

**UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO**

**CENTRO TECNOLÓGICO**

**DEPARTAMENTO DE INFORMÁTICA**

**Alan Herculano Diniz**

**Rafael Belmock Pedruzzi**

**Compactador de Arquivos: Trabalho para a disciplina de Estrutura de Dados I**

**Vitória**

**2018**

**Alan Herculano Diniz**

**Rafael Belmock Pedruzzi**

**Compactador de Arquivos: Trabalho para a disciplina de Estrutura de Dados I**

Trabalho para a disciplina de Estrutura de Dados I do curso de Ciência da Computação da Universidade Federal do Espírito Santo.

**Professor(a):** Patrícia Dockhorn Costa

**Disciplina:** Estrutura de Dados I

**Turma:** INF09292

**Vitória**

**2018**

Sumário

[1 – Introdução 4](#__RefHeading___Toc2064_1859988581)

[2 – Implementação 5](#__RefHeading___Toc2066_1859988581)

[2.1 – Árvore 5](#__RefHeading___Toc2068_1859988581)

# 1 – Introdução

Nos dias atuais, arquivos e dados possuem tamanho e complexidade muito grandes para serem enviados e recebidos sem nenhuma modificação. Portanto, é preciso de um método de compactar arquivos para que seu transporte seja viável e eficiente. A partir desse fato, têm-se a base deste segundo trabalho da disciplina de Estrutura de Dados I, em que o objetivo é programar um compactador na linguagem de programação C que possa criar uma versão compactada de um arquivo de entrada e que possa descompactar arquivos compactados.

Para criar esse compactador, será utilizado o algoritmo de Huffman, que é um método que usa o número de ocorrências dos bytes e suas probabilidades de ocorrerem para criar um arquivo com tamanho reduzido.

# 2 – Implementação

Para utilizar o algoritmo de Huffman, é necessário implementar certos tipos abstratos de dados que auxiliaram na construção do compactador. Nos próximos subitens, serão listados e explicados esses TAD’s.

## 2.1 – Árvore

O tipo árvore é um dos principais tipos utilizados na construção do compactador, já que ele é utilizado para codificar os bytes de tal forma que será possível construir o arquivo compactado. Ela possui os campos de ponteiros para as subárvores da direita e da esquerda, já que é uma árvore binária, outro para manter o dado e outro para manter a ocorrência desse dado.

Também são definidas funções que manipulam variáveis do tipo árvore, entre as principais, estão as seguintes:

Funções de criação e destruição na memória: essas funções são responsáveis por alocar dinamicamente e apagar da memória o espaço com o tamanho necessário para uma árvore. Na verdade, todo TAD possui essas funções, entretanto, a árvore possui um diferencial: ela possui duas funções de alocação dinâmica de espaço de memória, uma para criar um nó que possui subárvores e outra para criar uma nó folha (que não possui nenhuma subárvore). Essas funções possuem os seguintes formatos:

– Arvore\* Arvore\_CriaFolha(unsigned char caracter, int ocorrencias): recebe o dado e número de ocorrências desse dado no arquivo. A função retorna um ponteiro para um nó folha alocado dinamicamente e não possui condições ou efeitos colaterais.

– Arvore\* Arvore\_CriaArvore(int ocorrencias, Arvore\* esq, Arvore\* dir): recebe um número de ocorrências e ponteiros para as subárvores da direita e da esquerda e retorna um ponteiro para uma árvore. Também é necessário que as subárvores sejam existentes e estejam alocadas. Note que somente os nós folhas possuem referência para o dado do arquivo que será compactado.