Considerações sobre a implementação

Huffman

Para a implementação da árvore de Huffman, foi utilizado um vector para criar a tabela e a arvore. Os pais e filhos são definidos apenas como índices para outros elementos do vetor. Para a montagem da árvore foi utilizada a seguinte ideia: quando dois nós são unidos, eles passam a ter pai. O pai é inserido no final do vetor, o que significa que as folhas vão estar sempre na primeira metade do vetor, a raiz será o último campo, e os nós intermediários entre a raiz e as folhas. Assim, ao invés de remover os nós da lista, é feita uma verificação; se o nó tem pai ele não é um candidato para a união. Caso fosse aceito, um nó poderia ter mais de um pai ou teria apenas o ultimo pai considerado.
Um problema encontrado na hora de codificar usando huffman, é como buscar o caractere que desejamos codificar entre as folhas. Para resolver este problema, procuramos qual folha possui o caractere que desejamos codificar, e então subimos na árvore em direção a raiz. Dessa forma, o código(caminho) encontrado está invertido e portanto é necessário inverte-lo para termos o caminho(código) da raiz até a folha correspondente ao caractere que desejamos codificar.

arvoreB

Para a implementação da árvoreB, foi utilizada a ideia de página cheia e lotada. Cada página é composta por um vetor de chaves com 2*d+1 posições, onde d é a ordem da árvore, e um vetor de ponteiros para páginas filhas, com 2*d+2 posições. Uma página cheia respeita as propriedades de árvoreB referentes ao número mínimo, e máximo, de chaves e de ponteiros em uma única página. Para a implementação do trabalho foi considerado que cada página diferente da raiz precisa ter (d - 2*d) chaves e (d+1 - d*2+1) ponteiros para outras páginas(exceto as folhas).

Uma página lotada, é uma página que estava cheia e recebeu uma nova chave, com um novo ponteiro(pode ser NULL caso a insersão seja em uma folha). A nova chave e o novo ponteiro são alocados nas últimas posições dos vetores chaves e ponteiros. Desta forma, uma página lotada não cumpre as propriedades de número máximo de chaves e ponteiros, porém, estas posições servem apenas como uma região de overflow e os dados serão removidos o mais rápido possível. Após inserir uma nova chave, é realizada uma verificação para saber se foi usada a região de overflow. Caso tenha, é realizado o split na página lotada. Após o split, teremos duas páginas com d chaves, e (d+1) ponteiros, e o pai da página que sofreu split terá mais uma chave e mais um ponteiro, e a verificação de página lotada é realizada sucessivamente.

Para uma árvore de ordem 3, uma página está cheia quando possui 6 chaves e 7 filhas e lotada quando possui 7 chaves e 8 filhas. Como a página lotada não respeita as propriedades, realizaremos o split dela e assim teremos: duas páginas com 3 chaves e 4 filhas cada, e a página pai recebe uma chave e um ponteiro.

arvoreVP

Para a implementação da árvore vermelha e preta, utilizei o nó sentinela. O nó sentinela é filho de todos as folhas, e pai da raiz. Para verificar se um nó é folha, basta pegar seus dois filhos e ver se possuem a cor 2. Um nó é a raiz caso seu pai seja a sentinela. O nó sentinela tem uma cor "especial" atribuida no código para ele(2), mas para calcular a altura negra da árvore ele seria considerado preto.

A insersão é feita sempre inserindo um nó vermelho porque assim garantimos que o novo nó só pode ter alterado as propriedades de nó vermelho com pai vermelho ou raiz vermelha.