Universidade Federal do Espírito Santo - Departamento de Informática

2° Trabalho Prático

Compactador e Descompactador

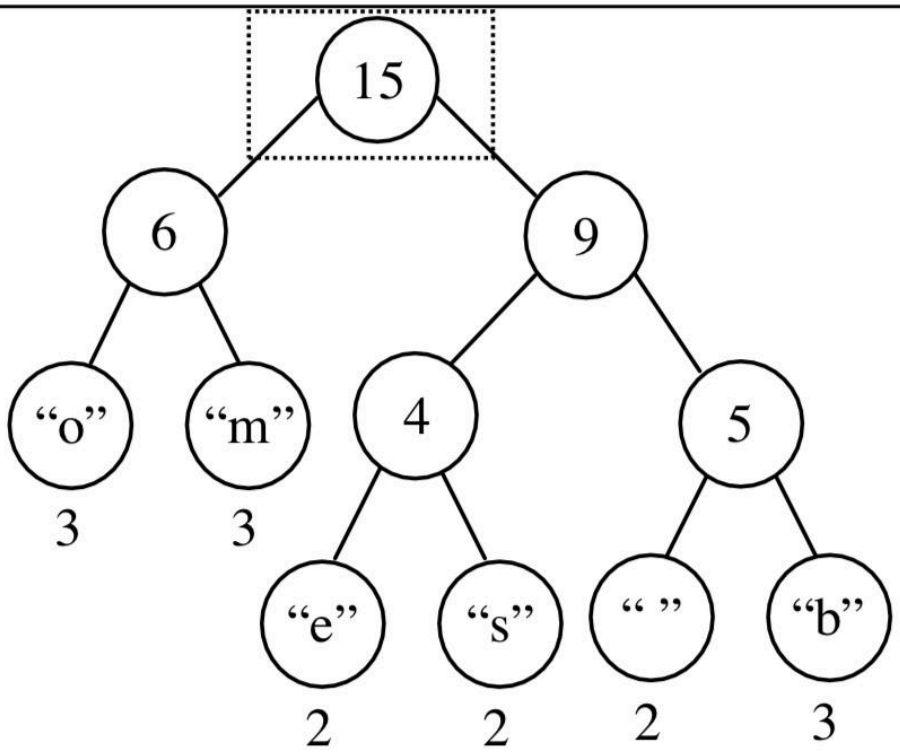
Estruturas de Dados (INF15974) - 2024/1

27 de julho de 2024

Alunas: Aline Mendonça Mayerhofer Manhães e Marcela Carpenter da Paixão

Introdução: O trabalho consiste em criar um compactador de arquivos a partir da codificação de Huffman. Devemos ler 1 byte de cada vez, e a partir da frequência de cada byte, montar uma árvore de codificação.

Como a árvore é organizada a partir da frequência, cada byte terá um número menor de bits que o representa. Isso possibilita a criação de um novo arquivo com menos bytes para representar o mesmo arquivo original. Esse arquivo menor é o arquivo compactado (.comp). Assim, para compactarmos o arquivo original lido, devemos percorrer a árvore até achar o byte que deve ser compactado e o seu número de binário de conversão. Esse número é criado a partir da seguinte lógica: para cada arco percorrido até chegarmos na folha que armazena o byte desejado acrescentamos um bit. Se esse arco for da esquerda, o bit acrescentado é 0. Se for o da direita, acrescentamos o bit 1.



Esse é um exemplo de árvore pós algoritmo de Huffman. Na árvore acima, a letra “s” teria a codificação 101, pois para chegar até ela percorremos: direita-esquerda-direita, a partir do nó inicial.

Uma vez que temos o arquivo compactado, devemos descompactá-lo retornando ao arquivo original. Para isso, remontamos a árvore de codificação e lemos o arquivo compactado. Dessa forma, usamos os códigos lidos do arquivo compactado e andamos pela árvore até acharmos um byte, que é escrito no arquivo com a extensão original.

Uma vez finalizado esse processo, o arquivo original será igual ao último arquivo criado.

Implementação: Os arquivos foram estruturados em:

- *bitmap.h/ bitmap.c*: arquivos fornecidos pela professora para manipular bits de bytes de forma individual.

- *binario.h/ binario.c*: nesses arquivos estão as funções de manipulação de arquivos em formato binário. Todos os arquivos foram tratados como binário, tanto o arquivo original a ser compactado, quanto o arquivo compactado em si. O mesmo vale para a recriação do arquivo original, ao descompactar.

- *huffman.h/ huffman.c*: nele estão as funções de aplicação do algoritmo de Huffman, feito por meio de uma lista de árvores (contendo um unsigned char e a sua frequência). Essa lista foi ordenada com base nas frequências e a partir da lista ordenada foi criada uma árvore completa com todos os unsigned char utilizados.

- *arvore.h/ arvore.c*: arquivo usado para as funções básicas de árvore e funções extras de codificação e decodificação que utilizem a árvore. Cada árvore contém um unsigned char, a frequência desse byte, um bitmap e suas árvores filhas, da esquerda e da direita. Essa árvore, depois de passar pelo algoritmo de Huffman, pode criar a conversão de cada unsigned char lido em um arquivo passado por parâmetro, permitindo converter esses bytes para um arquivo binário compactado, utilizando menos bits que o formato original de um byte. Para isso, é

preciso armazenar o caminho percorrido para chegar a cada folha da árvore, onde estão os unsigned char a serem convertidos, dentro de um bitmap (fornecido pela professora).

- *compactador.c*: é a main utilizada para compactar arquivos. Nela temos todo o algoritmo de compactação desenvolvido no trabalho.

- *descompactador.c*: é a main utilizada para descompactar arquivos. Nela temos todo o algoritmo de descompactação desenvolvido no trabalho.

- *Makefile*: ao rodar “make compacta” no terminal executa a compilação através do comando do terminal “gcc bitmap.c arvore.c huffman.c binario.c compactador.c -o compacta”, que compila todos os .c citados, transformando em .o, e depois gera o executável “compacta”. O mesmo ocorre com “make descompacta”, comando “gcc bitmap.c arvore.c huffman.c binario.c descompactador.c -o descompacta” e executável “descompacta”. Além disso, para executar o programa, o escolha o executável que será utilizado e rode no terminal com o nome do arquivo que deseja compactar/descompactar.

As relação às funções do programa, as principais, apresentadas nas “*mains”*, foram:

- *criaListaDeArvores*: inicia o algoritmo de Huffman, criando uma lista de árvores, nas quais estão guardados os unsigned chars lidos no arquivo de parâmetro e suas frequências. A função retorna a lista ordenada com base nas frequências.

- *criaArvoreUnica*: função que aplica efetivamente o algoritmo de Huffman, gerando, no fim, apenas uma árvore que contém todas as outras ao longo de suas subarvores esquerda e direita.

- *percorreArvoreBM*: função que, por meio de recursão, vai gerando os bitmaps (valores de conversão para binário) dos possíveis caminhos percorridos na árvore. Para cada folha da árvore (ou seja, um dos bytes lidos), guardamos esse bitmap.

- *escreveArvoreBinario*: transforma a árvore formada ao longo do algoritmo em binário por meio de bitmap. Se um dado elemento da

árvore não for folha, escrevemos 0. Se for folha, escrevemos 1 e o binário correspondente a esse unsigned char. Depois de convertida em binário, a árvore é escrita no arquivo compactado.

- *escreveBinario*: para cada unsigned char do texto original, procuramos esse byte na árvore e recebemos seu bitmap. Dessa forma, o escrevemos no bitmap final do arquivo compactado. Ao fim, esse bitmap final é escrito no arquivo compactado.

- *leArvoreBinario*: utilizamos o algoritmo inverso da função de escrita de árvore no binário. Para cada bit 0, criamos uma árvore com árvores vazias que serão formadas em recursão. Se o bit for 1, convertemos os bits lidos para unsigned char e o guardamos.

- *leBinario*: uma vez que temos a árvore reconstruída, para descompactar devemos ler o arquivo compactado e percorrer a árvore com base nas coordenadas lidas. Ao achar uma folha, esse é o unsigned char procurado e o escrevemos no arquivo descompactado.

Conclusão: Nós duas achamos o trabalho muito interessante, porém mais complexo para entender que o primeiro trabalho da disciplina. Acreditamos que evoluímos enquanto programadoras e fixamos o conteúdo aprendido em Estrutura de Dados até agora.

Onde encontramos maior dificuldade foi para entender o TADBitMap e pensar numa lógica que utilizasse ele para criar um código para cada unsigned char armazenado na árvore de codificação. Além disso, também fomos desafiadas ao ler arquivos de imagem e arquivos grandes, com mais de um mega, tendo então que pensar em uma forma de acrescentar novos bitmaps à medida que eles fossem lotando.

Bibliografia: O trabalho foi feito com base nos conhecimentos adquiridos através da disciplina de Estrutura de Dados, ministrada pela professora Patrícia Dockhorn Costa, além do algoritmo de Huffman e da utilização do TADBitMap fornecido. Não foram utilizados sites de pesquisa.