DCC207 – Algoritmos 2
Gustavo Ferreira Dias

Trabalho Prático 1 – Compressão LZ78 com dicionário Trie

1. Introdução

O objetivo do trabalho foi resolver o problema de comprimir arquivos de texto implementando o algoritmo de compressão LZ78 usando uma Trie como dicionário.

"O LZ78 foi proposto por Lempel e Ziv em 1978. A ideia do algoritmo é substituir strings que se repetem no texto por um código, diminuindo assim o número de bytes gravados na saída."

"Uma árvore Trie é uma estrutura de dados que é usada para armazenar um conjunto de strings de maneira eficiente. Na árvore Trie, cada nó representa um caractere, e cada caminho da raiz até uma folha representa uma string. As strings são armazenadas na árvore de maneira que as strings comuns compartilhem o mesmo caminho na árvore até um determinado ponto, e então se ramificam para formar novos caminhos que representam o restante da string."

2. Implementação

O programa foi feito na linguagem Python, na versão 3.8.10. O ambiente de execução foi o Windows Subsystem for Linux 2 (WSL2) com o sistema operacional Ubuntu 20.04.5 LTS.

A execução do programa é feita da seguinte maneira:

python3 LZ78_TRIE.py "p1" "p2" "p3" "p4"

Na qual p1 é o primeiro parâmetro, podendo ser "-c" para compressão ou "-x" para descompressão, p2 o segundo parâmetro como o nome ou o caminho do arquivo a ser comprimido ou descomprimido, como opcionais, p3 "-o" para dar nome ao arquivo de saída após compressão ou descompressão e p4 o nome do arquivo de saída.

Caso não seja dado o nome do arquivo de saída, na compressão o nome da saída será o nome de entrada com a extensão trocada para .z78 e na descompressão ao contrário, com a saída .txt.

2.1 LZ78TRIE.py

O programa possui 1 classe e 2 funções principais.

A classe é a "Node_trie" que consiste nos nós da árvore trie. Cada nó possui 3 dados, um código associado a ele, um prefixo e um set de filhos. A classe possui a função "procura_char" que referência o nó que foi chamado e recebe um char. Ela retorna se esse char está em um dos filhos desse nó, verdadeiro ou falso.

Das 2 funções principais, a primeira é a “compactar” que recebe como parâmetro 2 strings, a primeira sendo o arquivo a ser compactado e a segunda o nome do arquivo que ela deve gerar na saída.

A função compactar abre o arquivo passado e cria um nó do tipo Node_trie com o código 0 e o prefixo vazio. Ela passa por um loop para iterar pelo arquivo, letra a letra até o fim, nessa iteração ela executa o método LZ78 de criação de dicionário, porém utilizando os nós da árvore trie, ela conta a quantidade de nós que a árvore possui e verifica se algum caractere do arquivo de entrada faz parte do UNICODE. Essas verificações são necessárias para a criação da saída em bytes, porque, caso existam, entre 0 e 255, ela precisará usar 1 byte para representar os códigos em inteiro dos nós na saída, caso existam mais de 2^8 nós na árvore, ela usará 2 bytes, caso existam mais de 2^16 nós na árvore, ela usará 3 bytes e assim por diante. Assim como se algum caractere está no padrão UNICODE, caso existam somente caracteres representados por inteiros entre 0 e 255, a saída pode utilizar somente 1 byte para armazenar os dados, significa que o arquivo de entrada se encontra no padrão ASCII, caso existam caracteres que representados por inteiros ultrapassem 255, significa que o arquivo de entrada está no padrão UNICODE e precisa de 2 bytes para representação no arquivo de saída, esse cálculo também é feito para padrões UNICODE, UTF-16, e UTF-32, podendo usar até 4 bytes.

O loop armazena a saída em, caracteres e inteiros, para o dicionário em 2 listas diferentes, e após calcular a quantidade de nós e se é do padrão ASCII ou UNICODE ela escreve os 2 primeiros bytes do arquivo a quantidade de bytes usadas para representar o código e o caractere e após isso ela escreve essas 2 listas em um arquivo de saída, o método de escrita é, o mesmo índice para a lista de inteiros e para a lista de caracteres, um subsequente do outro, em alternância, após isso, se sobrou algo na cadeia de construção do dicionário, ela escreve essa última tupla código cadeia. Toda a escrita no arquivo de saída é feita em binário.

A função descompactar recebe também 2 strings como parâmetros, a primeira o nome do arquivo a ser descompactado e a segunda o nome do arquivo de saída. A função abre o arquivo a ser descompactado em leitura binária e o arquivo de saída em escrita normal. Os 2 primeiros bytes lidos são o formato de armazenamento dos códigos e dos caracteres, então ela entra em um loop que enquanto existir código, ela lê os bytes da forma correta e executa a descompressão LZ78, armazenando os caracteres em uma mesma lista com o índice sendo o código a cada iteração e escrevendo os valores associados aos índices juntamente com o caractere após o código no arquivo de saída.

As funções são chamadas de acordo com os parâmetros de entrada explicados na seção 2. Implementação e utiliza somente a biblioteca sys para manipulação dos parâmetros.

Observações importantes: Caso o arquivo de texto original seja criado no sistema operacional Windows e comprimido, a descompressão desse arquivo de texto utilizando Python em um sistema operacional baseado em UNIX muda o tipo de estilo e pode ocorrer um aumento na taxa de bytes bem pequeno entre o arquivo original e o arquivo descomprimido. O ideal é utilizar o mesmo sistema operacional

para criar o arquivo original, comprimir e descomprimir para que eles sejam idênticos.

2.2 Complexidade

A complexidade da função de compressão:

Seja C a quantidade de caracteres no texto e S a quantidade de nós na árvore trie que é formada. Para a criação da árvore cada caractere é procurado na árvore se ele existe ou não na árvore, então no pior caso chega a ser O(C log S) e para a escrita dos nós na saída, ele só itera por 2 listas com o código e o prefixo que, por consequência da compressão, são menores que C. Logo a complexidade total da compressão é O(C log S).

Em relação ao espaço, desconsiderando a abertura do arquivo que ocupa seu espaço em bytes na memória, o programa cria somente 2 listas representando o dicionário a ser escrito no arquivo de saída, que representam a média das taxas de compressões exemplificados no item 2.3 Taxas de Compressão a seguir.

A complexidade da função de descompressão:

A descompressão funciona da forma LZ78, para N a quantidade de bytes no arquivo comprimido, a complexidade é O(N), pois ele itera por todo o arquivo e armazena os dados em uma lista, essa lista é acessada diretamente pelo seu índice apontado pelo código lido, sem a necessidade de iterar na lista.

O espaço é a taxa de compressão a seguir, porém ao contrário, o programa durante a leitura da entrada, escreve na saída de forma descomprimida.

2.3 Taxas de Compressão

A seguir 10 exemplos de compressão de arquivos de texto de diferentes tamanhos:

Nome	Tamanho Original	Tamanho comprimido	Taxa comparado ao original
codigoLZ78TRIEgrande	92kb	36kb	39,13%
constituicao1988	637kb	427kb	67,03%
datasetcasas	40kb	21kb	52,50%
dom_casmurro	401kb	286kb	71,32%
enunciadotp1grande	282kb	103kb	36,52%
htmlemtxt	152kb	62kb	40,79%
LoremIpsumASCII	31kb	19kb	61,29%
LoremIpsumUNICODEcharacters	71kb	30kb	42,25%
os_lusiadas	337kb	189kb	56,08%

tabelalattes	1330kb	735kb	55,26%

Arquivos disponíveis na pasta ArquivosTeste, formato UTF-8, estilo Unix(LF)

3. Referências

Python Classes https://docs.python.org/3/tutorial/classes.html
Python built in functions https://docs.python.org/3/library/functions.html#open
Python std types https://docs.python.org/3/library/stdtypes.html
Trie https://en.wikipedia.org/wiki/Trie
LZ78 https://pt.wikipedia.org/wiki/LZ78#Exemplo