve-se ler o byte subsequente e analisar como é dado o formato dos seu primeiros bits analisando da esquerda para a direita, os possíveis casos são os seguintes: caso o primeiro bit do seja *0*, então basta decodificar o byte lido, caso os três primeiros bits sejam *110*, então é necessário ler mais um byte para decodificar aquele símbolo, caso os quatro primeiros dígitos sejam *1110* é necessário ler mais dois bytes e por fim se temos *11110* no inicio devemos ler mais 3 bytes para decodificar o símbolo.

Após a recuperação dos códigos do arquivo comprimido realizamos a sua inserção em um dicionário, que previamente é inicializado contendo a lista *[None,"]* associados à chave *0*. A inserção do código lido no dicionário ocorre da seguinte forma: é inserido a lista ***index,código*** na chave *pos*, onde *pos* representa a posição em que aquele código ocorre no texto comprimido. Para imprimir o conteúdo representado pelo código basta criar uma string em que seu símbolo mais a direita é o carácter lido no código, o símbolo a esquerda deste é dado pelo carácter presente na chave do dicionário referente ao **index** lido, então a partir da posição atual no dicionário realiza-se esse mesmo processo até que a chave acessada no dicionário, seja *0*, pois isso implica que todos os carácteres que são prefixos da palavra representada pelo código foram lidas. Por fim, basta imprimir a string obtida no texto e reiniciar o processo até que todo os códigos sejam lidos

4 Taxa de Compressão

A tabela abaixo indica a taxa de compressão de 10 arquivos com extensão .txt através da implementação realizada do algoritmo LZ78. A taxa de compressão é dada pela razão entre o tamnhao do arquivo original e o tamanho do arquivo comprimido.

Arquivo .txt	Tamanho	Tamanho Arquivo Comprimido	Taxa de Compressão
Hamlet	205,9 kB	117,5 kB	1.75
JuliusCaeser	137,1 kB	83,7 kB	1.63
KingRichard	197,1 kB	115,9 kB	1.70
Macbeth	125,8 kB	79,3 kB	1.58
MeasureForMeasure	217,4 kB	123,1 kB	1.76
MuchAdoAboutNothing	143,2 kB	86,3 kB	1.65
RomeoAndJuliet	163,6 kB	99,1 kB	1.65
TheMerchantOfVenice	141,9 kB	87,1 kB	1.62
TheTragedyOfKingLear	176,8 kB	106,5 kB	1.75
TwelthNight	136,2 kB	83,3 kB	1.63

5 Conclusão

O trabalho prático visou a implementação do algoritmo LZ78 com a utilização de uma árvore Trie como estrutura de dados auxiliar. Isso permitiu uma melhor compreensão do funcionamento das Árvores Tries visto em sala de aula, além disso o trabalho possibilitou o contato com algoritmos de compressão que foi algo que não havia ocorrido previamente no curso.

O algoritmo LZ78 basicamente tem como principio que a cada inserção realizada na árvore Trie deve haver um código correspondente a ela, e esse código representa o maior prefixo existente na árvore e o último carácter inserido.

6 Referências

:

Wikipedia. (2023, maio 9). LZ78. Em Wikipedia, A Enciclopédia Livre. Recuperado em maio 9, 2023, de https://pt.wikipedia.org/wiki/LZ78

Salomon, D. (2012). Data Compression: The Complete Reference (4th ed.). Springer.