UNIVERSIDADE DE SÃO PAULO
INSTITUTO DE CIÊNCIAS MATEMÁTICAS E DE COMPUTAÇÃO
http://www.icmc.usp.br/

Professor: Rodrigo Fernandes de Mello (mello at icmc.usp.br)

Trabalho 04: Compactação de Huffman

1 Prazos, Especificações e Observações importantes:

- O trabalho descrito a seguir é individual e não será tolerado qualquer tipo de plágio ou cópia em partes ou totalidade do código. Caso seja detectada alguma irregularidade, os envolvidos serão chamados para conversar com o professor responsável pela disciplina e os trabalhos serão zerados.

- A interpretação desta descrição faz parte do trabalho. Leia a descrição do trabalho com atenção e várias vezes, anotando os pontos principais e as possíveis formas de resolver o problema. Comece a trabalhar o quanto antes para não ficar dúvidas e você consiga entregar o trabalho a tempo.

- O trabalho deverá ser submetido em formato zip/Makefile contendo o arquivo principal, o Makefile e possíveis TAD's implementadas (pares `.h .c`). Atente para o fato que todos os arquivos enviados deveram conter um cabeçalho contendo nome e número USP.

- A implementação é livre, crie quantas TAD's julgarem necessárias. Importante notar que a modularização do código bem como a forma como as TAD's foram criadas serão levados em consideração na atribuição final da nota.

- Compile o programa em um sistema operacional Linux antes de submeter. **Trabalho em que o comando `make` e `make run` não funcione, não será corrigido e consequentemente receberá nota zero**. Caso não tenha o sistema operacional Linux instalado em sua maquina, aconselha-se a utilização de uma máquina virtual. Como instalar o sistema operacional Linux em uma maquina virtual está descrito no seguinte tutorial: `http://maistutoriais.com/2012/01/instalar-maquina-virtual-usando-o-virtual-box/`.

- Referencie, com um comentário no próprio código, qualquer algoritmo ou trecho de código retirado da internet. Código copiado sem a devida referencia é considerado plágio que por sua vez é crime.

- Nenhuma saída do trabalho será em arquivo, ou seja, o resultado dos algoritmos deverá ser exibidos utilizando o *stdout* (`printf`).

2 Restrições:

- O trabalho deverá obrigatoriamente utilizar o arquivo `trab4.c` sem que se tenha qualquer alteração até mesmo no nome do arquivo.

- Este trabalho deverá ser construído utilizando de um Heap Dinâmico. Caso sua implementação esteja correta e utilize Heap estático o seu trabalho receberá somente metade dos pontos (5 pontos). Ainda, caso seu trabalho não utilize qualquer forma de Heap, sua nota máxima é 3
- A implementação do Heap Dinâmico deverá ter a mesma complexidade computacional do Heap estático.

3 Huffman

A codificação de Huffman é um método de compressão que usa as probabilidades de ocorrência dos símbolos no conjunto de dados a ser comprimido para determinar códigos de tamanho variável para cada símbolo.

Para atribuir aos caracteres mais frequentes os códigos binários de menor comprimento, constrói-se uma arvore tomando como base as probabilidades de ocorrência de cada símbolo. Nesta árvore as folhas representam os símbolos presentes nos dados, associados com suas respectivas probabilidades de ocorrência. Os nós intermediários representam a soma das probabilidades de ocorrência de todos os símbolos presentes em suas ramificações e a raiz representa a soma da probabilidade de todos os símbolos no conjunto de dados. O processo se inicia pela junção dos dois símbolos de menor probabilidade, que são então unidos em um nó ao qual é atribuída a soma de suas probabilidades. Este novo nó é então tratado como se fosse uma folha da árvore, isto é, um dos símbolos do alfabeto, e comparado com os demais de acordo com sua probabilidade. O processo se repete até que todos os símbolos estejam unidos sob o nó raiz.

3.1 Proposta

3.1.1 Árvore de frequência

Para determinar o mapa de bit por letra é importante calcular a frequência com que os caracteres aparecem no texto. Em particular, em nosso trabalho, iremos trabalhar com os caracteres de a-z, A-Z, 0-9 e o caractere espaço ‘ ’.

Ao determinar a frequência dos termos, monta-se a árvore de Huffman em que nós a esquerda recebem bit zero e nós a direita bit 1. Assim, neste trabalho, deverá ser construído um Heap Dinâmico baseando-se nas probabilidades de ocorrência de cada símbolo em que as operações tenham a mesma complexidade computacional da implementação de um Heap Estático. Importante ressaltar que esta não é a única forma de implementar o algoritmo de Huffman. Sendo assim, caso sua implementação esteja correta e utilize Heap estático o seu trabalho receberá somente metade dos pontos (5 pontos). Ainda, caso seu trabalho não utilize qualquer forma de Heap, sua nota máxima é 3.

Uma vez que determinou-se quais os dois caracteres vão ser unidos em uma nova árvore, o novo nó terá como frequência a somas dos nós que lhe deram origem e o seu termo é a concatenação dos termos dos nós filhos.

3.1.2 Compactação de Huffman

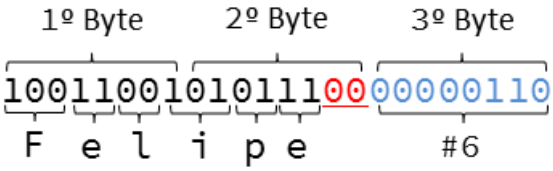
O sistema aqui implementado deverá ser capaz de compactar um texto plano, de no máximo 10000 (Dez mil) caracteres válidos, utilizando o algoritmo de Huffman. Uma vez que a árvore de frequência foi gerada (Heap Dinâmico), é necessário criar uma tabela que converta cada caractere do texto em um código binário único. Uma importante característica do processo de construção da tabela é que os caracteres com maior frequência

recebem o menor número de bits o que garante no final um texto representado com o menor número de bits possível no algoritmo de Huffman.

Essa tabela de símbolo/bits tem também a função de ajudar na descompactação da informação recebida o que torna necessário, ao compactar o texto original, que antes tenhamos a tabela de bits de cada termo. Uma possível abstração para a tabela de símbolo/bits para o texto "Felipe" seria:

```
F - 100
e - 11
i - 101
l - 00
p - 01
```

A compactação é feita substituindo-se os caracteres pelos bits correspondentes de acordo com a tabela já calculada. A cada 8 bits concatenados, gera-se 1 byte do texto compactado.



Um problema comum neste tipo de compactação é quando o número total de bits não é múltiplo de 8, ou seja, como somente é possível salva informações com a unidade de bytes, o ultimo byte do texto pode não ser totalmente utilizado. Com o intuito de resolver tal problema, existem varias formas de informar o fim do arquivo ou o número total de bits utilizado. Em nosso trabalho, utilizaremos o ultimo byte para indicar quantos bits foram usados no ultimo byte do texto compactado.

Na imagem anterior, o texto 'Felipe' utilizou somente 6 bits do último byte, desta forma os últimos dois bits foram completados com zero o ultimo byte traz o número 6 que indica quantos bits foram utilizados no byte final do texto.

Segue alguns links que podem ajudar na elaboração do algoritmo de Huffman e na transformação em byte do texto:

- `http://www.programminglogic.com/implementing-huffman-coding-in-c/`
- `http://michael.dipperstein.com/huffman/`

3.1.3 Descompactação de Huffman

A descompactação deverá fazer o serviço contrario da compactação. Este comando deverá ser capaz de processar um documento compactado e com a ajuda da tabela retornar a mensagem original. O resultado da descompactação do texto '???' é o texto original 'Felipe'.

4 trab4.c

```c

#include<stdio.h>
#include<stdlib.h>
#include<string.h>
```

```
5
//Include de TAD's particulares
7

9 int main(int argc, char *argv[]){

11   int i;
     char text[10001],
13        textCompac[10001],
          textDescompac[10001];
15   TABELA_HUFFMAN *tabela = criarTabelaHuffman();//cria a tabela de huffman

17   //le o texto que sera compactado
     for(i = 0; (text[i] = getchar()) != '\n' && i < 10001; ++i);
19   text[i] = '\0';

21   //compacta o texto e modifica a tabela
     textCompac = compactarHuffman(text, &tabela);
23   //descompacta o texto baseado na tabela criada
     textDescompac = descompactarHuffman(textCompac, tabela);
25
     //imprime o tamanho do texto original
27   printf("%d\n",(int)strlen(text));
     //imprime o tamanho do texto compactado
29   printf("%d\n",(int)strlen(textCompac));
     //imprime o texto descompactado
31   printf("%s\n",textDescompac);

33   //libera a memoria alocada para a tabela de Huffman
     liberarTabelaHuffman(&tabela);
35
     return 0;
37 }
```

Listing 1: trab4.c source code