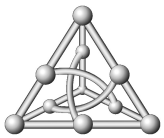


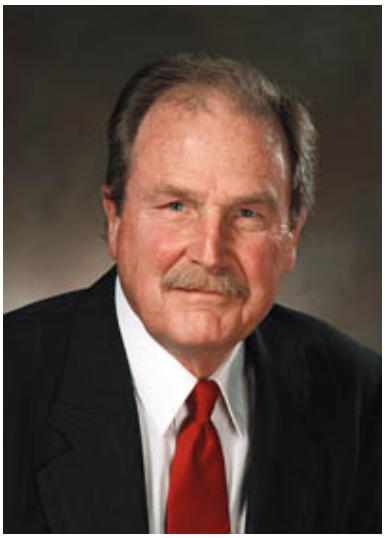
UNIVERSIDADE FEDERAL DE MATO GROSSO DO SUL
Faculdade de Computação

# Compressão de dados Huffman (contagem de "sobra")

# Trabalho 2

**Estruturas de Dados**

## 1 Descrição



David Albert Huffman

David Albert Huffman (⋆09/08/1925 – †07/10/1999) foi um pioneiro na ciência da computação. Ingressou no MIT[1] em 1953, indo em 1967 para a Universidade da Califórnia, Santa Cruz, como membro do corpo docente fundador do Departamento de Ciência da Computação. Aposentou-se em 1994, mas permaneceu em atividade como professor emérito, no ensino da teoria da informação e cursos de análise de sinais.

Huffman fez contribuições importantes em muitas outras áreas, incluindo a teoria da informação e codificação, projetos de sinal para radar, aplicações de comunicações e procedimentos para o projeto de circuitos lógicos assíncronos. Como conseqüência de seu trabalho sobre as propriedades matemáticas das superfícies com "curvatura zero de Gauss", Huffman desenvolveu suas próprias técnicas para dobrar papel em formas esculpidas incomuns (que deu origem ao campo do origami computacional).

É mais conhecido pelo desenvolvimento do Código de Huffman, técnica muito importante de compressão sem perda de dados com uma codificação de comprimento variável, resultado de um artigo que escreveu enquanto era estudante de doutorado no MIT.

Sua tarefa é escrever um programa em C ou C++ que, dado um arquivo de texto qualquer, seja capaz de compactá-lo e posteriormente descompactar o arquivo gerado, utilizando o algoritmo de Huffman conforme visto em sala.

Ao compactar, siga os seguintes passos:

1. Calcule a frequência dos caracteres
2. Crie a lista de min-prioridades (heap) e utilize-a para montar a árvore de Huffman
3. Mapeie os códigos dos caracteres em uma tabela de hash de mapeamento (hashmap) **ou** em um vetor
4. Gere o arquivo compactado com o cabeçalho e os dados

[1] Massachusetts Institute of Technology

Após escrever o último bit do texto compactado, você deve verificar quantos bits do último byte não foram utilizados (S = "sobra"). Escreva então um byte adicional no fim do arquivo para representar o valor S. Ao descompactar:

1. Leia o cabeçalho em binário e o valor de S (último byte)
2. Recrie a árvore de huffman a partir do cabeçalho
3. Descompacte o arquivo

Utilize o seguinte formato de cabeçalho:

| K | Letras do alfabeto | | | Bits do percurso pré-ordem | | | Texto compactado | | | | S |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 2 bytes | byte | ... | byte | byte | ... | byte | byte | byte | byte | ... | byte | byte |

O valor K (2 bytes de um `unsigned int`, pois pode chegar a 256) é número de caracteres do alfabeto. Observe que os bits do texto compactado terminam no penúltimo byte.

## 2 Entrada e saída

A operação a ser realizada (c ou d) é dada pela linha de comando como o primeiro parâmetro, o arquivo de entrada como o segundo e o arquivo de saída como o terceiro, por exemplo:

```
./programa c texto.txt texto.huff
```

```
./programa d texto.huff texto-descompactado.txt
```

Embora seja comum utilizarmos as extensões `.txt` e `.huff`, os arquivos dados podem ter qualquer extensão. Considere que os arquivos de texto a serem compactados poderão conter espaços, quebras de linha ou sinais de pontuação.

## 3 Exemplo de entrada

Um arquivo com o seguinte conteúdo:

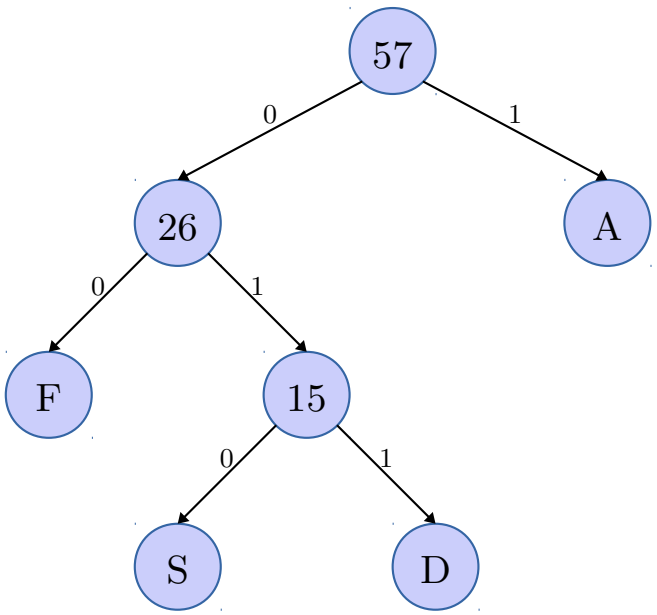
```
AFAFFADADSFDSAFDSAFADAFDASFDSAFADAFDSFAAAAAAAAAAAAAAAAAAA
```

Ao contrário do padrão seguido no Linux, vamos considerar que este arquivo não possui um `\n` no final do texto.

## 4 Exemplo de saída

Antes de analisar a saída gerada e para a melhor compreensão desta, vejamos a tabela de frequências e a árvore de Huffman, passos intermediários do processo de compactação.

| Símbolo | Valor | Frequência |
|---|---|---|
| A | 65 | 31 |
| D | 68 | 9 |
| F | 70 | 11 |
| S | 83 | 6 |




Finalmente, a saída (sequência de bits do arquivo compactado) gerada para o exemplo é[2]:

```
0000000: 00000100 00000000 01000110 01010011 01000100 ..FSD
0000005: 01000001 00101111 00100001 01110110 10000110 A/!v.
000000a: 10100011 01010010 11100011 10100001 10101001 .R...
000000f: 01110001 10100011 11111111 11111111 10000000 q....
0000014: 00000111                                      .
```

Na imagem acima, são exibidos 5 bytes por linha. A coluna mais à esquerda é o deslocamento no arquivo e a mais à direita é a representação em `char` de cada byte (faz sentido apenas para os caracteres do alfabeto). Bits em azul representam o tamanho K = 4 (2 bytes[3]) do alfabeto, em marrom os caracteres do alfabeto {F,D,S,A} e em vermelho os bits do percurso pré-ordem. Os bits restantes até o penúltimo byte representam o texto compactado (com bits de sobra em em lilás), e o último byte em verde representa o valor S = 7 (a quantidade de bits de sobra no penúltimo byte).

[2] Exibição da saída gerada pelo comando `xxd -c 5 -b arquivo_compactado.huff`
[3] Em little-endian

## 5 Exigências

A seguir você encontrará uma lista de exigências para o desenvolvimento deste trabalho. Caso alguma delas não seja cumprida, a nota do trabalho será ZERO.

Você DEVE utilizar os procedimentos de compactação e descompactação conforme visto em sala, além do formato de cabeçalho descrito na primeira seção.

NÃO É PERMITIDO utilizar qualquer estrutura de dados já implementada em bibliotecas (heap, por exemplo). Contudo você pode utilizar as classes `map`, `pair`, `string`, `vector`, `list` ou `bitset` em C++. Você deve implementar as estruturas de dados restantes. O uso de uma tabela de hash (com `map` ou construída por você) é opcional (vide critérios de correção).

Você DEVE, ao final do seu programa, liberar toda memória alocada dinamicamente (vetores, listas, e outras) e também garantir que seu programa não realiza acessos inválidos à memória. Para verificar essa questão, será utilizado o utilitário *Valgrind*, com o comando `valgrind --leak-check=full --show-reachable=yes --track-fds=yes --track-origins=yes ./programa <parâmetros>`. Embora execute mais lentamente, você pode **compilar** seu programa com a opção de depuração `-g`, permitindo que o valgrind detalhe mais a saída, incluindo os números das linhas de seu programa com eventuais problemas.

Fique atento às especificações de estruturas de dados utilizadas e da saída.

# 6 Entrega

Instruções para entrega do seu trabalho:

1. **Cabeçalho**

   Seu trabalho deve ter um cabeçalho com o seguinte formato:

```
/*************************************************
 *
 * Nome do(a) estudante
 * Trabalho 2
 * Professor(a): Nome do(a) professor(a)
 *
 */
#include <stdio.h>
```

2. **Compilador**

   Para a correção do trabalho, será utilizado o compilador da linguagem C++ da coleção de compiladores GNU `g++`, com as opções de compilação `-Wall -pedantic -std=c++11` para C++ ao corrigir os programas. Opcionalmente, você pode utilizar a flag `-g` para compilar seu programa e testá-lo pelo `valgrind`. Antes de entregar seu programa, verifique se ele tem extensão `.cpp`, compila sem mensagens de alerta e executa corretamente e não possui problemas de memória acusados pelo `valgrind`.

3. **Forma de entrega**

   A entrega será realizada diretamente na página da disciplina no AVA/UFMS. Um fórum de discussão deste trabalho já se encontra aberto. Após abrir uma sessão digitando seu *login* e sua senha, vá até o tópico "Trabalhos", e escolha "T2 - Entrega". Você pode entregar o trabalho quantas vezes quiser até às **06 horas e 00 minutos** do dia **06 de julho de 2020**. A última versão entregue é aquela que será corrigida. Encerrado o prazo, não serão mais aceitos trabalhos.

4. **Atrasos**

   Trabalhos atrasados não serão aceitos. Não deixe para entregar seu trabalho na última hora. Para prevenir imprevistos como queda de energia, problemas com o sistema, falha de conexão com a internet, sugerimos que a entrega do trabalho seja feita pelo menos um dia antes do prazo determinado.

5. **Erros**

   Trabalhos com erros de compilação receberão nota ZERO. Faça todos os testes necessários para garantir que seu programa está livre de erros de compilação.

6. **O que entregar?**

   Você deve entregar um único arquivo contendo APENAS o seu programa fonte com o mesmo nome de seu login no moodle, como por exemplo, `fulano_silva.c`. NÃO entregue qualquer outro arquivo, tal como o programa executável, já compilado.

7. **Verificação dos dados de entrada**

   Não se preocupe com a verificação dos dados de entrada do seu programa. Seu programa não precisa fazer consistência dos dados de entrada. Isto significa que se, por exemplo, o seu programa pede um número entre 1 e 10 e o usuário digita um número negativo, uma letra, um cifrão, etc, o seu programa pode fazer qualquer coisa, como travar o computador ou encerrar a sua execução abruptamente com respostas erradas.

8. **Arquivo com o programa fonte**

   Seu arquivo contendo o programa fonte na linguagem C++ deve estar bem organizado. Um programa na linguagem C++ tem de ser muito bem compreendido por uma pessoa. Verifique se seu programa tem a indentação adequada, se não tem linhas muito longas, se tem variáveis com nomes significativos, entre outros. Não esqueça que um programa bem descrito e bem organizado é a chave de seu sucesso. Não esqueça da documentação de seu programa e de suas funções.

   Dê o nome do seu usuário do moodle para seu programa e adicione a extensão `.cpp` a este arquivo. Por exemplo, `fulano_silva.cpp` é um nome válido.

9. **Conduta Ética**

   O trabalho deve ser feito INDIVIDUALMENTE. Cada estudante tem responsabilidade sobre cópias de seu trabalho, mesmo que parciais. Não faça o trabalho em grupo e não compartilhe seu programa ou trechos de seu programa. Você pode consultar seus colegas para esclarecer dúvidas e discutir idéias sobre o trabalho, ao vivo ou no fórum de discussão da disciplina, mas NÃO copie o programa!

   Trabalhos considerados plagiados terão nota ZERO.