UFES

Universidade Federal do Espírito Santo - Centro Tecnológico

Departamento de informática

Estrutura de dados I (INF09292) – Turma EC

Professora: Patrícia Dockhorn Costa

Estrutura de Dados: Compactador/Descompactador de Arquivos

Humberto Giuri Calente

Sérgio Vago Rodrigues de Melo

Dezembro/2018

Sumário

1. Introdução

Em nosso segundo trabalho, foi proposta a construção de um compactador e um descompactador de arquivos utilizando o Código de Huffman (Árvore de Huffman), algoritmo para compressão de arquivos, principalmente textos. O codificador Huffman cria uma árvore ordenada com todos os símbolos e a frequência com que aparecem. Os nós são construídos recursivamente começando com os símbolos menos frequentes. Sucessivamente, os dois símbolos de mais baixa frequência de aparecimento são retirados da lista e unidos a um núcleo cujo peso vale a soma das frequências dos dois símbolos. O símbolo de mais leve é atribuído à ramificação um, o outro à ramificação zero e assim por diante, considerando cada núcleo formado como um novo símbolo, até obter só um núcleo, chamado raiz. A figura 1, nos mostra o funcionamento da Codificação de Huffman.

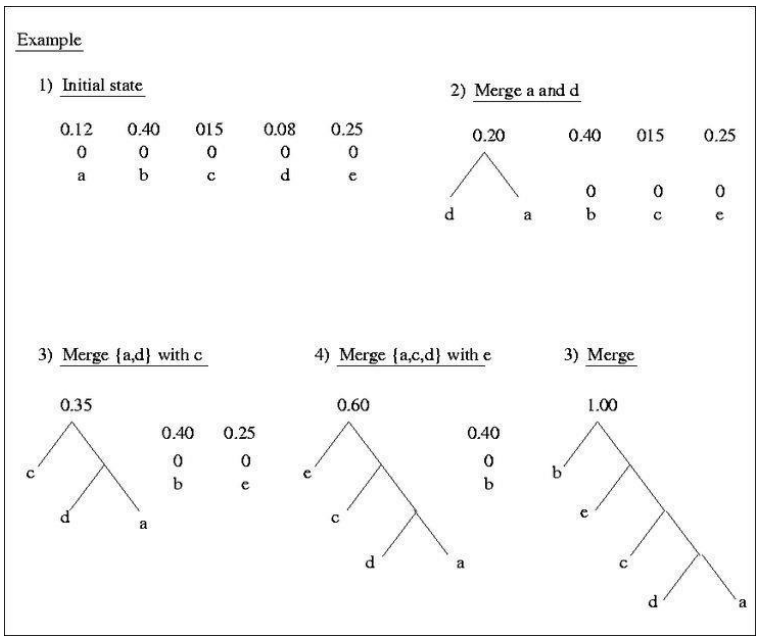

Figura 1: Criação da árvore de Huffman

2. Implementação

1. arvore.c:
    - Contém a estrutura arvore que engloba 4 parâmetros: int freq, char c, Arvore* direita, Arvore* esquerda. Esses parâmetros definem uma arvore binária que carrega consigo um caracter como informação e quantas vezes esse caracter aparece. As principais funções desse TAD serão apresentadas logo abaixo.
    - A função Arvore* IniciaArvore (int freq, char c) aloca um espaço para uma nova árvore e retorna essa árvore com os campos freq e c inicializados acompanhando os parâmetros que são passadas para ela.
    - A função void SomaFrequenciasAlteraValor (Arvore* arv1, Arvore* arv2, Arv* 3) é usada durante o processo de criação da árvore de Huffman onde ela soma a frequência de duas árvores e transfere para uma terceira árvore, conforme o algoritmo de Huffman ilustrado acima.
    - A função int CalculaAltura(Arvore* arv) recebe uma árvore e calcula e retorna o tamanho dela.
    - A função int EhFolha(Arvore* arv) recebe uma arvore e analisa se ela é nó folha ou não, retornando 1 caso seja e 0 caso contrário.

2. lista.c:
    - Contém duas estruturas, lista e célula, que definem uma lista simplesmente encadeada com sentinela, com a diferença de que dentro de cada célula contém uma árvore. Essa característica é fundamental na hora de construir a árvore de Huffman, que começa com uma lista de células que contém árvore nelas. As principais funções desse TAD serão apresentadas logo abaixo.
    - A função Lista* IniciaLista () aloca um espaço para uma nova lista, nesse caso a sentinela da lista, e retorna a lista com seus ponteiros primeiro e último apontados para NULL.

- A função void InsereNaLista (Lista* lista, Arvore* arv) recebe uma lista e uma arvore como parâmetros, aloca uma nova célula, faz o campo arvore da célula receber árvore que foi dada com parâmetro e insere essa célula na lista.
- A função Lista* CriaListaArvore (int* vet) recebe um vetor de inteiros como entrada e transfere os dados contidos nesse vetor para uma lista que a própria função tomará conta de criar, retornando a lista desejada.
- A função void OrdenaLista (Lista* lista) recebe uma lista como parâmetro e utiliza um algoritmo parecido com o bubble sort para ordenar a lista conforme a frequência do caracter que cada árvore de cada célula contém.
- A função Arvore* RemoveCelulaArvoreDaLista (Lista* lista, Arvore* arv) recebe uma lista e uma árvore como parâmetro e retira a célula que contém esse árvore da lista. Essa função é fundamental no momento de criar a árvore de Huffman, retirando as árvores que terão suas frequências somadas e criadas em uma árvore.
- A Função Tabela* CriaHuffman (Lista* lista) recebe uma lista como parâmetro e dentro dela faz todos os passos necessários para a criação da árvore de Huffman e ainda cria a tabela com os valores que cada caracter assume conforme o algoritmo de Huffman.

3. tabela.c:
    - Contém uma estrutura chamada tabela, que contém os campos char c, unsigned char vBin, int freq e Tabela* Prox, que são um caracter normal (char c), o valor binário que esse c assume conforme a árvore de Huffman (unsigned char vBin), quantas vezes esse caracter aparece (int freq) e um ponteiro para a próxima tabela (Tabela* Prox). Esse campo Prox configura uma lista sem sentinela e simplesmente encadeada. As principais funções desse TAD serão apresentadas logo abaixo.
    - A função Tabela* NovaTab () aloca um espaço para uma tabela. Depois, aloca um espaço para o campo vBin e atribui o char ‘\0’

nele, e também atribui para o campo frequência o valor 0 e faz o campo Prox apontar para NULL.

- Tabela* PreencheTabela (Arvore* arv, Tabela* tab) recebe uma árvore e uma tabela, ela roda a árvore em busca dos nós folhas, que conterão os caracteres lidos, após o processo de Huffman, e salva esses caracteres e suas frequências em uma lista encadeada.
- A função void DescobreValorBinario (Arvore* arv, int id, int cont, char* str, Tabela* tab) roda toda a árvore de Huffman procurando um nó folha e salvando o caminho que fez até essa folha, se for pra esquerda é 0 e se for pra direita é 1 criando uma sequência que será o campo vBin do caracter que foi achado naquela folha.
- A função Tabela* CriaTabelaBinaria (Arvore* arv) recebe uma árvore e basicamente usa a função preenche tabela e retorna essa tabela.

4. compactador.c:
    - É como um cliente que usará TADs já implementados para conseguir imprimir um arquivo de saída que será o arquivo compactado. O compactador. c possui algumas funções que serão apresentadas abaixo.
    - Void IniciaVetor (int* vet, int a) recebe um vetor de inteiros e um inteiro. Essa função inicia um vetor de 257 posições e atribui todos os campos dele com zero. A posição de cada campo do vetor servirá como o caracter que esse valor representa na tabela asc e o valor que ele guarda será a frequência do caracter.
    - A função int ContaTamanhoCabeçalho (Arvore* a) essa função roda a árvore contando o número de nós folhas e retorna essa quantidade para auxiliar na criação do cabeçalho, que será a base para a descompactação.
    - A função void CriaCabeçalho (Arvore* arv, bitmap* bm) recebe uma arvore e um bitmap e cria o bitmap do cabeçalho.

5. descompactador.c:

- É como um cliente que usará as informações dos TADs e a informação do cabeçalho criado para conseguir descompactar o arquivo, levando em conta uma condição de parada determinada pelo grupo, que é ´~. Quando o pedido de descompressão for executado, o programa irá ler o arquivo até encontrar essa sequência e irá parar de descomprimir. Logo, o arquivo para descompressão não poderá ter exatamente essa sequência por eventual azar, sem que seja o final do arquivo. O descompactador.c possui uma estrutura chamada cont que contém apenas um inteiro e uma função importante que será apresentada abaixo.
- A função Arvore* CriaArvore (char* c, Arvore* arv, int tam, Cont* cont, int id) Analisa o cabeçalho que foi impresso no arquivo durante a compactação e remonta a arvore de Huffman obtida no processo de compactação.

3. Conclusão

O trabalho serviu para mostrar possibilidades de uso de estrutura de dados para compressão e descompressão de alguns arquivos, utilizando árvores. Pudemos verificar, em nosso trabalho, que a compressão dos arquivos se dá de maneira mais acentuada nos arquivos de texto, baseando-se nos testes com textos e imagens.

A maior dificuldade do grupo foi, com certeza, entender o bitmap e aprender a usar ele, tendo em vista que isso só ocorreu depois de muito tempo após começar o trabalho e acabou problematizando o tempo para desenvolver o trabalho.

Outra grande dificuldade do grupo foi na hora da condição de parada, já relatado nesse documento, tornando falho o programa caso a sequência ´~ apareça sem que seja o final do arquivo.

4. Referência bibliográficas

- CELES, Cerqueira e Rangel, Introdução a Estrutura de Dados, Editora Elsevier, 2004.