Henrique Soares Assumpção e Silva
Matrícula: 2020006620
henriquesoares@dcc.ufmg.br

Trabalho Prático I
Algoritmos II
Universidade Federal de Minas Gerais

5 de junho de 2022

1 Introdução

O Trabalho Prático I propõe a implementação do algoritmo LZ78 [1] para codificação e decodificação de textos. O algoritmo foi proposto por Lempel e Ziv em 1978,e até hoje é uma das bases para ferramentas de compressão modernas. O algoritmo em questão demanda de um dicionário para manter um mapeamento entre substrings que já ocorreram no texto e códigos fornecidos, e tal dicionário deve ser implementado por meio de uma árvore Trie [2], que consiste em uma estrutura de dados famosa comumente utilizada no contexto de análise de textos.
Assim sendo, define-se como objetivos principais deste trabalho a implementação de uma versão correta e funcional do algoritmo LZ78, bem como a implementação de uma árvore trie para manejar os mapeamentos necessários pelo algoritmo.

2 Modelagem

O programa foi desenvolvido na linguagem Python $3.9.4$, no sistema operacional Windows 10. O programa foi inicialmente implementado e compilado em uma máquina com 16GB de RAM em um processador Intel Core I5-9400F. A biblioteca nativa *argparse* foi utilizada para implementar as especificações da linha de comando, e a biblioteca *NumPy* foi utilizada para realizar a operação de logaritmo na base 2, para calcular o número de bits necessários para o endereço no processo de descompressão, e também para ler e salvar os bits de um arquivo de texto de entrada.
O problema proposto para este trabalho prático foi dividido em duas componentes principais: (i) desenvolvimento da Árvore Trie para ser utilizada como dicionário no processo de compressão, e (ii) implementação do algoritmo LZ78 em si para comprimir/descomprimir arquivos de texto. Esta documentação irá discutir tais componentes de forma sequencial.

2.1 Árvore Trie

Uma Árvore Trie consiste de uma estrutura que organiza os dados em forma de árvore, i.e., um grafo conexo e acíclico, onde cada nó pode possuir um número arbitrário de filhos. Tais árvores geralmente são implementadas no contexto de processamento textual, e portanto define-se que cada nó deve armazenar um símbolo pertencente ao corpus de texto em questão, e portanto um caminho feito da raiz até um nó determina uma sequência de símbolos. Se tal sequência foi observada no texto, adiciona-se um índice identificador ao nó que contém o último símbolo da sequência, ou seja, um caminho feito da raiz até um nó indexado representa uma sequência de símbolos, i.e., palavra, que aparece no texto em questão. A árvore trie é interessante do ponto de vista de estruturas de dados justamente por permitir uma busca rápida na estrutura para descobrir se uma determinada sequência de símbolos já foi observada no texto.
Para o trabalho em questão, optou-se por implementar uma Árvore Trie padrão, i.e., onde cada nó armazena somente um símbolo. Para isso, definiu-se duas classes em Python para representar os Nós e a Árvore em si. Cada nó pode armazenar um índice identificador, um símbolo e uma lista de nós filhos, e tal nó também implementa um método para buscar por um dado símbolo em seus nós filhos. Cada árvore armazena um nó raiz padrão que não armazena nenhum símbolo nem índice, e tal árvore implementa dois métodos para auxiliar na manutenção da estrutura. Os métodos são descritos a seguir:

- *Método de inserção*: Método base da árvore que recebe uma seqûencia de símbolos e os insere na árvore de acordo com a lógica definida pela estrutura Trie, i.e., iniciamos a operação pela raiz e então comparamos o primeiro símbolo da sequência com os símbolos armazenados pelos filhos da raiz, caso algum nó possua tal símbolo, nos movemos para ele e repetimos o processo para o segundo símbolo da sequência, e caso nenhum nó possui tal símbolo, um novo nó filho é criado armazenando esse símbolo, e então nos movemos para ele e repetimos o processo para o segundo símbolo. Essa operação é repetida até que todos os símbolos da sequência tenham sido percorridos, ou seja, buscamos pelo maior prefixo da sequência existente na árvore e então criamos os demais nós necessários para acomodar a sequência. Note que, o nó que representa o fim da sequência é indexado de acordo com um determinado valor dado pelo usuário;

- *Método de busca*: Este método realiza uma busca por um determinado padrão dentro da árvore, seguindo uma lógica semelhante ao supracitado método de inserção. O método recebe uma sequência alvo, e então, a partir do nó raiz, busca recursivamente por nós filhos que armazenem um símbolo igual ao atual símbolo da sequência. Caso nem todos os símbolos da sequência tenham sido comparados mas o nó atual do método não possua filhos, retorna-se que a sequência não está na árvore. Caso todos os símbolos da sequência tenham sido comparados e o método tenha encontrado um nó que corresponde a tal sequência, checa-se se tal nó já está indexado, caso positivo retorna-se que a sequência está presente na árvore, caso contrário retorna-se que ela não está na árvore.

Com apenas os dois métodos descritos, é possível utilizar a Árvore Trie como dicionário para o algoritmo LZ78 de maneira eficiente, pois ambas as operações possuem custo de tempo da ordem do tamanho da sequência de entrada, ou seja, um custo praticamente constante quando comparado com o tamanho total do texto. No pior caso, ambas as operações possuem custo linear no tamanho do texto.

2.2 Compressão de Lempel-Ziv

A Compressão de Lempel-Ziv, também conhecida como método/algoritmo LZ78, foi um algoritmo proposto na década de 70 para a compressão de Stream Codes, i.e., sequências de símbolos que geralmente são fornecidas aos poucos para o usuário, ou seja, o usuário inicialmente não tem acesso a todo o corpus de texto. O algoritmo possui duas etapas: (i) Compressão, onde um dicionário, no caso Árvore Trie, é mantido de modo a mapear padrões novos observados no texto com índices numéricos, de modo que a sequência comprimida será formada por uma série de padrões do tipo *(índice de padrão observado;símbolo)*, e (ii) Descompressão, onde uma sequência comprimida é lida e o processo reverso é realizado, de modo que sempre que os índices de padrão são lidos eles são associados a um padrão, que então é concatenado com o símbolo fornecido e adicionado a sequência descomprimida.
A implementação realizada neste projeto desenvolve duas funções que implementam os supracitados processos de compressão e descompressão de um arquivo. Tais funções são descritas a seguir:

- *Compressão LZ78*: Para realizar a compressão, primeiramente instanciamos um objeto do tipo Árvore Trie e então inserimos o símbolo vazio indexado pelo valor zero. Primeiramente inicializamos uma variável controle vazia que irá conter uma sequência nova de símbolos observados, e então iteramos sobre cada símbolo do texto de entrada. Inicialmente concatenamos o novo símbolo à variável controle e então buscamos por este padrão na trie, caso ele exista e o caractere atual seja o último do texto, adicionamos o índice referente ao prefixo do padrão em conjunto com o novo caractere na compressão do texto, caso contrário prosseguimos para o próximo caractere - isso é realizado para garantir que, caso os últimos símbolos do texto representam um padrão previamente observado, tais símbolos sejam adicionados na string final de compressão. Caso o padrão não exista na trie, nós o inserimos na mesma e adicionamos o índice referente ao prefixo do padrão em conjunto com o novo caractere na compressão, e além disso resetamos a variável controle para a string vazia. Ao final das iterações, obtemos uma string de símbolos que representam a compressão do texto original;

- *Descompressão LZ78*: A descompressão implementa o processo reverso ao descrito anteriormente. Por motivos de eficiência, optou-se por utilizar dicionários da linguagem Python para implementar o mapeamento de índices para sequências, tendo em vista que a Árvore Trie realiza o mapeamento de sequências para índices, e nesse caso a operação de encontrar uma palavra dado um índice será sempre de custo linear no tamanho da árvore, que pode ficar significativamente grande. Inicialmente mapeamos o índice zero para a string vazia, e então percorremos o texto comprimido, de modo a recuperar o padrão do prefixo que é indexado pelo primeiro elemento da tupla *(índice de padrão observado;símbolo)*, e então adicionar tal padrão concatenado com o símbolo tanto ao dicionário quanto à string que representará a mensagem descomprimida. Ao final das iterações, obtemos uma string descomprimida idêntica a string original fornecida.

Com as duas funções descritas, é possível implementar o algoritmo LZ78 de forma completa. Como a operação de acesso tanto à árvore Trie quanto ao dicionário em Python possuem custo de ordem praticamente constante, a complexidade de tempo final de ambas as operações é praticamente linear no tamanho do texto de entrada. Vale ressaltar que empiricamente o processo de compressão é significativamente mais lento que o processo de descompressão, tendo em vista o alto custo de espaço para a árvore trie, bem como a outros fatores da própria linguagem Python que afetam a eficiência de estruturas implementadas por completo nessa linguagem.

2.3 Estrutura do Programa

O programa final segue a estrutura especificada pelo enunciado, i.e., recebe da linha de comando especificações para qual operação realizar (compressão ou descompressão), bem como o nome do arquivo alvo e uma

especificação opcional para renomear o arquivo caso desejado. Os arquivos comprimidos são salvos com a extensão ".z78". O programa final implementa a seguinte ordem de operações:

1. O programa recebe os argumentos da linha de comando, e então identifica qual operação deve realizar;
2. Caso o usuário queira comprimir um arquivo, o programa lê os bytes do arquivo e então os converte para bits. Tais bits são então comprimidos utilizando a função descrita na seção 2.2, e então os bits comprimidos são salvos em um arquivo ".z78";
3. Caso o usuário queira descomprimir um arquivo, o programa lê os bytes do arquivo alvo comprimido e então os converte para bits. Tais bits são então descomprimidos utilizando a função descrita na seção 2.2, e então os bits descomprimidos são salvos em um arquivo ".txt";

Após os supracitados passos, o programa encerra. Vale ressaltar que, apesar de não ser necessário, optou-se por realizar a compressão somente nos binários dos arquivos de texto. Isso facilita o processo de compressão e descompressão, além de possibilitar uma visualização mais intuitiva dos efeitos de compressão do algoritmo.

3 Resultados para alguns casos de teste

Esta seção irá apresentar os resultados quantitativos de compressão para alguns casos de teste propostos. Todos os casos de teste são textos reais disponíveis gratuitamente na internet, e os respectivos links para obtenção dos textos estão presentes nas referências deste documento. Os resultados serão listados a seguir:

- Compressão do Livro *"O Senhor dos Anéis: A Sociedade do Anel"*[4]: O arquivo original possui um tamanho de 988KB, enquanto o arquivo comprimido possui um tamanho de 656KB. A taxa de compressão obtida, definida como a razão γ entre o tamanho em bits do arquivo original e o arquivo comprimido, é dada por $\gamma \approx 1.54$, com um tempo de compressão de 112 segundos e descompressão de 2.98 segundos;
- Compressão do Livro *"Neuromancer"*[5]: O arquivo original possui um tamanho de 442KB, enquanto o arquivo comprimido possui um tamanho de 324KB. A taxa de compressão obtida é dada por $\gamma \approx 1.36$, com um tempo de compressão de 43 segundos e descompressão de 0.96 segundos;
- Compressão do Livro *"O Livro da Goetia do Rei Salomão"*[6]: O arquivo original possui um tamanho de 187KB, enquanto o arquivo comprimido possui um tamanho de 147KB. A taxa de compressão obtida é dada por $\gamma \approx 1.26$, com um tempo de compressão de 16 segundos e descompressão de 0.27 segundos;
- Compressão de texto gerado por *Ipsum Lorem*[7]: O arquivo original possui um tamanho de 29KB, enquanto o arquivo comprimido possui um tamanho de 25KB. A taxa de compressão obtida é dada por $\gamma \approx 1.19$, com um tempo de compressão de 2 segundos e descompressão de 0.04 segundos;
- Compressão da página da Wikipédia de *Código Lempel-Ziv*[1]: O arquivo original possui um tamanho de 5KB, enquanto o arquivo comprimido possui um tamanho de aproximadamente 5KB. A taxa de compressão obtida é dada por $\gamma \approx 1.001$, com um tempo de compressão de 0.37 segundos e descompressão de 0.01 segundos;
- Compressão da página da Wikipédia de *Kendrick Lamar*[8]: O arquivo original possui um tamanho de 50KB, enquanto o arquivo comprimido possui um tamanho 44KB. A taxa de compressão obtida é dada por $\gamma \approx 1.13$, com um tempo de compressão de 3.84 segundos e descompressão de 0.07 segundos;
- Compressão da página da Wikipédia de *Che Guevara*[9]: O arquivo original possui um tamanho de 107KB, enquanto o arquivo comprimido possui um tamanho 86KB. A taxa de compressão obtida é dada por $\gamma \approx 1.24$, com um tempo de compressão de 9 segundos e descompressão de 0.14 segundos;
- Compressão da página da Wikipédia do *Grêmio FBPA*[10]: O arquivo original possui um tamanho de 85KB, enquanto o arquivo comprimido possui um tamanho 72KB. A taxa de compressão obtida é dada por $\gamma \approx 1.18$, com um tempo de compressão de 6 segundos e descompressão de 0.11 segundos;
- Compressão da página da Wikipédia do *Governo Lula*[11]: O arquivo original possui um tamanho de 60KB, enquanto o arquivo comprimido possui um tamanho 50KB. A taxa de compressão obtida é dada por $\gamma \approx 1.19$, com um tempo de compressão de 6 segundos e descompressão de 0.12 segundos;
- Compressão da página da enciclopédia de Stanford sobre o *Problema da indução*[12]: O arquivo original possui um tamanho de 106KB, enquanto o arquivo comprimido possui um tamanho 84KB. A taxa de compressão obtida é dada por $\gamma \approx 1.25$, com um tempo de compressão de 9 segundos e descompressão de 0.14 segundos;

Referências

[1] https://pt.wikipedia.org/wiki/LZ78

[2] https://pt.wikipedia.org/wiki/Trie

[3] David Salomon. Data Compression: The complete reference. 4th edition. Springer

[4] https://www.ae-lib.org.ua/texts-c/tolkien__the_lord_of_the_rings_1__en.htm

[5] https://gist.github.com/m-242/ecb3e130b76a3b12f7ef41b04f486405

[6] https://archive.org/stream/ac_goetia/ac_goetia_djvu.txt

[7] https://www.lipsum.com/

[8] https://en.wikipedia.org/wiki/Kendrick_Lamar

[9] https://pt.wikipedia.org/wiki/Che_Guevara

[10] https://pt.wikipedia.org/wiki/Gr%C3%AAmio_Foot-Ball_Porto_Alegrense?tableofcontents=1

[11] https://pt.wikipedia.org/wiki/Governo_Lula

[12] https://plato.stanford.edu/entries/induction-problem/