LISTA DE CÓDIGOS

SUMÁRIO

1 Árvore B

```c
#ifndef BTREE_H
#define BTREE_H
// O máximo de chaves em um nó é 3 (grau máximo).
// Como t = (MAX+1)/2 = 2, cada nó pode ter entre 1 e 3 chaves.
// Exemplo: um nó pode ter {10, 20, 30}.

#define MAX 3

// Estrutura de um nó da árvore B
typedef struct No {
    int n;                 // Número atual de chaves armazenadas no nó
    int folha;             // 1 se for folha, 0 caso contrário
    int chave[MAX];        // Vetor de chaves (ordenadas)
    struct No *filho[MAX + 1]; // Ponteiros para os filhos (um a mais que o número de chaves)
} NoArvB, *ArvoreB;

int BuscaArvoreB(ArvoreB r, int k);
ArvoreB InsereArvoreB(ArvoreB r, int k);
void RemoveArvoreB(ArvoreB r, int k);
NoArvB *AllocateNode();
void SplitChildArvoreB(NoArvB *x, int i);
void InsereNaoCheioArvoreB(ArvoreB x, int k);
void PrintBTree(ArvoreB r, int level);
#endif
```

Código 1: Header Árvore B

```c
int BuscaArvoreB(ArvoreB r, int k) {
    if (r == NULL) {
        return -1; // Árvore vazia, chave não encontrada
    }

    int i = 0;
    // Avança até encontrar uma chave >= k ou acabar as chaves
    while (i < r->n && k > r->chave[i]) {
        i++;
    }

    if (i < r->n && k == r->chave[i]) {
        return r->chave[i]; // Chave encontrada
    } else if (r->folha) {
        return -1; // Não encontrou e é folha
    } else {
        // Continua a busca no filho apropriado
        return BuscaArvoreB(r->filho[i], k);
    }
}

/*
Exemplo:
Árvore:
           [20]
          /    \
      [10]      [30,40]

```

```
BuscaArvoreB(r, 40):
- r->chave[0] = 20 < 40 → vai para filho[1]
- no filho: encontra 40 → retorna 40
*/
```

Código 2: Busca em Árvore B

```
ArvoreB InsereArvoreB(ArvoreB r, int k) {
    int t = (MAX+1)/2;
    if (r->n == 2 * t - 1) { // raiz cheia
        NoArvB *s = AllocateNode();
        s->folha = 0;
        s->n = 0;
        s->filho[0] = r;
        SplitChildArvoreB(s, 0);
        InsereNaoCheioArvoreB(s, k);
        return s; // nova raiz
    } else {
        InsereNaoCheioArvoreB(r, k);
        return r;
    }
}

/*
Exemplo:
Raiz cheia [10,20,30]
Inserir(40):
- Cria nova raiz vazia
- Divide [10,20,30] em [10] e [30], sobe 20
- Nova raiz = [20], filhos [10], [30]
- Agora insere 40 em [30] → [30,40]
*/

NoArvB *AllocateNode() {
    NoArvB *z = malloc(sizeof(NoArvB));
    if (!z) return NULL;
    z->n = 0;        // Nó inicialmente sem chaves
    z->folha = 1;    // Por padrão é folha
    for (int j = 0; j < MAX + 1; j++) {
        z->filho[j] = NULL;
    }
    return z;
}

/*
Exemplo:
NoArvB *n = AllocateNode();
n é um nó folha vazio: { } com filhos NULL
*/

void SplitChildArvoreB(NoArvB *x, int i) {
    int t = (MAX+1)/2; // aqui t = 2
    NoArvB *z = AllocateNode();
    NoArvB *y = x->filho[i]; // filho cheio que vai ser dividido
    z->folha = y->folha;
    z->n = t - 1; // novo nó terá t-1 chaves

    // Copia as últimas t-1 chaves de y para z
    for (int j = 0; j < t - 1; j++) {
```

```
53          z->chave[j] = y->chave[j + t];
54      }
55
56      // Se não é folha, também copia os filhos
57      if (!y->folha) {
58          for (int j = 0; j < t; j++) {
59              z->filho[j] = y->filho[j + t];
60          }
61      }
62
63      y->n = t - 1; // reduz número de chaves de y
64
65      // Move filhos de x para abrir espaço
66      for (int j = x->n; j >= i + 1; j--) {
67          x->filho[j + 1] = x->filho[j];
68      }
69      x->filho[i + 1] = z;
70
71      // Move chaves de x para abrir espaço
72      for (int j = x->n - 1; j >= i; j--) {
73          x->chave[j + 1] = x->chave[j];
74      }
75
76      // Sobe a chave mediana
77      x->chave[i] = y->chave[t - 1];
78      x->n++;
79  }
80
81  /*
82  Exemplo:
83  y = [10,20,30] cheio
84  Split → y = [10], z = [30], sobe 20 para pai
85  Pai antes: [ ] → depois: [20]
86  */
87
88  void InsereNaoCheioArvoreB(ArvoreB x, int k) {
89      int t = (MAX+1)/2;
90      int i = x->n - 1;
91
92      if (x->folha) {
93          // Desloca chaves maiores que k
94          while (i >= 0 && k < x->chave[i]) {
95              x->chave[i + 1] = x->chave[i];
96              i--;
97          }
98          // Insere k
99          x->chave[i + 1] = k;
100         x->n++;
101     } else {
102         // Acha o filho correto
103         while (i >= 0 && k < x->chave[i]) {
104             i--;
105         }
106         i++;
107         // Se filho está cheio, divide
108         if (x->filho[i]->n == 2 * t - 1) {
109             SplitChildArvoreB(x, i);
110             if (k > x->chave[i]) {
```

```
                i++;
            }
        }
        // Insere recursivamente
        InsereNaoCheioArvoreB(x->filho[i], k);
    }
}

/*
Exemplo:
Raiz [20], filhos [10,15] e [30,40]
InsereNaoCheioArvoreB(r, 25):
→ vai pro filho direito [30,40]
→ insere 25 deslocando: [25,30,40]
*/
```

Código 3: Inserção em Árvore B

```
void PrintBTree(ArvoreB r, int level) {
    if (r != NULL) {
        printf("Nível %d: [", level);
        for (int i = 0; i < r->n; i++) {
            printf("%d", r->chave[i]);
            if (i < r->n - 1) printf(", ");
        }
        printf("] (folha: %s)\n", r->folha ? "sim" : "não");

        if (!r->folha) {
            for (int i = 0; i <= r->n; i++) {
                PrintBTree(r->filho[i], level + 1);
            }
        }
    }
}
```

Código 4: Imprime estrutura da Árvore B

2 Árvore Radix (Árvores digitais de busca)

```
#ifndef RADIX_H
#define RADIX_H

#define bits_na_chave 32  // Número de bits na chave (unsigned int)

// Estrutura de um nó da árvore Patricia
typedef struct No {
    unsigned chave;      // Valor armazenado no nó
    int bit;             // Índice do bit usado para decisão (não usado
   na versão simplificada)
    struct No *esq;      // Filho à esquerda (bit 0)
    struct No *dir;      // Filho à direita (bit 1)
} No;

// Funções da árvore Patricia
void inicializa(No **arvore);
No *busca(No *arvore, unsigned x);
void insere(No **arvore, unsigned chave);
```

```

#endif
```

Código 5: Header da Árvore Radix

```
#include "radix.h"
#include <limits.h>
#include <stdio.h>
#include <stdlib.h>

// Inicializa a árvore Patricia
void inicializa(No **arvore) {
    *arvore = malloc(sizeof(No));    // Aloca nó raiz
    (*arvore)->chave = UINT_MAX;     // Marca com chave "inválida" (raiz fictícia)
    (*arvore)->esq = NULL;           // Inicialmente sem filhos
    (*arvore)->dir = NULL;
    (*arvore)->bit = -1;             // Nível raiz
}

/*
Exemplo:
No *raiz;
inicializa(&raiz);
raiz:
chave = UINT_MAX
esq = dir = NULL
bit = -1
*/
```

Código 6: Inicialização de Árvore Radix

```
// Função que retorna o k-ésimo bit da chave
unsigned bit(unsigned chave, int k) {
    // bits_na_chave - 1 - k: começa do bit mais significativo
    // & 1: retorna apenas 0 ou 1
    return (chave >> (bits_na_chave - 1 - k)) & 1;
}

/*
Exemplo:
chave = 5 (000...0101 em 32 bits)
bit(chave, 0) = 0  -> bit mais significativo
bit(chave, 30) = 1 -> penúltimo bit
bit(chave, 31) = 1 -> último bit
*/
```

Código 7: Checagem de bit em Árvore Radix

```
// Função de busca "externa"
No *busca(No *arvore, unsigned x) {
    return busca_rec(arvore, x, 0); // Inicia do nível 0
}


// Função recursiva de busca
No *busca_rec(No *arvore, unsigned x, int nivel) {
    if (arvore == NULL) {
        return NULL; // Não encontrou
```

```
    }
    if (x == arvore->chave) {
        return arvore; // Encontrou a chave
    }

    // Escolhe o caminho de acordo com o bit atual
    if (bit(x, nivel) == 0) {
        return busca_rec(arvore->esq, x, nivel + 1); // Vai para a
    esquerda
    } else {
        return busca_rec(arvore->dir, x, nivel + 1); // Vai para a
    direita
    }
}

/*
Exemplo:
Árvore:
          [5]
         /   \
      [2]     [7]

busca_rec(raiz, 7, 0)
- nível 0: bit(7,0) = 0 ou 1? 1 → vai para dir
- nível 1: checa chave → retorna nó com chave 7
*/
```

Código 8: Busca em Árvore Radix

```
// Função de inserção "externa"
void insere(No **arvore, unsigned chave) {
    *arvore = insere_rec(*arvore, chave, 0); // Inicia do nível 0
}

// Função recursiva de inserção
No *insere_rec(No *arvore, unsigned chave, int nivel) {
    No *novo;

    if (arvore == NULL) {
        // Nó vazio → cria novo nó
        novo = malloc(sizeof(No));
        novo->esq = novo->dir = NULL;
        novo->chave = chave;
        return novo;
    }

    if (chave == arvore->chave) {
        // Chave já existe → não insere duplicata
        return arvore;
    }

    // Escolhe caminho com base no bit
    if (bit(chave, nivel) == 0) {
        arvore->esq = insere_rec(arvore->esq, chave, nivel + 1);
    } else {
        arvore->dir = insere_rec(arvore->dir, chave, nivel + 1);
    }

    return arvore;
```

```
}

/*
Exemplo:
Inserindo 5, 2, 7 na árvore vazia
1) insere(&raiz,5) → cria nó raiz com 5
2) insere(&raiz,2) → bit(2,0)=0 → esquerda → cria nó com 2
3) insere(&raiz,7) → bit(7,0)=1 → direita → cria nó com 7
Árvore final:
         [5]
        /   \
     [2]     [7]
*/
```

Código 9: Inserção em Árvore Radix

```
#ifndef PATRICIA_TRIE_H
#define PATRICIA_TRIE_H

#define bits_na_chave 32  // número de bits de cada chave

// Estrutura de um nó da árvore Patricia
typedef struct No {
    unsigned chave;      // valor armazenado no nó
    int bit;             // índice do bit usado para decisão
    struct No *esq;      // ponteiro para filho esquerdo (bit 0)
    struct No *dir;      // ponteiro para filho direito (bit 1)
} No;

// Funções da árvore Patricia
void inicializa(No **arvore);
unsigned bit(unsigned chave, int k);
No *busca(No *arvore, unsigned x);
No *busca_rec(No *arvore, unsigned x, int w);
void insere(No **arvore, unsigned chave);
No *insere_rec(No *arvore, unsigned chave, int w, No *pai);

#endif
```

Código 10: Header da Árvore Patricia Trie

3 Árvore Patricia Trie

```
#include "patricia_trie.h"
#include <stdio.h>
#include <stdlib.h>
#include <limits.h>

// Inicializa a árvore Patricia
void inicializa(No **arvore) {
    *arvore = malloc(sizeof(No));
    (*arvore)->chave = UINT_MAX; // raiz fictícia
    (*arvore)->esq = (*arvore)->dir = *arvore; // aponta para si mesmo (
    ciclo)
    (*arvore)->bit = -1; // bit inicial
}

```

```
/*
Explicação:
- Raiz é fictícia e cria um ciclo para simplificar inserção/busca
- Esse nó nunca contém chave real
*/
```

Código 11: Inicialização de Árvore Patricia Trie

```
// Retorna o k-ésimo bit (do mais significativo para o menos
    significativo)
unsigned bit(unsigned chave, int k) {
    return (chave >> (bits_na_chave - 1 - k)) & 1;
}

/*
Exemplo:
chave = 5 (000...0101)
bit(chave,0) = 0 -> bit mais significativo
bit(chave,31)=1 -> bit menos significativo
*/
```

Código 12: Checagem de bit em Árvore Patricia Trie

```
// Busca "externa" que retorna NULL se não encontrou
No *busca(No *arvore, unsigned x) {
    No *t = busca_rec(arvore->esq, x, -1); // inicia do filho esquerdo
    da raiz fictícia
    return t->chave == x ? t : NULL; // verifica se realmente encontrou
}

// Busca recursiva
No *busca_rec(No *arvore, unsigned x, int w) {
    // Se chegamos em um nó com bit <= w, paramos (não desce mais)
    if (arvore->bit <= w) {
        return arvore;
    }

    // Escolhe direção pelo bit atual
    if (bit(x, arvore->bit) == 0) {
        return busca_rec(arvore->esq, x, arvore->bit);
    } else {
        return busca_rec(arvore->dir, x, arvore->bit);
    }
}

/*
Exemplo de fluxo:
- arvore->bit indica o bit a ser checado
- w mantém bit do nó pai para não descer demais
*/
```

Código 13: Busca em Árvore Patricia Trie

```
// Inserção externa
void insere(No **arvore, unsigned chave) {
    int i;
    // busca nó mais próximo
    No *t = busca_rec((*arvore)->esq, chave, -1);

```

```
    // Se já existe, não insere duplicata
    if (chave == t->chave) return;

    // Encontra o primeiro bit diferente entre a chave nova e a existente
    for (i = 0; bit(chave, i) == bit(t->chave, i); i++);

    // Insere recursivamente
    (*arvore)->esq = insere_rec((*arvore)->esq, chave, i, *arvore);
}

/*
Exemplo:
- chave = 5, t->chave = 3
- compara bits de 0 a 31 até encontrar diferença
- w = índice do bit onde as chaves diferem
- insere_rec cria novo nó nesse ponto
*/

// Inserção recursiva avançada
No *insere_rec(No *arvore, unsigned chave, int w, No *pai) {
    // Se nó atual está no nível >= w ou nó abaixo do pai, cria nó novo
    if ((arvore->bit >= w) || (arvore->bit <= pai->bit)) {
        No *novo = malloc(sizeof(No));
        novo->chave = chave;
        novo->bit = w;

        // Define filhos do novo nó baseado no bit da chave
        if (bit(chave, novo->bit) == 1) {
            novo->esq = arvore; // 0 aponta para nó antigo
            novo->dir = novo;   // 1 aponta para si mesmo
        } else {
            novo->esq = novo;   // 0 aponta para si mesmo
            novo->dir = arvore; // 1 aponta para nó antigo
        }
        return novo;
    }

    // Recursão: desce para esquerda ou direita
    if (bit(chave, arvore->bit) == 0) {
        arvore->esq = insere_rec(arvore->esq, chave, w, arvore);
    } else {
        arvore->dir = insere_rec(arvore->dir, chave, w, arvore);
    }

    return arvore; // garante retorno em todos os caminhos
}

/*
Exemplo:
- Inserindo 5 em árvore com raiz fictícia
- Busca nó mais próximo → t
- Determina bit de diferença → w
- insere_rec cria nó novo no ponto correto
- Faz referência correta para nó existente e para o novo nó
*/
```

Código 14: Inserção em Árvore Patricia Trie