Algoritmo de Huffman

INSTITUTO FEDERAL DE SANTA CATARINA

ENGENHARIA ELETRÔNICA

PROGRAMAÇÃO II

ALUNAS: JADE DUTRA LOPES E RAQUEL DUTRA KOTZIAS

O que é?

- A codificação de Huffman é um método de compressão;
- Utiliza a probabilidade de ocorrência de símbolos de um conjunto de dados para determinar códigos de tamanho variável para cada símbolo.

```
enquanto tamanho(alfabeto) > 1:
  S0 := retira_menor_probabilidade(alfabeto)
  S1 := retira_menor_probabilidade(alfabeto)
  X  := novo_nó
  X.filho0 := S0
  X.filho1 := S1
  X.probabilidade := S0.probabilidade + S1.probabilidade
  insere(alfabeto, X)
fim enquanto

X = retira_menor_símbolo(alfabeto) # nesse ponto só existe um símbolo.

para cada folha em folhas(X):
  código[folha] := percorre_da_raiz_até_a_folha(folha)
fim para
```

Exemplo do algoritmo

- Entrada do arquivo
- Incrementa a quantidade de cada símbolo
- Enfileira cada símbolo de acordo com a frequência

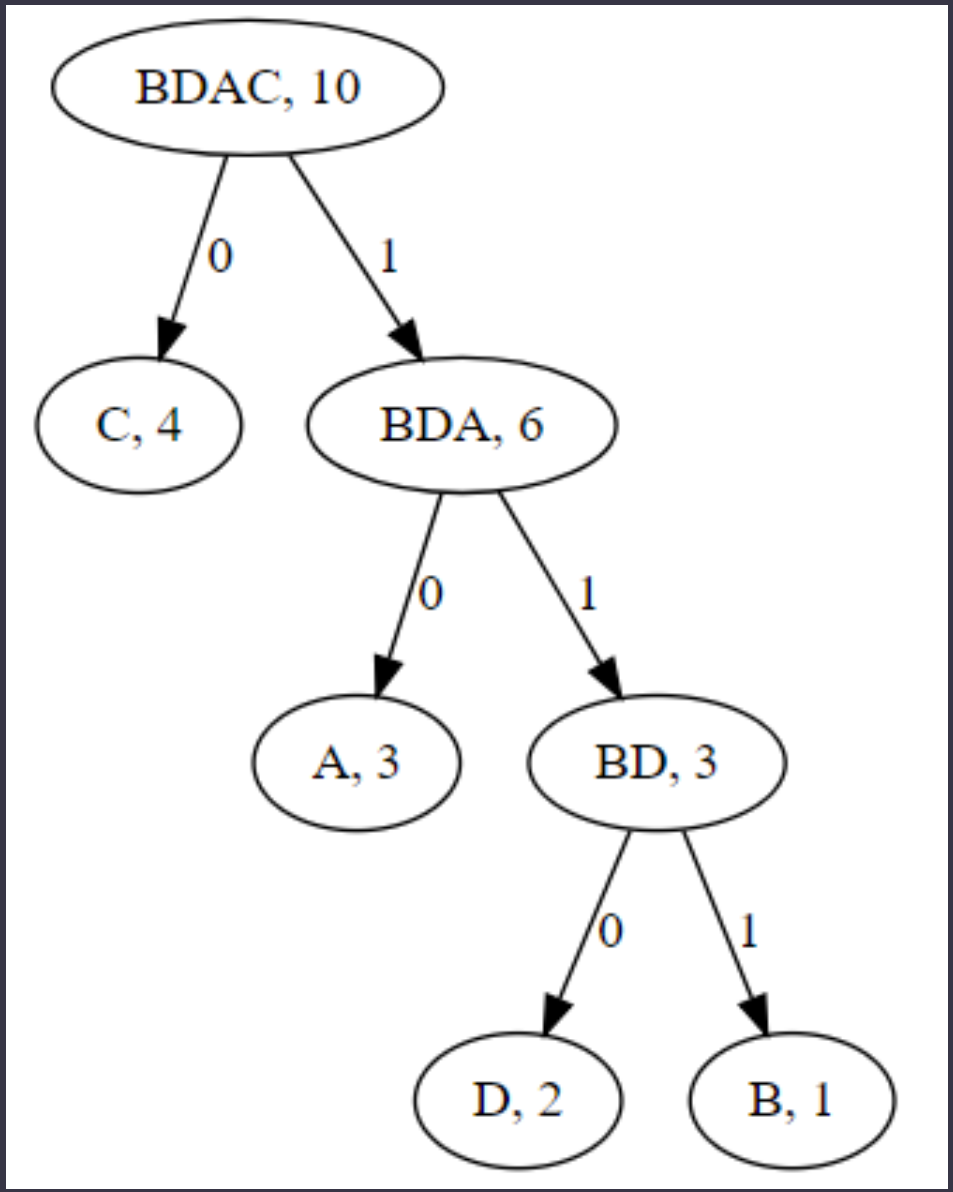
AAABCCCCDD

Quantidade de ocorrências:

- A – 3
- B – 1
- C – 4
- D – 2

Fila de prioridades:

(B, 1) (D, 2) (A, 3) (C, 4)



C	**0**
A	**10**
B	**110**
D	**111**

Arquivo de entrada: "Bolo bombom, bom!"

- Leitura binária
- Valores de acordo com a tabela ASCII

```
Valor: 108
Quantidade: 1
Valor: 66
Quantidade: 1
Valor: 44
Quantidade: 1
Valor: 33
Quantidade: 1
Valor: 32
Quantidade: 2
Valor: 98
Quantidade: 3
Valor: 109
Quantidade: 3
Valor: 111
Quantidade: 5
```

Construção do código

- **Huffman.c**
- *fila_t * read_file(char *filename);*
- **Tree.c**
- *tree_t* create_tree (int (c)(void, void*));*
- *void tree_add_node (tree_t* t, node_t* father, node_t* node);*
- *void tree_add_root(tree_t* tree, node_t* node);*
- **Fila.c**
- **Lista.c**
- **No.c**

```c
tree_t* create_huffmanTree (fila_t* Q){
    if (Q == NULL){
        perror("Erro huffman->create_huffmanTree: ponteiro invalido");
        exit(-1);
    }

    tree_t* tree = create_tree(comp_symbol);
    symbol_t* s;
    node_t* n;
    node_t* n2;
    node_t* f;

    while(fila_tamanho(Q)>1){
        s = malloc(sizeof(symbol_t));

        n = dequeue(Q);
        n2 = dequeue(Q);
        s->qty = get_qty(node_get_data(n)) + get_qty(node_get_data(n2));

        f = create_t_node(s);
        tree_add_node(tree, f, n);
        tree_add_node(tree, f, n2);

        enqueue(f, Q);
    }

    n = dequeue(Q);

    tree_add_root(tree, n);

    return tree;
```

Complexidade

Fila	**$O(n^2)$**
Árvore	**$O(m*\log n)$**
Total	**$O(n^2 + m*\log n)$**