INF1018 - Software Básico
Primeiro Trabalho

Compactação usando códigos livres de prefixo

O objetivo desse trabalho é implementar em Linguagem C uma função que permita compactar um texto e descompactá-lo usando um mapa de codificação, como aquele fornecido pelo algoritmo de Huffman.

Instruções Gerais

Leia com atenção o enunciado do trabalho e as instruções para a entrega. Em caso de dúvidas, não invente. Pergunte!

- O trabalho deve ser entregue no prazo informado no EaD da disciplina.
- Trabalhos entregues após o prazo perderão um ponto por dia de atraso.
- Trabalhos que não compilem (i.e., que não produzam um executável) não serão considerados, ou seja, receberão grau zero.
- Os trabalhos podem ser feitos em grupos de no máximo dois alunos.
- Alguns grupos poderão ser chamados para apresentações orais / demonstrações dos trabalhos entregues.

Introdução

A compactação de Huffman é um método de compressão de dados sem perdas e livre de prefixo que utiliza códigos de comprimento variável para representar os diferentes símbolos de um texto. Esse método foi desenvolvido por David A. Huffman em 1952, enquanto ele era estudante de pós-graduação no MIT.

Para mais detalhes sobre como funciona a codificação de Huffman ver o apêndice deste enunciado. Para o presente trabalho iremos considerar que o algoritmo nos fornece uma tabela contendo cada um dos símbolos, a sua representação em bits e a quantidade de bits necessária para representar o símbolo. Segue um exemplo de tal tabela:

Carac	Cód	# bits
' '	111 (7)	3
'e'	000 (0)	3
'a'	010 (2)	3
'f'	1101 (13)	4
'n'	0010 (2)	4
't'	0110 (6)	4
'm'	0111 (7)	4
'i'	1000 (8)	4
'h'	1010 (10)	4
's'	1011 (11)	4
'o'	00110 (6)	5
'u'	00111 (7)	5
'x'	10010 (18)	5
'p'	10011 (19)	5
'r'	11000 (24)	5
'l'	11001 (25)	5

Sendo que a terceira coluna, contendo a quantidade de bits usada por um símbolo, existe para facilitar a tarefa de decodificação.

Utilizando a tabela, o texto a seguir

```
this is an example of a huffman tree
```

ficaria compactado, em binário como mostrado a seguir.

Sequência bits separados por letras:

```
0110 1010 1000 1011 111 1000 1011 111 010 0010 111 000 10010 010 0111 10011 11001 000 111 00110 1101 111 010 111 1010 00111 1101 1101 0111 010 0010
```

Ou seja, o fluxo de bits seria esse:

```
01101010100010111111000101111101000101110001001001001111001111001000111001101101111010111101000111101110101110100010111011011000000000
```

Agora, separando em bytes, temos:

```
01101010 10001011 11110001 01111101 00010111 0001001 00100111 10011110 01000111 00110110 11110101 11101000 11111011 10101110 10001011 10110110 00000
```

Implementação

Para implementar o seu trabalho, considere um arquivo que contenha apenas letras maiúsculas (sem sinais diacríticos), símbolo de espaço, vírgula, ponto, \n (<ENTER>) e EOT (End of Transmission).

Em um arquivo chamado **codifica.c**, crie as duas funções descritas mais abaixo. O header com os protótipos das funções estão aqui. Esse arquivo não pode ser modificado porque ele não fará parte da entrega do trabalho.

Use a seguinte estrutura da tabela de compactação/descompactação de um código livre de prefixo (como o de Huffman por exemplo) para esse trabalho.

```
struct compactadora {
    char simbolo;
    unsigned int codigo;
    int tamanho;
};
```

Faça uma função chamada **compacta** que recebe um ponteiro para um arquivo texto já aberto para leitura (a função chamadora deve abrir o arquivo), um ponteiro para um arquivo binário já aberto para escrita (aqui, novamente, a função chamadora deve abrir o arquivo), e um vetor de 32 posições do tipo da estrutura representada anteriormente.

```
void compacta(FILE *arqTexto, FILE *arqBin, struct compactadora *v);
```

A sua função deverá ler o arquivo texto, compactar seguindo a tabela fornecida e salvar a codificação do texto no arquivo binário. No final do arquivo binário, o seu programa deverá gravar o caractere de fim de arquivo (EOT). Considere que esse caractere está na posição 31 do vetor de compactação.

Faça uma função chamada descompacta que recebe um ponteiro para um arquivo binário já aberto para leitura (a função chamadora deve abrir o arquivo), um ponteiro para um arquivo texto já aberto para escrita (aqui, novamente, a função chamadora deve abrir o arquivo), e um vetor de 32 posições do tipo da estrutura representada anteriormente. Considere que o vetor está preenchido com valores semelhantes à tabela apresentada anteriormente.

```
void descompacta(FILE *arqBin, FILE *arqTexto, struct compactadora *v);
```

A função deverá ler o arquivo binário compactado seguindo um algoritmo livre de prefixo (como o de Huffman por exemplo) e salvar os dados descompactados no arquivo texto.

Dicas

- Implemente seu trabalho por partes, gerando o teste para cada parte implementada antes de prosseguir. Por exemplo, você pode implementar primeiro a operação de compactação, e para testá-la usar alguma função auxiliar que realize um dump do arquivo binário usado para armazenar o texto compactado.
- Repare que uma função pode ser usada para testar a outra função (por exemplo, você pode compactar um arquivo e depois descompactá-lo e verificar se o resultado é igual ao original!).
- Não se esqueça de ter planejado um roteiro robusto de testes antes de entregar seu trabalho! Para cada operação implementada, pense nos diferentes casos relevantes.
- Veja aqui um exemplo de tabela e de estrutura compactadora. Mas é apenas um exemplo!!!

Entrega

Devem ser entregues via Moodle pelo dois arquivos.

1. O arquivo **codifica.c**. Coloque no início do arquivo fonte, como comentário, os nomes dos integrantes do grupo, da seguinte forma:

   ```
   /* Nome_do_Aluno1 Matricula Turma */
   /* Nome_do_Aluno2 Matricula Turma */
   ```

   Lembre-se que este arquivo não deve conter a função main!
2. Um arquivo texto, chamado **relatorio.txt**, relatando o que está funcionando e, eventualmente, o que não está funcionando. Explique sua estratégia geral de teste, como organizou seu roteiro e consequentemente seu arquivo de teste.

   Você não deve explicar a sua implementação neste relatório. Seu programa deve ser suficientemente claro e bem comentado.

   Coloque também no relatório o nome dos integrantes do grupo.

Coloque na área de texto da tarefa do Moodle os nomes e turmas dos integrantes do grupo.

Para grupos de alunos da mesma turma, apenas uma entrega é necessária (usando o login de um dos integrantes do grupo).

Se o grupo for composto por alunos de turmas diferentes, os dois alunos deverão realizar a entrega, e avisar aos professores!

Apêndice A

O algoritmo de compactação de Huffman funciona da seguinte maneira:

1. Frequência de Ocorrência: Primeiro, a frequência de ocorrência de cada símbolo no conjunto de dados é determinada.
2. Construção da árvore de Huffman: Com base nas frequências dos símbolos, uma árvore de Huffman é construída. Essa árvore é uma árvore binária em que os símbolos menos frequentes estão nas folhas mais distantes da raiz e os símbolos mais frequentes estão mais próximos da raiz.
3. Atribuição de Códigos: Os códigos de Huffman são atribuídos aos símbolos com base na sua posição na árvore. Os símbolos mais frequentes recebem códigos mais curtos, enquanto os menos frequentes recebem códigos mais longos.
4. Compactação: O conjunto de dados original é percorrido e cada símbolo é substituído pelo seu código de Huffman correspondente. Isso resulta em uma sequência de bits mais curta, que representa o conjunto de dados compactado.
5. Descompactação: Para descompactar os dados, a árvore de Huffman original é reconstruída com base nos códigos de Huffman e, em seguida, a sequência de bits compactada é percorrida para recuperar os símbolos originais.

A compactação de Huffman é eficiente para conjuntos de dados em que alguns símbolos ocorrem com mais frequência do que outros sendo amplamente utilizada em algoritmos de compressão de arquivos.

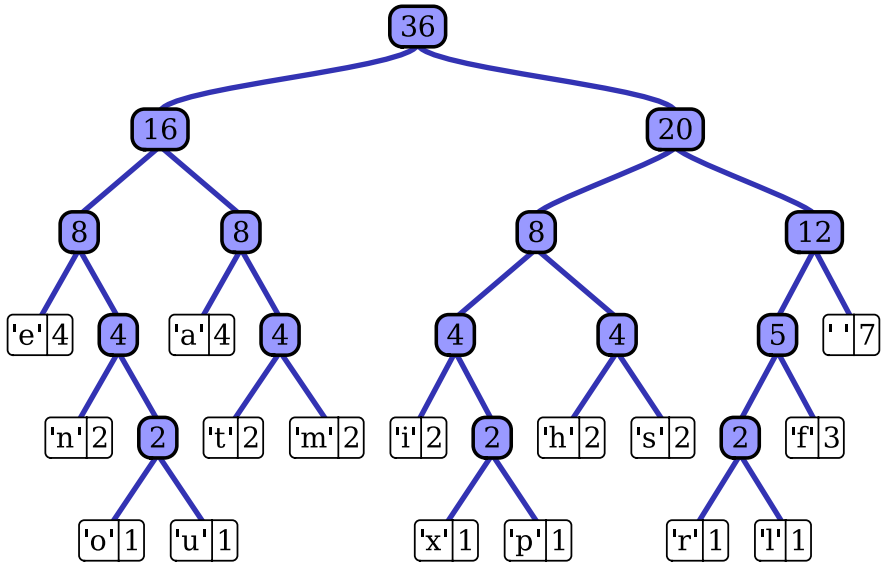
Exemplo (extraído da Wikipedia)

Considere o seguinte texto:

```
this is an example of a huffman tree
```

Aplicando a compactação de Huffman a esse texto, temos a seguinte árvore. Os nós internos armazenam a quantidade de ocorrência dos símbolos abaixo dele. Cada folha apresenta o símbolo e o seu número de ocorrências no texto. Cada ramo para a direita inclui um bit em 1 ao código, enquanto que cada ramo para a esquerda, um bit em 0. Por exemplo, considerando o símbolo 'f', a partir da raiz, desviamos para a direita (1) da raiz até o nó 20, depois novamente para a direita

(11), do nó 20 para o nó 12, agora para a esquerda (110) do nó 12 para o nó 5 e, finalmente, para a direita (1101) do nó 5 para a folha com o símbolo 'f', formando o código 1101.



Apêndice B

Valem algumas dicas para o trabalho 1 de Software Básico.

Sobre manipulação de arquivos:

Para abrir um arquivo para leitura usem a função:

```
fp = fopen("","r");
```

Se forem abrir para escrita

```
fp = fopen("","w");
```

e lembrem de adicionar ao modo de abertura a letra **b** caso estejam abrindo um arquivo binário.

Para ler um arquivo neste trabalho vocês podem usar as seguintes funções:

```
fread
fgetc
```

E para escrever num arquivo podem usar estas funções:

```
fwrite
fputc
```

observação: estas são sugestões de funções e você pode usar outras se preferir.

Sobre testes

É importante terem arquivos para testar o trabalho de vocês. Com base na tabela compactadora passada de exemplo no enunciado, vocês podem criar arquivos ASCCI e os correspondentes binários. Por exemplo, para um arquivo texto com o seguinte conteúdo:

```
BOM\n
DIA\n
```

Teremos a seguinte sequencia de bits de acordo com a tabela exemplo fornecida neste enunciado:

```
000001 10000 11001 00000001 00001 1101 0101 00000001 000000000000
```

O que corresponde à seguinte sequencia de byte em hexadecimal:

```
06 19 01 0E A8 08 00 00
```

Desta forma para este exemplo acima podemos usar estes dois arquivos para testar nosso trabalho:

- teste1.txt
- teste1.bin