Universidade Federal do Espírito Santo

UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO

CENTRO TECNOLÓGICO

DEPARTAMENTO DE INFORMÁTICA

Alan Herculano Diniz

Rafael Belmock Pedruzzi

Compactador de Arquivos: Trabalho para a disciplina de Estrutura de Dados I

Vitória

2018

Alan Herculano Diniz
Rafael Belmock Pedruzzi

Compactador de Arquivos: Trabalho para a disciplina de Estrutura de Dados I

Trabalho para a disciplina de Estrutura de Dados I do curso de Ciência da Computação da Universidade Federal do Espírito Santo.

Professor(a): Patrícia Dockhorn Costa
Disciplina: Estrutura de Dados I
Turma: INF09292

Vitória
2018

Sumário

1 – Introdução

Nos dias atuais, arquivos e dados possuem tamanho e complexidade muito grandes para serem enviados e recebidos sem nenhuma modificação. Portanto, é preciso de um método de compactar arquivos para que seu transporte seja viável e eficiente. A partir desse fato, têm-se a base deste segundo trabalho da disciplina de Estrutura de Dados I, em que o objetivo é programar um compactador na linguagem de programação C que possa criar uma versão compactada de um arquivo de entrada e que possa descompactar arquivos compactados.

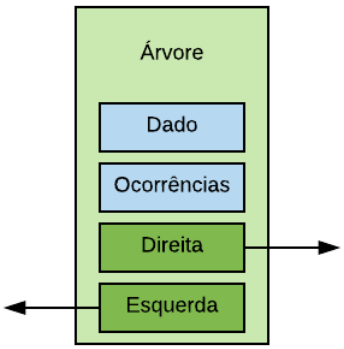
Para criar esse compactador, será utilizado o algoritmo de Huffman, que é um método que usa o número de ocorrências dos bytes e suas probabilidades de ocorrerem para criar um arquivo com tamanho reduzido.

2 – Implementação

Para utilizar o algoritmo de Huffman, é necessário implementar certos tipos abstratos de dados que auxiliaram na construção do compactador. Nos próximos subitens, serão listados e explicados esses TAD's.

2.1 – Árvore

O tipo árvore é um dos principais tipos utilizados na construção do compactador, já que ele é utilizado para codificar os bytes de tal forma que será possível construir o arquivo compactado. Ela possui os campos de ponteiros para as subárvores da direita e da esquerda, já que é uma árvore binária, outro para manter o dado e outro para manter a ocorrência desse dado. O diagrama abaixo representa a estrutura de dados:



Também são definidas funções que manipulam variáveis do tipo árvore, entre as principais, estão as seguintes:

Funções de criação e destruição na memória: essas funções são responsáveis por alocar dinamicamente e apagar da memória o espaço com o tamanho necessário para uma árvore. Na verdade, todo TAD possui essas funções, entretanto, a árvore possui um diferencial: ela possui duas funções de alocação dinâmica de espaço de memória, uma para criar um nó que possui subárvores e outra para criar uma nó folha (que não possui nenhuma subárvore). Note que somente nós folha possuem referência para dados do arquivo que será compactado, isso é importante para o algoritmo de Huffman (que será explicado mais a frente).

Também há funções que retornam os dados existentes em um nó de uma árvore (dado, número de ocorrências e subárvores). Novamente, todo TAD possui esse tipo de funções.

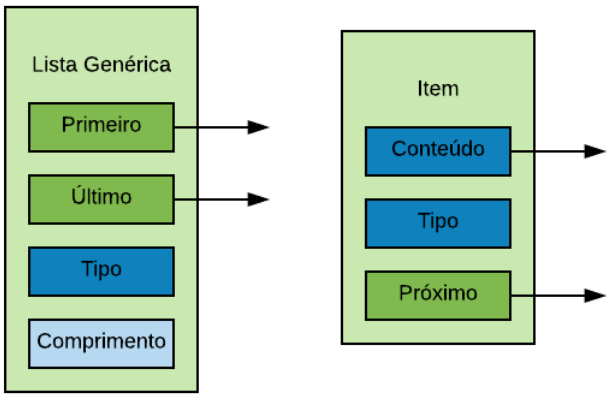
Porém, essas funções não são as únicas que são relacionadas com o tipo árvore. Existe uma função que verifica se um nó de uma árvore é ou não um nó folha, outra que verifica se um certo dado está ou não na árvore, outra que retorna uma lista com todos os nós que pertencem ao caminho entre o nó raiz e outro nó dado com entrada e uma função que imprime a árvore.

2.2 – Lista Genérica

Esse TAD é usado para a construção da árvore binária que codifica os dados de um arquivo para criar uma versão compactada dos dados.

A lista usada nessa implementação do código de Huffman é uma implementação genérica, ou seja, ela pode ser aplicada e reutilizada para qualquer tipo de dado. Foram implementados dois tipos de dados, um para a lista (o sentinela da lista, para ser mais específico) e outro para o item de uma lista, o qual aponta para o conteúdo desejado.

A lista possui os seguintes campos: ponteiros para o primeiro e o último item da lista, uma tag com o tipo de dado que a lista possui (para evitar que uma lista possua itens de tipos diferentes) e a quantidade de itens na lista. Enquanto isso, o item possui os seguintes campos: um ponteiro opaco para o conteúdo do item, uma tag com o tipo do conteúdo e um ponteiro para o próximo item da lista. Os diagramas abaixo mostram isso:

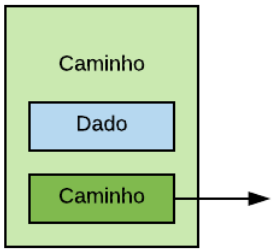


Existem funções para alocação dinâmica de uma nova variável de algum dos dois tipos e também de sua destruição na memória. Também há funções que verificam certas

condições tanto da lista quanto do item e também existem aquelas que manipulam variáveis desse tipo, assim como também existem funções que retornam os conteúdos dos campos dessas variáveis.

2.3 – Caminho

Esse TAD é um simplesmente um tipo auxiliar da árvore. Basicamente, ele é uma adaptação da lista genérica explicada a pouco em que o conteúdo da cada item é um bit. Essa lista representa o caminho do nó raiz de uma árvore de Huffman até uma folha qualquer de entrada.



2.4 – Compactador

O compactador não é um TAD, mas sim uma biblioteca de funções que são usadas para compactar um arquivo. Essa biblioteca possui três funções principais: uma função que monta uma árvore de Huffman, outra que constrói um arquivo compactado a partir de um dado arquivo e outra que descompacta um dado arquivo compactado.

Para criar a árvore de Huffman, primeiramente é preciso calcular o número de ocorrências de cada dado no arquivo e, então, será criado um nó de árvore para cada dado e seu número de ocorrências. Depois disso, todos esses nós serão colocados em uma lista. Essa lista será ordenada de forma crescente em relação ao número de ocorrências.

Depois disso, o seguinte processo é realizado até que só exista uma árvore na lista: as duas primeiras árvores da lista são selecionadas e colocadas como subárvores de uma nova árvore, que terá como peso (número de ocorrências) a soma dos pesos de

suas subárvores, que serão retiradas da lista mas não serão apagadas da memória. Enfim, cada caractere terá um código que será a representação de seu caminho da raiz da árvore até seu nó em binário (0 para a esquerda e 1 para a direita).

Depois da construção da tabela de compactação, o arquivo compactado será construído. Isso é feito lendo os dados do arquivo original e para cada dado lido, a sua codificação será colocada no bitmap que será criado e, só depois disso, é que o bitmap será impresso no arquivo compactado. Entretanto, é preciso tomar cuidado com certos detalhes: a árvore de codificação precisa estar no cabeçalho do arquivo compactado. Também é preciso tratar os bits extras que são postos pelo sistema operacional, já que a menor unidade de dados que ele trabalha são bytes, e não menos que isso.

Para descompactar o arquivo, é necessário primeiramente colocar o arquivo de entrada num bitmap. Nele, os três primeiros bits são usados para ver quantos bits estão sobrando no mapa, para evitar que o programa tente ler esses bits quando ele for realmente descompactar o arquivo. Depois disso, a árvore de Huffman vai ser recriada para que o programa tenha a referência de cada código de cada caractere. Então, ele vai começar a ler cada bit restante do mapa para ver qual caractere deve ser impresso no arquivo. Isso é feito lendo o bit do mapa e vendo o caminho que deve ser feito, se o nó atual for folha, é porque é um caractere e então o loop começa outro ciclo. Isso é feito até chegar nos bits sobressalentes.

3 – Conclusão

Com o desenvolvimento desse programa, percebe-se que existem várias aplicações importantes para as implementações de árvore binária, e que também é necessário saber e dominar os conceitos e técnicas para que os programas criados sejam criados da forma mais otimizada possível.

4 – Bibliografia

– Celes, Cerqueira e Rangel. Introdução a Estruturas de Dados, Editora Campus.