| */\* Busca o índice de inserção utilizando busca binária \*/* int BuscaIndiceArvoreB(struct NoArvoreB \*p, int \*valor){  int e = -1, d = p->numero\_chaves, m;  while(e < d-1){  m = (e+d)/2;  if(p->chaves[m] < \*valor){  e = m;  }  else{  d = m;  }  }  return d; } |
| --- |

Na função de busca utiliza-se a busca binária para encontrar o valor dentro da página. Atribui-se o número de chaves existentes na árvore na variável “d”, e o índice -1 na variável “e”.

Em seguida, são realizadas iterações até que e seja menor que d-1.

Em cada iteração soma-se “e” e “d” para obter o valor de “m”, então compara-se o valor armazenado na página com índice “m” com o valor desejado.

Caso o valor armazenado na página seja menor que o valor desejado, ele está à direita na página e portanto o valor inferior “e” recebe “m”.

Caso o valor armazenado na página seja maior ou igual o valor desejado, ele está à esquerda na página e portanto o valor superior “d” recebe “m”.

| void InsereArvB(NoArvoreB\_ptr \*Ap, int \*valor) {  bool propagar;  NoArvoreB\_ptr ApRet, ApTemp;   InsereRecArvB(\*Ap, &ApRet, valor, &propagar);  if (propagar) {  ApTemp = (NoArvoreB\_ptr) malloc(sizeof(NoArvoreB));  ApTemp->numero\_chaves = 1;  ApTemp->chaves[0] = \*valor;  ApTemp->filhos[1] = ApRet;  ApTemp->filhos[0] = \*Ap;  \*Ap = ApTemp;  } } |
| --- |

Na função Insere, recebe-se como parâmetro a árvore e o valor a ser inserido e então chama-se o Insere Recursivo, que faz diversas chamadas de inserção até retornar. Quando as inserções são concluídas e ocorre a necessidade de propagar para o nó raiz, ele entra dentro do propagar.

No propagar, ele cria um novo nó raiz, atribui uma chave com o valor retornado pelo insere recursivo e atribui os filhos esquerdo e direito nesse novo nó raiz.

| bool InsereRecArvB(NoArvoreB\_ptr Ap, NoArvoreB\_ptr \*ApRet, int \*valor, bool \*propagar) {  int i, j;  NoArvoreB\_ptr ApTemp;   if (Ap == NULL) {  \*propagar = true;  (\*ApRet) = NULL;  return true;  }   i = BuscaIndiceArvoreB(Ap, valor);  if (i < Ap->numero\_chaves && \*valor == Ap->chaves[i]) {  printf(" Erro: valor já existe\n");  \*propagar = false;  return false;  }   bool inseriu = InsereRecArvB(Ap->filhos[i], ApRet, valor, propagar);  if (\*propagar) {  if (Ap->numero\_chaves < REGISTROS) {  InsereNaPagina(Ap, \*valor, \*ApRet);  \*propagar = false;  } else {  ApTemp = (NoArvoreB\_ptr) malloc(sizeof(NoArvoreB));  ApTemp->numero\_chaves = 0;  ApTemp->filhos[0] = NULL;  if (i < ORDEM + 1) {  InsereNaPagina(ApTemp, Ap->chaves[REGISTROS-1], Ap->filhos[REGISTROS]);  Ap->numero\_chaves--;  InsereNaPagina(Ap, \*valor, \*ApRet);  } else {  InsereNaPagina(ApTemp, \*valor, \*ApRet);  }  for (j = ORDEM + 2; j <= REGISTROS; j++) {  InsereNaPagina(ApTemp, Ap->chaves[j - 1], Ap->filhos[j]);  }  Ap->numero\_chaves = ORDEM;  ApTemp->filhos[0] = Ap->filhos[ORDEM+1];  \*valor = Ap->chaves[ORDEM];  \*ApRet = ApTemp;  }  }   return inseriu; } |
| --- |

Na função de InsereRecursivo temos diversos elementos importantes. Declara-se Ap como apontador do nó raiz, porém este ponteiro também é usado para transmitir de volta os elementos recursivamente, assim como um apontador auxiliar ApRet, um ponteiro para o valor e um ponteiro para informar se houve propagação.

Primeiro verificamos se Ap é nulo, indicando que este é um critério de parada da recursão pois atingiu um nó folha. Caso nulo, a função InsereRecursiva retorna para a função InsereRecursiva que a chamou, indicando que há necessidade de propagação.

Em seguida, busca-se o índice para inserção dentro da árvore utilizando o algoritmo de busca binária. Localizado o índice, verificamos se o valor já existe dentro da árvore, caso já exista a recursão é interrompida e nenhum valor é inserido (retornando false e não propagando).

Logo depois, o InsereRecursivo é chamado apontando para os filhos da página atual. Esta função é utilizada para descer na árvore até encontrar a posição de inserção (quando Ap == NULL) , indicando que trata-se do nó folha que chamou esta última recursão. Portanto, ao retornar, ocorre a propagação e a inserção do valor na página no nível inferior.

Na propagação, ocorre a verificação:

Se a página contém menos registros do que o número máximo de registros no nó (2\*B), onde B é a ordem.

-> Ocorre a inserção e não há necessidade de propagação pois não alterou a altura

Se a página contém o número máximo de registros (2B)

-> Ocorre a inserção e há necessidade de propagação

Na propagação, criamos uma página vazia e atribuímos seu filho esquerdo como nulo. Então verificamos a posição de inserção do novo valor.

Se a posição de inserção for menor que ORDEM + 1, isso significa que a inserção ocorrerá na metade esquerda da página. Nesse caso, o valor mais à direita da página atual é movido para a nova página ApTemp, e o valor a ser inserido é colocado na página atual. A página atual então perde um registro (pois um deles foi movido para ApTemp, e o valor a ser propagado para cima na árvore é o valor que foi inserido.

Se a posição de inserção for maior ou igual a ORDEM + 1, isso significa que a inserção ocorrerá na metade direita da página. Nesse caso, o valor a ser inserido é colocado diretamente na nova página ApTemp.

Em seguida, os valores restantes na metade direita da página atual são movidos para a nova página ApTemp. A página atual então perde metade de seus registros, que foram movidos para ApTemp. O valor a ser propagado para cima na árvore é o último valor que permanece na página atual.

Então, o filho direito da página atual é movido para ser o filho esquerdo da nova página ApTemp, e a nova página ApTemp é retornada para ser o novo filho direito da página atual.

Se a propagação continuar até a raiz da árvore, uma nova raiz é criada e o valor propagado é colocado nessa nova raiz. A árvore então aumenta em altura.

| */\* Insere um valor na página \*/* void InsereNaPagina(NoArvoreB\_ptr Ap, int valor, NoArvoreB\_ptr ApDir) {  int posicao = BuscaIndiceArvoreB(Ap, &valor);   for(int i = Ap->numero\_chaves; i >= posicao ; i--){  Ap->chaves[i] = Ap->chaves[i-1];  Ap->filhos[i+1] = Ap->filhos[i];  }   Ap->chaves[posicao] = valor;  Ap->filhos[posicao+1] = ApDir;  Ap->numero\_chaves++; } |
| --- |

A função InsereNaPagina busca o índice de inserção utilizando a função de busca, então ela desloca todas as chaves começando do número de chaves até a posição de inserção. Ela “puxa” as chaves para direita e os filhos para a direita logo depois do ponto de inserção, liberando espaço para a inserção no local correto. Então é realizada a inserção do valor na posição, acrescentado o número de chaves na página e inserido o ponteiro do seu filho direito.

| */\* Remove um valor da árvore \*/ void RemoveArvoreB(NoArvoreB\_ptr \*Ap, int valor) {  bool propagar;  NoArvoreB\_ptr Aux;  RemoveRecArvoreB( p, valor, &propagar);  if (propagar && (\*Ap)->numero\_chaves == 0) {  Aux = \*Ap;  \*Ap = Aux->filhos[0];  free(Aux);  } }* |
| --- |

A função de remoção tem o princípio inverso da inserção. A propagação ocorre fazendo remoções até atingir a raiz. Caso a remoção se propague para a raiz, esta é removida.

| *void RemoveRecArvoreB(NoArvoreB\_ptr \*Ap, int valor, bool \*propagar) {  int j, i = 1;  NoArvoreB\_ptr Pag;  if (\*Ap == NULL) {  printf("Erro: registro nao esta na arvore\n");  \*propagar = false;  return;  }  Pag = \*Ap;  while (i < Pag->numero\_chaves && valor > Pag->chaves[i-1]) i++;  if (valor == Pag->chaves[i-1]) {  if (Pag->filhos[i-1] == NULL){  Pag->numero\_chaves--;  \*propagar = (Pag->numero\_chaves < ORDEM);  for (j = i; j <= Pag->numero\_chaves; j++){  Pag->chaves[j-1] = Pag->chaves[j];  Pag->filhos[j] = Pag->filhos[j+1];  }  return;  }  Antecessor(\*Ap, i, Pag->filhos[i-1], propagar);  if (\*propagar) {  Reconstitui(Pag->filhos[i - 1], \*Ap, i - 1, propagar);  }  return;  }  if (valor > Pag->chaves[i-1]){  i++;  }  RemoveRecArvoreB(&Pag->filhos[i-1], valor, propagar);  if (\*propagar) {  Reconstitui(Pag->filhos[i-1], \*Ap, i - 1, propagar);  } }* |
| --- |

Na função RemoveRecursivo recebe-se como parâmetro um ponteiro para a árvore, o valor a ser removido por parâmetro e uma variável booleana para indicar se existe a necessidade de propagação da da remoção na árvore para cima.

Primeiramente, a função verifica se a página é nula. Caso esta seja nula, a árvore inteira já foi percorrida e nenhum valor foi encontrado, retornando a mensagem de que o registro não está na árvore.

Em seguida, caso este não seja nulo a função procura o valor na página percorrendo todas as chaves na página até encontrar o valor ou até que todas as chaves tenham sido verificadas.

Caso o valor seja encontrado a função verifica se a página é uma folha e chama a função Antecessor para encontrar o valor que deve substituir o valor a ser removido, em seguida chama a função Reconstitui para reorganizar a árvore após a remoção.

Se o valor não for encontrado no nó atual, a função chama a si mesma recursivamente para o filho apropriado do nó atual.

Após a chamada recursiva, a função verifica novamente a variável propagar. Se a remoção deve ser propagada para cima na árvore, a função Reconstitui é chamada novamente.