TRABALHO PRÁTICO 1
Manipulação de sequências

RELATÓRIO
Algoritmos II

Denilson Martins Vieira Santos

1. Introdução

A proposta deste trabalho prático consiste no desenvolvimento de um compressor de arquivos voltado para a compressão de texto usando o algoritmo LZ78. Esse algoritmo consiste na utilização de um dicionário para verificar sequências já inseridas e dá como saída os caracteres a serem inseridos no final de uma sequência anterior já presente na trie.

Um pseudocódigo de como a compressão funciona é o seguinte:

- Crie um dicionário vazio
- Para cada caractere na sequência de entrada:
- Vá percorrendo os caracteres presentes no dicionário até não poder mais
- Insira esse último caractere no final da sequência percorrida
- Adicione essa sequência no dicionário
- Adiciona na saída do programa que foi inserido o caractere ‘c’ na sequência de índice ‘index’

Para fazer a descompressão, o algoritmo é semelhante

- Crie um dicionário vazio
- Para cada caractere ‘c’ e índice ‘index’ dos dados comprimidos:
- Encontre no dicionário a sequência de índice ‘index’
- Adicione essa sequência na saída
- Adicione ‘c’ na saída
- Adicione uma nova sequência que é composta pela sequência anterior mais o caractere ‘c’

Para o dicionário, será utilizado uma trie. Uma vantagem de se utilizar a trie é a possibilidade de inserir uma nova sequência apenas inserindo um nó nessa trie.

2. Implementação

2.1 Compressão

Para a compressão, é inicialmente feita a leitura do arquivo de entrada considerando a leitura de um arquivo binário. Pois um arquivo de texto também é um arquivo binário e esse algoritmo também é capaz de comprimir arquivos que não sejam de texto, embora para alguns casos, não seja muito eficiente.

Após isso é gerado os dados comprimidos. A implementação da compressão foi feita da seguinte forma:

É criado um vetor ‘nodes’ que é responsável por guardar os nós da trie. Cada nó da trie guarda duas informações: seu índice e o índice de seus filhos. Também é criado outro vetor ‘output’ que é responsável por guardar a saída da compressão.

Passo A: É aplicado o algoritmo de compressão anteriormente descrito com um pequena mudança. Essa mudança consiste em uma verificação se o byte sendo lido é o último byte da entrada. Caso isso seja verdade, se medidas para remediar não sejam tomadas, o algoritmo continuaria lendo bytes da entrada que já teria acabado até encontrar um valor que não estivesse na trie, gerando comportamento indefinido. Para isso não acontecer, se a verificação é verdadeira o algoritmo insere um novo termo na saída comprimida que não há problemas em ser redundante já que é o último byte da entrada e termina.

Passo B: Após isso, outra função recebe essa saída do algoritmo de compressão e a comprime mais ainda. Essa compressão é feita ao reduzir o tamanho dos inteiros sendo utilizados. O algoritmo de compressão gera pares de byte e inteiros sem sinal de 64 bits. O algoritmo de compressão reduz esses inteiros de 64 bits para o menor tamanho possível que ainda seja capaz de representar o maior inteiro. Assim é reduzido o espaço gasto e todos os inteiros são representados.

A saída desse algoritmo é um vetor de inteiros sem sinal de 64 bits que possui sua informação guardada nos bits desses inteiros. Nesse vetor os primeiros 7 bits representam um inteiro sem sinal que indica o número de bits gasto pelos inteiros. Vamos chamar esse valor de ‘s’. Após isso vem ‘s’ bits que representam um inteiro que indica o número de elementos do resultado da compressão. Vamos chamar isso de ‘n’. Após isso vem ‘n’ pares de um byte e inteiro de tamanho ‘s’ representado o byte e seu índice respectivamente.

Após isso, uma função recebe esse vetor e o salva no arquivo de saída.

2.2 Descompressão

O procedimento de descompressão consiste em inicialmente carregar os bytes comprimidos do arquivo de entrada. Após isso, esse bytes comprimidos passam por uma função para descomprimi-los, gerando os diversos pares de byte e inteiro de 64 bits. Essa função é responsável por inverter o processo feito no passo B anterior.

Para o algoritmo de descompressão principal foram tomadas algumas medidas diferentes do algoritmo de compressão. Para a descompressão, cada nó além de guardar seu índice e o índice de seus filhos, guarda também seu pai e o valor da aresta que conecta seu pai a ele mesmo. Foi necessário adicionar essas duas novas informações pois é necessário conhecer o valor do byte de cada nó e também é necessário percorrer a trie de baixo para cima para obter a sequência.

A implementação do pseudocódigo de descompressão foi feita da seguinte forma:

Para encontrar no dicionário a sequência de índice ‘index’, obtemos o nó de índice ‘index’ e percorremos a trie de baixo para cima até a raiz. A cada byte percorrido, esse byte é inserido em uma stack auxiliar pois esses bytes estarão invertidos e depois essa stack auxiliar é desempilhada na saída, invertendo os bytes que estavam invertidos, fazendo assim com que ficassem na ordem correta.
Depois disso é inserido o byte ‘c’ na saída e também é inserido um novo nó na trie para o novo caractere.

Após isso é obtido a sequência de bytes original e esse bytes são salvos no arquivo de saída.

3. Resultados

A seguir seguem os resultados de compressão do algoritmo para os arquivos de texto passados pelo professor:

Especificação:
Entrada: 6551
Saída: 4743
Compressão: 27.6 %

Os Lusíadas:
Entrada: 344538
Saída: 195228
Compressão: 43.34 %

Dom Casmurro:
Entrada: 409610
Saída: 231538
Compressão: 43.48 %

Constituição de 1988:
Entrada: 651790
Saída: 278282
Compressão: 57.31 %

Arquivos de **Lorem Ipsum**:

Entrada: 5909
Saída: 3831
Compressão: 35.17 %

Entrada: 13922
Saída: 8160
Compressão: 41.39 %

Entrada: 20039
Saída: 11403
Compressão: 43.1 %

Entrada: 73589
Saída: 33680
Compressão: 54.24 %

Entrada: 100314
Saída: 44300
Compressão: 55.84 %

Entrada: 100324
Saída: 44366
Compressão: 55.78 %

Entrada: 100336
Saída: 44611
Compressão: 55.54 %

Entrada: 1923394
Saída: 722488
Compressão: 62.44 %

Arquivos de código fonte:

src/compression.hpp:
Entrada: 273
Saída: 278
Compressão: -1.83 %

src/argReader.cpp:
Entrada: 346
Saída: 314
Compressão: 9.25 %

Makefile:
Entrada: 564
Saída: 496
Compressão: 12.06 %

src/argReader.hpp:
Entrada: 890
Saída: 714
Compressão: 19.78 %

runTests.sh:
Entrada: 1462
Saída: 1022
Compressão: 30.1 %

src/main.cpp:
Entrada: 2855
Saída: 1895
Compressão: 33.63 %

src/compression.cpp:
Entrada: 7360
Saída: 4197
Compressão: 42.98 %

Arquivos de executável **ELF**:

Entrada: 39448
Saída: 20601
Compressão: 47.78 %

Entrada: 131816

Saída: 64978
Compressão: 50.71 %

Entrada: 2455936
Saída: 1459846
Compressão: 40.56 %

Arquivo de vídeo mp4:

Entrada: 5356675
Saída: 6288513
Compressão: -17.39 %

Com os resultados é possível observar que quanto maior o arquivo, maior a tende a ser a compressão. No entanto para arquivos já fortemente comprimidos como arquivos de vídeo, a compressão passa a ser negativa pelo fato desses arquivos já removerem boa parte de suas sequências repetidas.