**Compactação:**

20| FILE \*arquivo = **fopen**(nome\_do\_arquivo, "rb");

+ FILE \* cria um ponteiro para um arquivo desejado.

+ fopen = função em C que abre um arquivo externo, recebe como parâmetro: (nome\_do\_arquivo, modo de abertura).

+ rb = read binary.

30| HT \*ht = **criar\_hash\_table**();

31| **adicionar\_frequencia**(arquivo, ht);

+ Após criar o ponteiro para o arquivo txt e uma hash table, passamos ambos para uma função que irá ler byte a byte do txt e registrará a frequência.

32| **rewind**(arquivo);

+ Traz o indicador do ponteiro de volta ao início do txt após percorrê-lo.

34| FILA \*fila = **criar\_fila**();

35| fila = **fila\_prioridade**(ht, fila);

+ Com a hash table preenchida, criamos uma fila de prioridade da menor até a maior frequência, a partir dela criaremos a árvore.

37| **criar\_arvore\_huffman**(fila);

38| NO \*arvore = fila->cabeca;

+ A partir da fila realizamos o procedimento de montagem da árvore, depois disso, definimos a raiz da árvore como sendo a cabeça da fila por ela ser exatamente o único elemento remanescente dela.

40| **criar\_caminho**(arvore, ht, caminho, 0);

+ Criamos o caminho, ex: 01, 101... para cada elemento **FOLHA** da árvore.

44| lixo = **calcula\_tamanho\_lixo**(ht);

45| inteiro = lixo;

46| **calcula\_tamanho\_arvore**(arvore, &tamanho);

+ Definimos a quantidade de bits do arquivo correspondente a lixo a partir da hash table, nela contém o caminho de cada folha e sua frequência.

+ Calculamos o tamanho da árvore passando ela e o ponteiro de tamanho.

48| inteiro = inteiro << 13;

+ Movemos o valor do lixo em 13 bits para que assuma a posição padrão do header do huffman, ex: lixo = 111, após isso = 11100000 00000000.

49| inteiro = **setar\_bits**(inteiro, &tamanho);

+ Passamos o tamanho da árvore e o tamanho do lixo para a função de junção delas nos bytes, ex: lixo = 11100000 00000000, tamanho = \_\_\_00000 00001101, resultado = 11100000 00001101.

51| \*nome\_do\_arquivo = **\*strtok**(nome\_do\_arquivo, ".");

+ strtok: é usada para dividir uma string em tokens com base em um caractere delimitador escolhido, então ela é dividida em duas partes, a primeira (até o ponto) continua armazenada no nome\_do\_arquivo.

52| \*nome\_do\_arquivo = **\*strcat**(nome\_do\_arquivo, ".huff");

+ strcat: é usada para concatenar a nova extensão “.huff” com a primeira parte do nome do arquivo, para que a string final tenha a extensão correta.

53| FILE \*saida = **fopen**(nome\_do\_arquivo, "wb");

+ Criação e abertura do arquivo de saida em binário.

55| auxiliar = inteiro >> 8;

+ Criamos um auxiliar para guardar somente o valor do primeiro byte do header.

56| **fputc**(auxiliar, saida);

57| **fputc**(inteiro, saida);

+ Anexamos o primeiro e o segundo byte do header no arquivo.

58| **imprimir\_pre\_ordem**(saida, arvore);

+ Lê a árvore em pré-ordem e armazena na header do arquivo.

59| **fseek**(saida, (2 + tamanho), SEEK\_SET);

+ fseek: função usada para definir a posição do ponteiro do arquivo, seleciona o arquivo de saída, pula os bits de tamanho, em relação ao início do arquivo (seek\_set).

60| **imprimir\_bits**(arquivo, saida, ht);

+ Recebe como parâmetro: o arquivo inicial a ser lida, o arquivo final a ser escrito e a hash table para correlacionar cada byte inicial, ao binário final.

**Descompactação:**

33| **fseek**(compactado, 0, SEEK\_END);

+ Posiciona o ponteiro do arquivo criado no fim (seek\_end) do txt.

35| tamanho\_arquivo = **ftell**(compactado);

+ Com o ponteiro posicionado ao final do arquivo a função ftell retorna a posição do ponteiro, que nesse caso corresponde ao tamanho do arquivo.

38| primeirobyte = **fgetc**(compactado);

40| segundobyte = **fgetc**(compactado);

+ Pega o primeiro e o segundo char do arquivo, que formam o header.

44| tamanho\_lixo = primeirobyte >> 5;

+ Faz o shift bit no primeiro byte pra armazenar o tamanho do lixo

45| tamanho\_arvore = ((primeirobyte << 8) | segundobyte) & tamanho\_arvore;

+ Define o tamanho do lixo a partir do primeiro e segundo byte, como o “tamanho\_arvore” foi definido como 8191 = 00011111 11111111, a operação & faz com que a variável não extrapole esse limite.

47| raiz = **montagem\_arvore**(compactado);

+ A função montagem\_arvore monta ela a partir do header em pré-ordem e retorna a raiz dela pro nó criado previamente “raiz”

48| **fseek**(compactado, tamanho\_arvore + 2, SEEK\_SET);

+ Setta o ponteiro para o tamanho da árvore + 2 (bytes do header), a partir do início do arquivo para iniciar a leitura.

49| **printar\_byte**(compactado, descompactado, raiz, tamanho\_arvore, tamanho\_lixo, (tamanho\_arquivo-tamanho\_arvore-2));

+ Com o arquivo de leitura e o de escrita abertos passamos o tamanho da árvore e do lixo + o tamanho do arquivo, no 1 for percorremos byte a byte, caso não seja o byte final no 2 for percorremos bit a bit e procuramos folhas na árvore