Estruturas

Eduardo Freitas Fernandes

Março 2025

1 Resolução

Exercício 1

```c
#define SIZE 1009

typedef struct no {
    char matricula[6];
    struct no *next;
} No;

typedef No *Tabela[SIZE];
```

Implemente uma função `int hash(char matricula[6])` de hash razoável para este problema:

```c
int hash(char matricula[6]) {
    int hash = 0;
    for (int i = 0; i < 6; i++) {
        hash += matricula[i];
    }

    return hash;
}
```

Implemente a função `int insert(Tabela t, char matricula[6])` de inserção de uma matrícula na tabela, garantindo que não se armazenam matrículas repetidas.

```c
int insert(Tabela t, char matricula[6]) {
    int index = hash(matricula) % SIZE;

    No *temp = t[index];
    while (temp != NULL && strcmp(t, temp->matricula) != 0) {
        temp = temp->next;
    }

    int result = 0;
    if (temp == NULL) {
        temp = malloc(sizeof(No));
        if (temp == NULL) {
            result = 1;
```

```
        } else {
            strcpy(temp->matricula, matricula);
            temp->next = t[index];
            t[index] = temp;
            result = 0;
        }
    }

    return result;
}
```

Exercício 2

```
typedef struct no {
    int info;
    int bal;
    struct no *esq, *dir;
} No;

typedef No *Arvore;
```

Implemente a função `Arvore rr(Arvore arv)` que faz uma rotação simples para a direita numa determinada sub-árvore

```
Arvore rr(Arvore arv) {
    if (arv != NULL) {
        Arvore left = arv->esq;
        // if root is left and left sub tree is not left
        if (arv->bal < 0 && left->bal > -1) {
            arv->bal = 0; // BALANCED
        } else {
            arv->bal = 1; // RIGHT
        }

        left->bal = 1; // RIGHT

        arv->esq = left->dir;
        left->dir = arv;

        return left;
    }

    return NULL;
}
```

Exercício 3

```
void bals(Arvore a) {
    if (!a) return;
    a->bal = altura(a->dir) - altura(a->esq);
    bals(a->esq);
    bals(a->dir);
}
```

Complexidade da função `bals()`:

$$T(N) = \begin{cases} 0, & \text{se } N < 1 \\ 2 \times (N/2) + 2 \times \log N, & \text{se } N > 0 \end{cases}$$

$$T(N) = \sum_{i=1}^{\log N} N = O(N \times \log N)$$

Exercício 4

```c
typedef struct s {
    char *sin;
    struct s *next;
} Sin;

typedef struct p {
    char *pal;
    Sin *sins;
    struct p *next;
} Pal;

#define TAM ...
typedef Pal *Dic[TAM];

int hash(char *pal);
```

Implemente a função `void sinonimos(Dic d, char *pal)` que, dada uma palavra, imprime todos os seus sinónimos:

```c
void sinonimos(Dic d, char *pal) {
    if (d != NULL && pal != NULL) {
        int index = hash(pal) % TAM;

        Pal *temp = d[index];

        while (temp != NULL && strcmp(temp->pal, pal) != 0) {
            temp = temp->next;
        }

        // found the word
        if (temp != NULL) {
            Sin *aux = temp->sins;

            while (aux != NULL) {
                if (aux->sin != NULL)
                    printf("%s\n", aux->sin);
                aux = aux->next;
            }

        }
    }
}
```

Exercício 5

```c
#define TAM ...

typedef int Heap[TAM];

typedef struct nodo {
    int val;
    struct no *esq, *dir;
} Nodo, *Tree;
```

Implemente uma função que converte uma min-heap representada num array para uma min-heap representada como uma árvore do tipo indicado:

```c
Tree heap_to_tree(Heap h, int start) {
    Tree result = NULL;
    if (start < TAM && start >= 0) {
        result = malloc(sizeof(Nodo));

        if (result == NULL) {
            // error handling ??
            return NULL;
        }

        result->val = h[start];
        result->esq = heap_to_tree(h, 2 * start + 1);
        result->dir = heap_to_tree(h, 2 * start + 2);
    }

    return result;
}
```

Exercício 6

Exercício 7

```c
typedef struct node {
    into info;
    struct node *esq, *dir;
} *Node;
```

Escreva uma função `buildBST()` que, dado um array de inteiros ordenado por ordem crescente `arr` com `n` elementos, constrói uma árvore binária de procura balanceada com todos os elementos do array.

```c
Node buildBST(int arr[], int n) {

}
```

Exercício 8

Exercício 9

```
#define MaxH ...

typedef struct mHeap {
    int tamanho;
    int heap [MaxH];
} *MinHeap;
```

Defina uma função que altera o valor do elemento que está na posição `pos` para `valor`, fazendo as trocas necessárias para que se mantenham as propriedades da heap `h`.

```
void muda (MinHeap h, int pos, int valor) {

}
```

Exercício 10