Compressor Huffman

Generated by Doxygen 1.15.0

Chapter 1

File Index

1.1 File List

Here is a list of all files with brief descriptions:

Chapter 2

File Documentation

2.1 main.c File Reference

Implementação completa do algoritmo de compressão e extração Huffman.

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

Data Structures

- struct Node

  Nó da árvore de Huffman. *More...*
- struct List

  Lista encadeada ordenada usada para construir a árvore. *More...*

Macros

- #define FREQ_SIZE 256
- #define MAX_FILE_NAME_SIZE 1000
- #define TREE_BUFFER_MAX_SIZE 10000
- #define RESET "\033[0m"
- #define BOLD "\033[1m"
- #define CYAN "\033[36m"
- #define GREEN "\033[32m"
- #define YELLOW "\033[33m"
- #define RED "\033[31m"
- #define MAGENTA "\033[35m"

Typedefs

- typedef struct Node Node
- typedef struct List List

Functions

- void clear_screen ()

  Limpa o terminal.
- void get_freq (unsigned char *content, unsigned int freq[ ], size_t size)

  Conta a frequência de cada caractere em um conteúdo.
- void create_list (List *list)

  Inicializa uma lista vazia.
- void insert_sorted (List *list, Node *node)

  Insere um nó na lista em ordem crescente de frequência.
- void fill_list (unsigned int freq[ ], List *list)

  Preenche a lista com nós criados a partir das frequências.
- Node * remove_begin (List *list)

  Remove e retorna o primeiro nó da lista.
- Node * build_tree (List *list)

  Constrói a árvore de Huffman a partir da lista ordenada.
- void free_tree (Node *root)

  Libera recursivamente toda a árvore de Huffman.
- int tree_height (Node *root)

  Calcula a altura da árvore.
- char ** allocate_table (int columns)

  Aloca a tabela de códigos binários dos caracteres.
- void build_table (char **table, Node *root, char *path, int columns)

  Constrói a tabela de códigos binários recursivamente.
- int size_str (char **table, unsigned char *content, size_t size)

  Retorna o tamanho total da string codificada.
- char * encode (char **table, unsigned char *content, size_t size)

  Codifica o conteúdo original usando a tabela de Huffman.
- unsigned int is_bit_one (unsigned char byte, int i)

  Verifica se um determinado bit é 1 em um byte.
- void build_tree_buffer (Node *root, char *buffer, int *index)

  Gera a sequência binária da árvore de Huffman (pré-ordem).
- void write_bit_str (FILE *file, char *bit_str, int bit_qnt)

  Escreve uma sequência de bits como bytes em arquivo.
- void mount_header (FILE *file, unsigned char str[ ], Node *root)

  Monta e grava o cabeçalho do arquivo .huff.
- void compress (unsigned char str[ ], char file_name[ ], Node *root)

  Comprime o conteúdo e escreve o arquivo .huff.
- Node * build_tree_from_bits (unsigned char *buffer, int *index, int total_bits)

  Reconstrói a árvore a partir dos bits do cabeçalho.
- Node * build_tree_from_header (FILE *file)

  Lê o cabeçalho do arquivo .huff e reconstrói a árvore.
- char * file_with_extra_huff (char *file_name)

  Gera o nome do arquivo comprimido (.huff).
- char * remove_huff_extension (const char *file_name)

  Remove a extensão .huff do nome do arquivo.
- void extract (char huff_name[ ])

  Extrai um arquivo comprimido (.huff) para o original.
- int main ()

  Função principal do programa (menu interativo).

2.1.1 Detailed Description

Implementação completa do algoritmo de compressão e extração Huffman.

Este programa realiza compressão e descompressão de arquivos utilizando o algoritmo de Huffman, preservando a estrutura da árvore de codificação no cabeçalho do arquivo gerado (.huff).

O código é dividido em módulos lógicos:

- Utilitários gerais (funções auxiliares)
- Estruturas e manipulação de lista ordenada
- Construção e destruição da árvore de Huffman
- Geração e leitura do cabeçalho
- Compressão e extração de arquivos
- Interface simples via terminal

Author

Victor Oliveira, Lorenzo Holanda, Lucitanea Lopes

Version

1.0

Date

2025-10-23

2.1.2 Data Structure Documentation

2.1.2.1 struct Node

Nó da árvore de Huffman.

Cada nó pode representar um caractere (folha) ou um nó interno com dois filhos (esquerdo e direito).

Data Fields

void *	c	Ponteiro genérico para o caractere armazenado.
void *	freq	Frequência do caractere.
void *	left	Filho esquerdo (subárvore esquerda).
void *	next	Próximo nó da lista encadeada.
void *	right	Filho direito (subárvore direita).

2.1.2.2 struct List

Lista encadeada ordenada usada para construir a árvore.

Data Fields

void *	head	Cabeça da lista.
void *	size	Quantidade de nós armazenados.

2.1.3 Macro Definition Documentation

2.1.3.1 BOLD

```
#define BOLD "\033[1m"
```

2.1.3.2 CYAN

```
#define CYAN "\033[36m"
```

2.1.3.3 FREQ_SIZE

```
#define FREQ_SIZE 256
```

2.1.3.4 GREEN

```
#define GREEN "\033[32m"
```

2.1.3.5 MAGENTA

```
#define MAGENTA "\033[35m"
```

2.1.3.6 MAX_FILE_NAME_SIZE

```
#define MAX_FILE_NAME_SIZE 1000
```

2.1.3.7 RED

```
#define RED "\033[31m"
```

2.1.3.8 RESET

```
#define RESET "\033[0m"
```

2.1.3.9 TREE_BUFFER_MAX_SIZE

```
#define TREE_BUFFER_MAX_SIZE 10000
```

2.1.3.10 YELLOW

```
#define YELLOW "\033[33m"
```

2.1.4 Typedef Documentation

2.1.4.1 List

```
typedef struct List List
```

2.1.4.2 Node

```
typedef struct Node Node
```

2.1.5 Function Documentation

2.1.5.1 allocate_table()

```
char ** allocate_table (
            int columns)
```

Aloca a tabela de códigos binários dos caracteres.

2.1.5.2 build_table()

```
void build_table (
            char ** table,
            Node * root,
            char * path,
            int columns)
```

Constrói a tabela de códigos binários recursivamente.

2.1.5.3 build_tree()

```
Node * build_tree (
            List * list)
```

Constrói a árvore de Huffman a partir da lista ordenada.

2.1.5.4 build_tree_buffer()

```
void build_tree_buffer (
            Node * root,
            char * buffer,
            int * index)
```

Gera a sequência binária da árvore de Huffman (pré-ordem).

2.1.5.5 build_tree_from_bits()

```
Node * build_tree_from_bits (
            unsigned char * buffer,
            int * index,
            int total_bits)
```

Reconstrói a árvore a partir dos bits do cabeçalho.

2.1.5.6 build_tree_from_header()

```
Node * build_tree_from_header (
            FILE * file)
```

Lê o cabeçalho do arquivo .huff e reconstrói a árvore.

2.1.5.7 clear_screen()

```
void clear_screen ()
```

Limpa o terminal.

2.1.5.8 compress()

```
void compress (
            unsigned char str[],
            char file_name[],
            Node * root)
```

Comprime o conteúdo e escreve o arquivo .huff.

2.1.5.9 create_list()

```
void create_list (
            List * list)
```

Inicializa uma lista vazia.

2.1.5.10 encode()

```
char * encode (
            char ** table,
            unsigned char * content,
            size_t size)
```

Codifica o conteúdo original usando a tabela de Huffman.

2.1.5.11 extract()

```
void extract (
             char huff_name[])
```

Extrai um arquivo comprimido (.huff) para o original.

2.1.5.12 file_with_extra_huff()

```
char * file_with_extra_huff (
             char * file_name)
```

Gera o nome do arquivo comprimido (.huff).

2.1.5.13 fill_list()

```
void fill_list (
             unsigned int freq[],
             List * list)
```

Preenche a lista com nós criados a partir das frequências.

2.1.5.14 free_tree()

```
void free_tree (
             Node * root)
```

Libera recursivamente toda a árvore de Huffman.

2.1.5.15 get_freq()

```
void get_freq (
             unsigned char * content,
             unsigned int freq[],
             size_t size)
```

Conta a frequência de cada caractere em um conteúdo.

Parameters

content	Vetor de bytes do arquivo
freq	Vetor de frequências (tamanho 256)
size	Tamanho do conteúdo

2.1.5.16 insert_sorted()

```
void insert_sorted (
            List * list,
            Node * node)
```

Insere um nó na lista em ordem crescente de frequência.

2.1.5.17 is_bit_one()

```
unsigned int is_bit_one (
            unsigned char byte,
            int i)
```

Verifica se um determinado bit é 1 em um byte.

2.1.5.18 main()

```
int main ()
```

Função principal do programa (menu interativo).

2.1.5.19 mount_header()

```
void mount_header (
            FILE * file,
            unsigned char str[],
            Node * root)
```

Monta e grava o cabeçalho do arquivo .huff.

2.1.5.20 remove_begin()

```
Node * remove_begin (
            List * list)
```

Remove e retorna o primeiro nó da lista.

2.1.5.21 remove_huff_extension()

```
char * remove_huff_extension (
            const char * file_name)
```

Remove a extensão .huff do nome do arquivo.

2.1.5.22 size_str()

```
int size_str (
            char ** table,
            unsigned char * content,
            size_t size)
```

Retorna o tamanho total da string codificada.

2.1.5.23 tree_height()

```
int tree_height (
            Node * root)
```

Calcula a altura da árvore.

2.1.5.24 write_bit_str()

```
void write_bit_str (
            FILE * file,
            char * bit_str,
            int bit_qnt)
```

Escreve uma sequência de bits como bytes em arquivo.