Trabalho Prático – AEDS II
Gustavo Cavalieri Fernandes – Universidade Federal de Minas Gerais
Documentação
Trabalho Prático 2 – 1o Semestre 2011

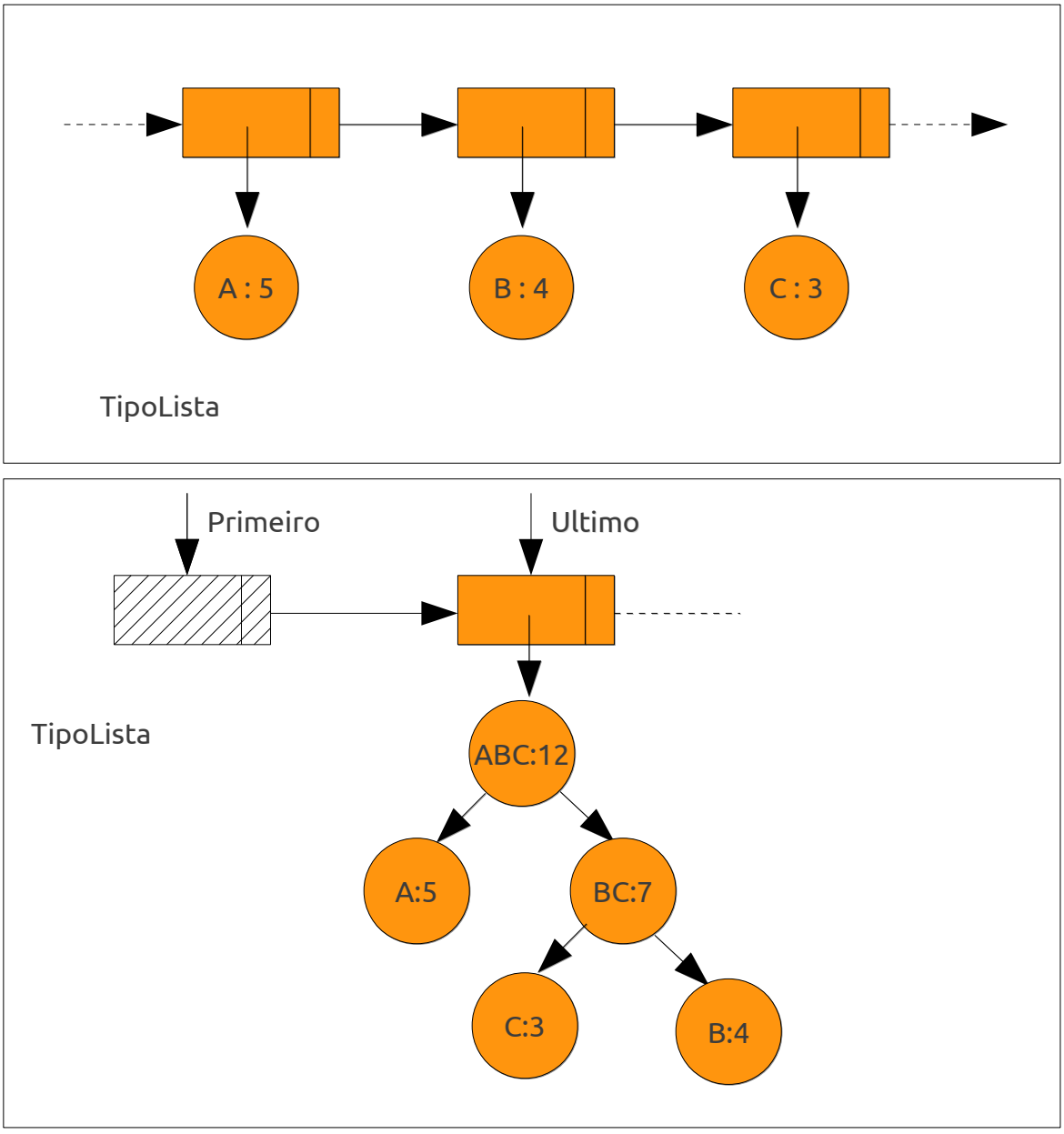
1. Introdução

O objetivo desse trabalho foi implementar um compressor de arquivos utilizando o método de Huffman. Deseja-se também praticar os conceitos de listas encadeadas e árvores binárias que são de extrema importância para o curso de Ciência da Computação.

2. Implementação:

Estrutura de Dados

O trabalho foi implementado através de uma Lista Encadeada onde cada célula contém uma árvore binária. Cada caractere lido do arquivo é guardado na raiz de uma árvore binária em uma célula. A figura abaixo demonstra bem a estrutura utilizada:



Ao final da execução do algoritmo, a lista terá duas células: A cabeça(Nula) e a árvore de Huffman.

Funções e Procedimentos:
O TAD da Árvore Binária possui as seguintes funções:

void insereitem(Ponteiro *a, TipoItem t); Recebe um item e o insere na árvore apontada por **a**

void inserehuffman(TipoArvore *a, Ponteiro t1, Ponteiro t2); Recebe uma nova raiz de árvore e direciona os ponteiros(esquerda, direita) da raiz para os itens recebidos. A função também guarda na raiz os novos caracteres inseridos na árvore.

void learvore(FILE *fp , char **buffer, Ponteiro a); Recebe o buffer e percorre o caminho da árvore ae que o caractere seja achado. Caso o caractere não seja encontrado, o resto é sempre devolvido ao buffer.

void escrevearvore(Ponteiro a , unsigned char c, char ** buffer); Recebe um caractere **c**, percorre a árvore construida e guarda o caminho dos bits em um buffer.

void freearvore(Ponteiro *a); Recebe a árvore criada e desaloca a memória usada para armazenar a mesma. A função desaloca também as strings usadas em cada Nó.

O TAD da Lista Encadeada possui as seguintes funções:

void FazListaVazia(TipoLista *l); Inicializa a lista com o primeiro elemento sendo a cabeça(Nula).

void inserelista(TipoLista *l , TipoItem t); Insere uma nova célula no final da lista e chama a função insereitem para a inserção do item na árvore.

void crialista(TipoLista *l); Cria a lista inserindo todos os elementos do vetor auxiliar utilizado para a primeira leitura do arquivo. Cada nova célula e sua árvore são inseridos no final da lista.

void removemenor(TipoLista *l , Ponteiro *n); Remove o primeiro elemento da lista, que é o elemento de menor frequência. Guarda a árvore da célula removida em um apontador ***n**.

void inserefinal(TipoLista *l, int freq); Insere a nova subarvore de Huffman no final da lista caso sua frequência seja a maior.

void insereordenado(TipoLista *l , Ponteiro t1, Ponteiro t2); Insere a nova árvore criada de maneira ordenada na lista, de acordo com sua frequência. Chama ainda a função **inserehuffman** para a inserção dos nós/folhas na nova raiz.

void freelista(TipoLista *l); Desaloca o primeiro(cabeça), o último(Árvore de Huffman) elemento da árvore e da free também no vetor auxiliar de leitura.

O TAD de procedimentos possui as seguintes funções:

void primeiraleitura(FILE *fp , TipoLista *l); Lê cada caractere do arquivo e sua frequência e guarda no vetor auxiliar da lista.

int pesquisacaractere(TipoItem *t,int n, unsigned char c); Pesquisa se o caractere já foi inserido no vetor auxiliar. Retorna **TRUE** or **FALSE.**

void ordenaauxiliar(TipoLista *l); Ordena o vetor auxiliar da lista pelo método ShellSort.

void huffman(TipoLista *l); Realiza o algoritmo de Huffman chamando os procedimentos: **removemenor** e **insereordenado.**

void segundaleitura(FILE *fp , TipoLista *l); Inicialmente escreve a tabela no arquivo e em seguida lê os caracteres do arquivo de entrada novamente. A partir do caractere, a árvore é percorrida e o caminho dos bits anotado em um buffer que será escrito quando for maior ou igual a 8 bits.

void escrevetabela(FILE *fw , TipoLista *l); Escreve os caracteres e suas respectivas frequências do vetor auxiliar no arquivo.

void tratabuffer(char ** buffer, FILE *fw); Recebe o buffer utilizado e imprime a sequencia de bits lida da arvore no arquivo como um caractere de 1 Byte. O resto dos bits não imprimidos é retornado para o início do buffer.

void freeall(TipoLista *l); Desaloca toda memória usada, chamando as funções **freearvore** e **freelista.**

void imprimebytes(char *imprime, FILE *fw); Converte o número binário de 1 byte para decimal, e imprime o decimal como caractere no arquivo.
void imprimeresto(char *resto , FILE *fw); Imprime o resto do buffer após a leitura de todo o arquivo.

void letabela(FILE *fp , TipoLista *l); Le os caracteres e frequencias do arquivo e guarda no vetor auxiliar da lista.

void descomprime(FILE *fp , Ponteiro a); Lê os caracteres do arquivo comprimido. Em seguida chama os procedimentos **convertedecimal e learvore** para converter o caractere novamente para a sequência de bits e o caminho da árvore ser refeito.

void convertedecimal(int c, char **caminho, int resto); Converte um número decimal para binário.

Programa Principal:

- Comprime:

O programa principal abre os arquivos de leitura e em seguida chama as funções dos TAD na seguinte ordem e procedimentos:
1.Inicializa a Lista → função FazListaVazia()
2.Faz a primeira leitura do arquivo → função primeiraleitura()
3.Ordena o vetor auxiliar → Função ordenaauxiliar()
4.Cria a lista com o vetor auxiliar → Função crialista()
5.Cria a árvore de Huffman→ Função huffman()
6.Faz a segunda leitura do arquivo para a compressão → Função segundaleitura()

- Descomprime

O programa principal abre os arquivos de leitura e em seguida chama as funções dos TAD na seguinte ordem e procedimentos:
1.Inicializa a Lista → função FazListaVazia()
2.Le a tabela de caracteres do arquivo → função letabela()
3.Recria a árvore de Huffman → Função huffman()
4.Descomprime o arquivo → Função descomprime()

Organização do Código, Decisões de Implementação e Detalhes Técnicos:

- O código está dividido em 8 arquivos, sendo 5 sources(.c) e 3 headers(.h).O header's arvore.h , lista.h e procedimentos.h guardam a estrutura e as funções da árvore, da lista e dos procedimentos respectivamente. Já os sources arvore.c , lista.c, procedimentos.c ,main.c implementam as funções e procedimentos.
- Os valores definidos para as constantes BYTE, MAXBUFFER TRUE e FALSE foram : 8 30 , 1 , 0 respectivamente.
- O compilador utilizado foi o GCC (GNU project C and C++ compiler) em um ambiente operacional Linux.

3.Análise de Complexidade

Definimos: **t** o número total de caracteres presente no arquivo original, **u** o numero de caracteres imprimidos na compactação, **n** o número de caracteres distintos do arquivo original.

void insereitem(Ponteiro *a, TipoItem t); Levando em conta o número de atribuições, a função faz sempre a inserção na raiz. Assim a função é **O(5) = O(1).**

void inserehuffman(TipoArvore *a, Ponteiro t1, Ponteiro t2); A função compara as strings de caracteres presentes nos nós recebidos. Ai temos 4 possibilidades. Sendo a e b os nós recebidos:(a é null e b é null , a não é null e b é null, a é null e b não é null, e ambos não são null). Levando em conta o número de comparações a função é então no pior caso O(4) = O(1), que também corresponde ao melhor caso. Função **O(1).**

void learvore(FILE *fp , char **buffer, Ponteiro a); Para um caminho de **b** bits recebido(8≤**b**≤30), realiza uma interação com 3 comparações. Em seguida guarda o resto de b-k bits no buffer, onde k é o número de bits usados para achar o caractere na árvore. Portanto temos uma função de melhor caso O(8.3) e pior caso O(30.3). Ambos os casos definimos a função como **O(1).**

void escrevearvore(Ponteiro a , unsigned char c, char ** buffer); A função percorre não recusivamente a árvore de **n** elementos na busca por um caractere. O pior caso realiza uma comparação a mais que o melhor caso. Sendo o número de comparações uma constante(**c) ,** e o número de interações total igual a n(log(n)), por ser uma busca em árvore binária, a função é **O(c.nlog(n)) = O(nlog(n)).**

void freearvore(Ponteiro *a); A função percorre toda a árvore recursivamente para desalocar a memória usada. Em cada chamada ela faz 3 comparações. Como nossa árvore possui **n** elementos, a função é **O(3n) = O(n).**

void FazListaVazia(TipoLista *l); Realiza quatro atribuições, logo a função é **O(4) = O(1).**

void inserelista(TipoLista *l , TipoItem t); Insere um elemento no final da lista(realizando 4 atribuições) e insere o elemento na árvore pelo procedimento **insereitem(O(1)).** Portanto a função é **O(1).**

void crialista(TipoLista *l); Insere os **n** caracteres distintos do arquivo por meio de uma interação e a função **inserelista(O(1)).** Portanto a função é **O(n).O(1) = O(n).**

void removemenor(TipoLista *l , Ponteiro *n); Remove o primeiro elemento da lista e guarda a árvore em um apontador. A função realiza apenas atribuições e free(), sendo assim **O(1).**

void inserefinal(TipoLista *l, int freq); Insere a nova árvore de Huffman criada por meio de 5 atribuições. Função **O(5) = O(1).**

Void insereordenado(TipoLista *l , Ponteiro t1, Ponteiro t2); Insere uma nova frequência na posição certa da lista. No melhor caso(Frequência é a menor da lista) é O(1). No pior caso ela é O(n-2k), onde k é o numero de vezes que foi construida uma subárvore de Huffman, e a frequência é sempre inserida no final da lista de n-2k elementos.

void freelista(TipoLista *l); Remove o vetor auxiliar, a cabeça e a árvore de Huffman que são células da lista. Usa 3 vezes a função free(). Portanto considerando free() como O(1) a função é **O(3) = O(1).**

int pesquisacaractere(TipoItem *t,int n, unsigned char c); Pesquisa se o caractere lido do arquivo já foi inserido no vetor auxiliar. No melhor caso a função é **O(1)**, enquanto no pior caso ela é **O(n)**, já que todos os caracteres distintos do arquivo já foram lidos e inseridos.

void primeiraleitura(FILE *fp , TipoLista *l); Le os **t** caracteres do arquivo e para cada interação, chama a função **pesquisacaractere** como uma comparação. Em média a função será O(t.n), pois a medida que o arquivo é lido a função **pesquisacaractere** tende a **O(n).**

void ordenaauxiliar(TipoLista *l); Ordena o vetor auxiliar de acordo com as frequências pelo método ShellSort. Logo a função é **O(nlog(n)).**

void huffman(TipoLista *l); Para os n-1 elementos da lista, chama os procedimentos **removemenor(O(1)) e insereordenado.** Como insereordenado possui melhor e pior caso, a função **huffman** será:
Melhor Caso: O(n-1).O(1) = O(n-1) = **O(n)**
Pior Caso: O(n-1).O(n-2k) = O(n^2 -2kn -n + 2k) = (Em média) **O(n^2).**

void segundaleitura(FILE *fp , TipoLista *l); Inicialmente escreve a tabela de **n** caracteres e em seguida para os **m** caracteres do arquivo comprido chama os procedimentos:

void escrevetabela(FILE *fw , TipoLista *l); Imprime os **n** caracteres do vetor auxiliar. Temos portanto uma função **O(n).**

void tratabuffer(char ** buffer, FILE *fw); Para um buffer de tamanho **b** , realiza **b** comparações em uma interação. Em seguida realiza apenas desalocações e atribuições. Portanto a função é O(b). Mas 8<b<30, então a função é **O(1).**

void freeall(TipoLista *l); Chama os procedimentos **freearvore(O(n)) e freelista(O(1)).** Temos então para essa função **O(n).**

void imprimebytes(char *imprime, FILE *fw); Para um buffer de tamanho máximo igual a **8** , converte os **b** bits do buffer para decimal, e imprime o número decimal como caractere. Logo a função é no pior caso O(8) = **O(1).**

void imprimeresto(char *resto , FILE *fw); Chama a função **imprimebytes(O(1))** para imprimir o resto de bits do buffer. Então no pior caso , imprimeresto é **O(8) = O(1)** que vale também para o melhor caso.

void letabela(FILE *fp , TipoLista *l); Lê os **n** elementos escritos anteriormente pela função **escrevetabela**. Portanto temos uma função **O(n).**

void descomprime(FILE *fp , Ponteiro a); Para os **u** caracteres do arquivo compactado, chama os procedimentos: **learvore(O(1)) e convertedecimal(O(1)).** Portanto a função é **O(u).**

void convertedecimal(int c, char **caminho, int resto); Realiza uma série de interações e atribuições para a conversão de um inteiro decimal para binário. No caso, o número binário tem no máximo 8 bits, então nossa função será **O(8) = O(1)** para o melhor e pior caso.

4.Testes:

Testes foram realizados com o programa de forma a verificar o seu funcionamento. Os testes foram realizados em um AMD X2, com 4 Gb de memória. A figura abaixo mostra a saída escrita em um arquivo de uma execução para a seguinte entrada especificada:

```
Boletus edulis, commonly known as bun bun, porcino or cep, is a basidiomycete fungus, and the type species of the genus Boletus. Widely distributed in the
Northern Hemisphere across Europe, Asia, and North America, it does not occur naturally in the Southern Hemisphere, although it has been introduced to
southern Africa, Australia and New Zealand. Several closely related European mushrooms formerly thought to be varieties or forms of B. edulis have been
shown using molecular phylogenetic analysis to be distinct species, and others previously classed as separate species are conspecific with this species.
The western North American species commonly known as the California king bolete (Boletus edulis var. grandedulis) is a large, darker-coloured variant that
was first formally identified in 2007.

The fungus grows in deciduous and coniferous forests and tree plantations, forming symbiotic ectomycorrhizal associations with living trees by enveloping
the tree's underground roots with sheaths of fungal tissue. The fungus produces spore-bearing fruit bodies above ground in summer and autumn. The fruit
body has a large brown cap which on occasion can reach 35 cm (14 in) in diameter and 3 kg (6.6 lb) in weight. Like other boletes, it has tubes extending
downward from the underside of the cap, rather than gills; spores escape at maturity through the tube openings, or pores. The pore surface of the B. edulis
fruit body is whitish when young, but ages to a greenish-yellow. The stout stipe, or stem, is white or yellowish in colour, up to 25 cm (10 in) tall and 10
cm (3.9 in) thick, and partially covered with a raised network pattern, or reticulations.

Prized as an ingredient in various foods, B. edulis is an edible mushroom held in high regard in many cuisines, and is commonly prepared and eaten in
soups, pasta, or risotto. The mushroom is low in fat and digestible carbohydrates, and high in protein, vitamins, minerals and dietary fibre. Although it
is sold commercially, it has not been successfully grown in cultivation. Available fresh in autumn in Central, Southern and Northern Europe, it is most
often dried, packaged and distributed worldwide. Keeping its flavour after drying, it is then reconstituted and used in cooking. Boletus edulis is one of
the few fungi that are sold pickled. The fungus also produces a variety of organic compounds with a diverse spectrum of biological activity, including the
steroid derivative ergosterol, a sugar binding protein, antiviral compounds, antioxidants, and phytochelatins, which give the organism resistance to toxic
heavy metals.
```

```
./comprime entrada.txt comprime.txt
```

```
./descomprime comprime.txt saida.txt
```

Obviamente o arquivo saida.txt é igual ao arquivo de entrada entrada.txt.

5.Conclusão:

A implementação do trabalho transcorreu sem maiores problemas e os resultados ficaram dentro do esperado. Entretanto, a maior dificuldade encontrada foi a escrita de um byte em um arquivo e a sua respectiva descompressão.

Referências

[1] Ziviani, N., Projeto de Algoritmos com Implementações em Pascal e C, 2a Edição, Editora Thomson, 2004.
Curso de linguagem C – UFMG – disponível em
http://mico.ead.cpdee.ufmg.br/cursos/C/index.html
http://pt.wikipedia.org/wiki/Codifica%C3%A7%C3%A3o_de_Huffman

Anexos:

- lista.h
- lista.c
- arvore.h
- arvore.c
- procedimentos.h
- procedimentos.c
- comprime.c
- descomprime.c