UFMG

Algoritmos 2 - Trabalho Prático 1
Manipulação de sequências

Arthur Pontes Nader - 2019022294
Belo Horizonte - MG - Brasil

1) Introdução

Um importante tópico que um cientista da computação deve ter conhecimento é sobre como utilizar estruturas de dados para realizar a compressão de arquivos. Isso possibilita, dentre várias aplicações, que a transmissão de uma certa informação seja feita mais rapidamente e que o armazenamento de determinada quantidade de dados possa ser feito usando menor quantidade de espaço em disco.

Dessa forma, o objetivo deste trabalho prático foi realizar a implementação do algoritmo de compressão sem perdas LZ78 utilizando uma árvore de prefixos, também denominada “trie”. Esse algoritmo se baseia essencialmente na substituição de strings que já ocorreram no texto por um determinado código, diminuindo assim o número de bytes necessários para se gravar a mesma informação em disco. Para implementação do programa, utilizou-se a linguagem de programação Python.

2) Implementação

Primeiramente, procurou-se definir qual formato de codificação do texto inicial o algoritmo receberia. Assim, decidiu-se que a abertura do arquivo a ser comprimido seria feita usando o padrão UTF-8, pois dessa maneira, o compressor criado pode ser utilizado para uma maior variedade de textos. Essa decisão de projeto permitiu, por exemplo, que um texto com presença de caracteres do alfabeto russo pudesse ser comprimido e descomprimido adequadamente.

O arquivo comprimido é salvo em formato binário, o que permite uma maior taxa de compressão. Adotou-se a seguinte representação para leitura e escrita do arquivo binário:

- O primeiro byte trata-se de uma informação sobre os próximos bytes do arquivo. Define-se a variável “n” como sendo o valor decimal associado aos 4 bits mais significativos e a variável “c” como o valor decimal correspondente aos 4 bits menos significativos desse byte .

- Os próximos “n” bytes, quando convertidos para decimal, indicarão qual é o código do pai relativo ao nó em questão.
- Os próximos “c” bytes se referem ao caractere associado ao nó de acordo com o padrão de codificação UTF-8.
- Retorna-se a o primeiro passo e o processo é repetido iterativamente até que o procedimento seja concluído.

Para implementação do algoritmo LZ78 usando a trie, foi necessário uma classe e quatro funções, descritas a seguir:

- Classe Node:

Essa classe é usada para criação de um nó da trie. Objetos criados utilizando essa classe possuem como atributos: o caractere associado ao nó, um dicionário que guarda os caracteres e os filhos correspondentes, um número que indica o código do pai e outro que guarda o código do próprio nó.

- Função create_trie_and_encoding:

Essa função recebe como parâmetro um texto e, por meio da criação de uma trie, realiza a compressão desse texto. A árvore de sufixos é construída de tal forma que um novo nó só é adicionado caso ele seja referente a um prefixo que ainda não ocorreu no texto. Essa propriedade, em conjunto com a associação de um código a esse novo nó, é o que permite que a compressão do texto ocorra. A cada novo prefixo encontrado, realiza-se a codificação em um arquivo binário tal qual a representação descrita anteriormente.

- Função decoding:

Já essa função recebe o nome de um arquivo comprimido e, por meio de uma iteração manipulando os bytes, realiza a descompressão do arquivo utilizando duas listas. À medida que os bytes vão sendo decodificados, adiciona-se às listas o código relativo ao restante do prefixo do trecho a ser decodificado e o último caractere responsável pela geração desse novo prefixo. Essas adições nas listas são fundamentais pois demais bytes do arquivo binário geralmente necessitam de acesso a essas informações para decodificar outros trechos do arquivo.

- Função convert_to_bytes:

Essa é uma função auxiliar que tem como objetivo transformar os parâmetros de entrada (número de bytes do código, número de bytes do caractere, código e caractere) em bytes a serem escritos no arquivo binário.

- Função main:

A função main possibilita uma interface entre a linha de comando e o programa por meio da utilização da biblioteca argparse. É nessa parte do código que será definido se o programa deverá realizar uma compressão ou uma descompressão.

3) Resultados

Uma métrica muito importante usada para avaliar o resultado obtido em um algoritmo de compressão de dados é a taxa de compressão. Basicamente, trata-se de uma razão que expressa o quanto o arquivo inicial foi comprimido, sendo calculada dividindo-se o tamanho inicial do arquivo pelo tamanho após compressão. A tabela a seguir mostra as taxas de compressão obtidas para cada um dos arquivos comprimidos pelo algoritmo:

Nome do arquivo	Tamanho inicial (bytes)	Tamanho após compressão (bytes)	Taxa de compressão
linguagensdeprogramacao	35.952	31.733	1.133
documentation	79.479	64.990	1.223
relativity	188.814	132.214	1.428
domcasmurro	387.791	270.835	1.432
historyofscience	518.289	348.432	1.487
principia	835.542	554.949	1.506
elements	1.125.058	535.659	2.100
julioverne	1.341.852	939.534	1.428
formula1	1.739.656	489.612	3.553
warandpeace	1.904.546	1.107.980	1.719

Percebe-se que há uma certa tendência de quanto maior o tamanho do arquivo inicial, maior será a taxa de compressão. Isso ocorre devido ao fato de que quanto mais texto houver, maior será a chance de um prefixo já ter ocorrido anteriormente. Dessa maneira, esses prefixos podem ser aproveitados mais vezes para economia de bytes utilizados para guardar informação sobre um determinado trecho do texto.

Por exemplo, o arquivo “linguagensdeprogramação” é um texto referente a uma página da Wikipédia. Por ser um arquivo relativamente pequeno, sua taxa de compressão foi bem baixa, o que é um indicativo que não há tanta repetição de prefixos como desejado para que o algoritmo obtenha uma alta taxa de compressão.

Já o arquivo “formula1” é um bom exemplo para mostrar os efeitos da compressão em um texto com muita repetição de termos. Ele consiste na descrição de um banco de dados na linguagem SQL em que se repete muito o comando de inserção em uma tabela. Isso faz com que o prefixo referente a esse comando seja bastante comum, ocasionando assim na maior taxa de compressão obtida dentre os arquivos testados.

4) Conclusão

Os resultados obtidos permitem concluir que maiores taxas de compressão dependem muito da repetição de prefixos ao longo do texto, sendo que quanto maior o texto maior será a probabilidade de reaproveitamento desses prefixos. Assim, o algoritmo implementado aparenta ser bastante útil para comprimir textos maiores ou que apresentam muitos termos repetidos.

Além disso, a realização do trabalho prático foi uma boa oportunidade de aplicar os conceitos de árvore de prefixos e manipulação de sequências aprendidos durante a disciplina em uma importante área da computação: a compressão de dados.

5) Referências Bibliográficas

- SALOMON, David. Data Compression. The Complete Reference. Fourth Edition. Springer, 2007. 1092 p.
- SEDGEWICK, R.; WAYNE, K. Algorithms. Fourth Edition. Addison-Wesley Professional, 2011. 976 p.