Trabalho Prático **01**

Documentação

DCC207 Algoritmos 2

Universidade Federal de Minas Gerais ***2023/01***
Bernardo Reis de Almeida *(2021032234)*

1. Introdução

O presente trabalho prático tem por objetivo a manifestação e o desenvolvimento das habilidades teóricas trabalhadas sobre o tema **algoritmos para manipulação de sequências**, o que será feito em uma aplicação prática, seja ela a **compressão de arquivos de texto**.

A fim de atingir essa meta, será implementado um programa que realiza a compressão desses arquivos pelo método **LZ78**, tomando como base uma **árvore Trie compacta** como estrutura de dados auxiliar. Desenvolvido em 1978 por Abraham Lempel e Jacob Ziv, tal algoritmo é baseado na ideia de se codificar sequências de caracteres que aparecem no texto e as armazenar em uma estrutura de dados que permita recuperar esses códigos e os substituir pelas aparições sucessivas da mesma sequência, de modo a reduzir a quantidade de bits utilizada para representar toda a informação. Usualmente, utiliza-se um dicionário para o mapeamento de códigos a sequências, mas, na presente implementação, isso será feito por meio de uma árvore Trie compacta, cujos nós armazenam prefixos e, possivelmente, seus códigos.

A implementação foi feita por meio da **linguagem C++** em um ambiente **Linux** (Debian GNU/Linux 11 (bullseye)). Um Makefile é disponibilizado para a geração do executável. Detalhes sobre o funcionamento do algoritmo e sobre as estruturas de dados utilizadas podem ser encontrados na seção **[Implementação]**, enquanto as instruções para compilação e para execução podem ser encontradas na seção **[Considerações Finais]**.

2. Implementação

O programa possui dois principais elementos: **a implementação de uma árvore Trie compacta** que atuará como estrutura de dados auxiliar para o processamento das informações e **a implementação dos algoritmos de compressão e de descompressão** em si.

A árvore Trie compacta foi implementada por meio de duas classes: uma que modela **um nó da árvore** e outra que modela **a estrutura da árvore** (a relação entre os nós). A cada nó, é associada uma string relativa à sequência que aquele nó representa, a qual, no caso, pode ser algum prefixo comum de sequências previamente inseridas na árvore ou o sufixo de alguma delas, indicando sua existência na estrutura de dados. A cada um, também é associado um código, o qual assume o valor -1 para nós que representam apenas prefixos em comum e assume um valor positivo para aqueles que representam as sequências inseridas na árvore. O código é utilizado para fins de manipulação no algoritmo de compressão e seus valores são controlados pela árvore. No caso da descompressão, o código passa a ser uma string, o que será discutido mais a fundo adiante. Por fim, a cada nó, também é associado um mapa que mapeia caracteres a outros nós, indicando justamente a relação de parentesco entre estes e permitindo recuperar filhos com base no primeiro caractere da string que eles armazenam. A estrutura da árvore em si possui um contador de códigos que incrementa a cada sequência nela inserida e uma referência para a raiz da árvore, a partir da qual todas as operações são realizadas de maneira recursiva.

Existem três principais operações de manipulação da árvore: uma **operação de inserção**, uma **operação de verificação de existência** e uma **operação de busca de código**. A operação de inserção recebe uma sequência e a insere na árvore de maneira conforme com o funcionamento de uma Trie compacta (o maior prefixo comum das sequências inseridas é sempre extraído e encapsulado em um nó pai e cada nó é indexado pelo primeiro caractere da sequência que armazena). A operação de verificação de existência busca recursivamente na árvore por uma dada

string e retorna um valor booleano referente a essa condição. Por fim, a operação de busca de código recebe uma dada string e retorna seu código.

Uma observação pertinente é que **a implementação realizada não é generalizada: ela foi adaptada ao funcionamento do algoritmo de compressão/descompressão**. A árvore Trie em si, o que inclui as operações de inserção e de verificação de existência, comporta-se da maneira esperada para essa estrutura de dados. Algumas especificidades incluem o fato de strings repetidas serem ignoradas e de não haver um marcador claro para nós internos que também sejam sufixos (o uso de um marcador, como $, causou conflitos, de modo que essa diferenciação foi feita com o uso dos códigos de cada nó), mas ela é geral. Porém, a designação de códigos aos nós assume que sempre serão inseridas strings que ainda não estejam na árvore e a busca de códigos na estrutura de dados assume que a string sendo buscada já tenha sido inserida. Não foi realizado o tratamento para os casos em que é necessário associar mais de um código a uma mesma string e em que o código de uma string que não pertence à árvore for buscado pois, dada a maneira como o algoritmo funciona, julga-se que estes nunca ocorrerão.

Uma segunda observação é que, da maneira como o algoritmo LZ78 realiza inserções na árvore, por mais que esta seja uma implementação compacta, **ela se comportará, em geral, como uma Trie convencional**. A razão para isso se deve ao fato de que as novas strings inseridas se diferenciam das anteriores em apenas um caractere: o último acrescentado, isto é, toda vez que uma string com mais de um caractere for inserida, tem-se que todos os seus prefixos já terão sido inseridos, o que resulta em um ramo de prefixos de apenas um caractere em comum. Em outras palavras, todos os nós da árvore armazenam apenas um caractere e não há "compactação".

O **algoritmo LZ78**, como já mencionado, foi implementado com base na árvore Trie compacta, ao invés de em um dicionário convencional. No caso da **compressão**, inicialmente, o arquivo é lido uma primeira vez e sua correspondente árvore Trie é construída para a verificação da quantidade de códigos necessária para a representação de toda a informação. Esta será utilizada para determinar a quantidade de bytes utilizada para a escrita dos códigos no arquivo de saída e será o primeiro elemento escrito neste. Em seguida, a árvore é resetada e o algoritmo propriamente dito é executado: começando com uma STRING vazia, para cada novo caractere C do texto, caso a sequência STRING + C não esteja na árvore, imprime-se no arquivo de saída o código de STRING seguido do caractere C, a sequência em si é inserida na estrutura de dados e a STRING é resetada. Caso ela já esteja na árvore, atualiza-se STRING com STRING + C e se repete os mesmos passos para o próximo caractere. Ao final, caso a leitura do arquivo de entrada termine e haja elementos em STRING, há a emissão de um par código + caractere representando-a. A **descompressão**, por sua vez, exige ainda mais adaptações que permitam um uso eficiente da árvore Trie compacta como estrutura de dados auxiliar. Em particular, nesse caso, os códigos das sequências serão utilizados para a inserção na árvore, isto é, como a "sequência" em si, enquanto as sequências a eles correspondentes serão utilizadas como o "código". Em outras palavras, os papéis de código e de sequência foram invertidos, o que é natural, dado que a descompressão realiza o oposto da compressão. Dessa forma, além dos atributos, os métodos de inserção e de busca de código foram adaptados e a Trie opera de maneira inversa, porém, ainda como uma Trie deveria operar. As mesmas observações quanto às especificidades de implementação ainda são pertinentes. Com isso, inicialmente, o algoritmo de descompressão lê a quantidade de bits utilizada para a representação dos códigos e, em seguida, lê do arquivo de entrada os pares código + caractere, inserindo-os progressivamente na árvore com um contador de códigos incremental para recuperação póstuma e imprimindo na saída a sequência referente àquele código seguida do caractere em questão, o qual marca o final de uma sequência ainda não vista.

3. Exemplos

Todos os arquivos de texto utilizados nessa amostra exemplar estão disponíveis na pasta "examples" do repositório do programa.

The Philistine: a periodical of protest (Vol. II, No. 3, February 1896)
Tamanho Inicial: **72 KB**
Tamanho Final: **49 KB**
Taxa de compressão de **~32%**.

Christmas Builders
Tamanho Inicial: **45 KB**
Tamanho Final: **31 KB**
Taxa de compressão de **~31%**.

Natalie: Ein Beitrag zur Geschichte des weiblichen Herzens
Tamanho Inicial: **381 KB**
Tamanho Final: **272 KB**
Taxa de compressão de **~28.5%**.

Ubirajara: Lenda tupi
Tamanho Inicial: **225 KB**
Tamanho Final: **129 KB**
Taxa de compressão de **~42.5%**.

The Prince
Tamanho Inicial: **301 KB**
Tamanho Final: **164 KB**
Taxa de compressão de **~45.5%**.

The Adventures of Tom Sawyer
Tamanho Inicial: **424 KB**
Tamanho Final: **300 KB**
Taxa de compressão de **~29%**.

Walden
Tamanho Inicial: **652 KB**
Tamanho Final: **446 KB**
Taxa de compressão de **~31.5%**.

The Expedition of Humphry Clinker
Tamanho Inicial: **867 KB**
Tamanho Final: **571 KB**
Taxa de compressão de **~34%**.

The Adventures of Roderick Random
Tamanho Inicial: **1093 KB**
Tamanho Final: **682 KB**
Taxa de compressão de **~37.5%**.

Uarda, Complete: A Romance Of Ancient Egypt
Tamanho Inicial: **1056 KB**
Tamanho Final: **664 KB**
Taxa de compressão de **~37%**.

4. Considerações Finais

Instruções para compilação e para execução

1. Dentro do diretório raiz do repositório, execute o comando **< $ make >**, o qual deverá gerar um executável denominado **tp1**.

2. A execução do programa é realizada por meio de dois comandos:

< $ ./tp1 -c input.txt -o output.z78 >

...sendo **input.txt** o arquivo de texto a ser comprimido e **output.z78** o arquivo final da compressão. A passagem deste último é opcional e, caso não seja feita, um novo arquivo final será criado com o nome do original e com extensão **.z78**.

< $ ./tp1 -x input.z78 -o output.txt >

...sendo **input.z78** o arquivo de texto a ser descomprimido e **output.txt** o arquivo final da descompressão. A passagem deste último é opcional e, caso não seja feita, um novo arquivo final será criado com o nome do original e com extensão **.txt**.

3. A saída será retornada pelo arquivo passado como parâmetro ou, caso este não exista, por um novo arquivo criado conforme as diretrizes de nomeação de arquivos do programa.

Referências Bibliográficas

Stack Exchange, Inc.. *Stack Overflow* [Online]. Disponível em: https://stackoverflow.com/ (Acessado em: 07/05/2023).

Wikipédia, a Enciclopédia Livre. *LZ78* [Online]. Disponível em: https://pt.wikipedia.org/wiki/LZ78 (Acessado em: 07/05/2023).

Hart, M. *Project Gutenberg* [Online]. Disponível em: https://www.gutenberg.org/ (Acessado em: 07/05/2023).