Instituto de Ciências Matemáticas e de Computação – USP

Henrique de S. Q. dos Santos

Witor M. A. de Oliveira

Relatório Projeto 1

Codificação de Huffman

São Carlos

2019

1. Introdução

Buscando uma solução geral para comprimir um dado arquivo de texto, foi criado um programa responsável por codifica-lo utilizando a **Codificação de Huffman** que, basicamente, constrói uma árvore binária baseada na frequência de caracteres e gera uma tabela com um código único para cada caracter, de forma que os caracteres mais frequentes serão representados por menos bits, e assim um novo arquivo seja gerado a partir desta tabela, novo arquivo sendo que este é uma compressão do arquivo anterior.

2. Desenvolvimento

Desenvolvemos o programa utilizando a linguagem C++ e utilizando a biblioteca padrão dela(bits/stdc++.h).

Primeiramente fizemos a estrutura de árvore, que além dos atributos comuns às outras estruturas de árvore convencionais, tem como dado principal um caractere e recebe mais um atributo correspondente à frequência do caractere.

No programa lemos um arquivo de texto como entrada, guardamos este texto em uma string, contamos a frequência de cada caractere da tabela ASCII simples e armazenamos esses caracteres com suas respectivas frequências, na forma de uma fila de prioridade de nós da árvore de Huffman.

Após isso, seguimos o algoritmo e retiramos os dois nós de menores frequências da min-heap, fazemos com que eles se tornem filhos de um novo nó alocado, que possui a frequência igual a soma das frequências de seus filhos, depois reinserimos este novo nó na heap. Repetimos estes passos até a heap possuir apenas um elemento, que será a árvore de Huffman.

Posteriormente, criamos a tabela de compressão, fazendo a busca em profundidade por todos os caracteres da tabela ASCII na árvore e armazenando o código de cada um dos caracteres encontrados na busca (os códigos são dados por uma string que recebe 0 para cada vez que a busca vai para um nó esquerdo da árvore e 1 para cada vez que vai para um nó direito). Depois já criamos também a tabela de descompressão apenas invertendo a posição da chave e do valor da tabela de compressão.

Terminada a parte de geração de códigos, criamos o arquivo comprimido, para isso, escrevemos os códigos dos caracteres do arquivo original, presentes na tabela de compressão, na mesma ordem.

Como os códigos estão no formato de string e não possuem tamanho fixo, fomos concatenando eles em uma string auxiliar de tamanho 8, transformando em um char esse 8 bits e escrevendo no arquivo de saída. Ainda que em alguns casos o último código não caiba por completo na string auxiliar,

o resto dele será escrito na próxima iteração do loop no programa. Isso também implica que caso o último conteúdo a ser escrito no arquivo seja menor do que os 8 bits da string auxiliar, poderíamos ter problemas na descompressão, para resolver isso, escrevemos um último byte no arquivo que representa a quantidade de bits inúteis do byte que for lido antes deste.

Já na descompressão do arquivo comprimido gerado, ele é completamente lido como uma string, e cada caracter dessa string (8 bits), tem seu valor convertido para um número binário representado por meio de um string. Esses valores em binário são concatenados em uma única string a medida que vão sendo convertidos.

Para identificar os códigos dos caracteres e ser possível a descompressão, a string binária é lida bit a bit (aqui nos referimos a bit como sendo um caracter na string binária), onde partimos do primeiro bit e buscamos ele na tabela de descompressão, caso ele não seja uma chave válida, formamos uma nova string dele com o próximo bit e realizamos a busca novamente. Esse processo se repete até ser encontrada uma chave válida na busca, quando isso acontece esse mesmo processo se repete, mas dessa vez partindo-se do bit posterior ao da última chave encontrada. Tudo isso é feito respeitando o número de bits úteis na string, que é **bu = ts - (8+bi)**, onde bu = número bits úteis na string, ts = total de bits da string e bi = bits inúteis do último caractere, o 8 representa o caractere utilizado para indicar a quantidade de bits inúteis do último caractere escrito.

Tendo terminada a descompressão, geramos um novo arquivo no qual colocamos os dados descomprimidos, para facilitar a verificação da corretude do algoritmo pelo avaliador e mostramos na tela o tamanho do arquivo original e do arquivo comprimido em bytes.

3. Entrada e Saída

O programa será responsável por receber um arquivo de texto e comprimi-lo utilizando a codificação de Huffman. Ao executar o programa, o nome do arquivo a ser comprimido deverá ser digitado e, logo em seguida, o nome para o arquivo gerado. Após isso, o arquivo que foi comprimido é descomprimido em um arquivo de texto com nome sendo igual ao nome do arquivo de entrada + “PraComparar.txt”, para que o avaliador possa utilizar algum programa (diff ou meld por exemplo) para comparar o arquivo original com o descomprimido e ver se o conteúdo é o mesmo. Após a finalização da execução do programa, o tamanho do arquivo original será exibido em bytes na tela, junto com o tamanho do arquivo comprimido;

4. Conclusão

Ao fim desse trabalho pudemos entender como funciona o código de Huffman aplicado para compressão de arquivos, além de ver na prática a

aplicação de uma estratégia gulosa, gerar códigos menores para o caracteres mais frequentes de forma que no conteúdo gerado pelos códigos apareçam menos bits, ou seja, para que conteúdo original sofra uma compressão de tamanho, sem perder dados.