

Hashing Aberto ou Encadeado: Exercício

Implemente uma função que especifique se uma determinada chave esta contida na tabela hashing aberta em questão.

```
Hash* localizarHash (Tabela tabela, int num)
```

```
{
```

```
    int pos = funcaoHashing (num);
```

```
    Hash* aux;
```

```
    if (tabela[pos] != NULL)
```

```
        if (tabela[pos]->chave==num)
```

```
            return (tabela[pos]);
```

```
        else
```

```
        {
```

```
            aux=tabela[pos]->prox;
```

Hashing Aberto ou Encadeado: Exercício

```
while (aux != NULL && aux ->chave != num)
    aux = aux->prox;
return (aux);
}
else
    return NULL;
}
```

Hashing Aberto ou Encadeado: Exercício

Implemente a função de remoção de uma chave na tabela hashing aberta em questão.

```
void excluirHash (Tabela tabela, int num) {  
    int pos = funcaoHashing (num);  
    Hash* aux;  
    if (tabela[pos] != NULL) {  
        if (tabela [pos] -> chave==num) {  
            aux = tabela [pos];  
            tabela [pos] = tabela [pos] ->prox;  
            free (aux);  
        }  
        else  
        {
```

```

Hash* ant=tabela [pos];
aux=tabela [pos]->prox;
while (aux != NULL && aux ->chave != num) {
    ant = aux;
    aux = aux->prox;
}
if (aux != NULL) {
    ant->prox = aux->prox;
    free (aux);
}
else
    printf("\nNumero nao encontrado");
}
}
else
    printf("\nNumero nao encontrado"); }

```

Hashing Aberto ou Encadeado: Exercício

Para uma melhor fixação do tópico em estudo com base na definição do TAD abaixo implemente suas funções ainda não definidas.

```
#define numEntradas 8
typedef struct _Hash
{
    int chave;
    struct _Hash *prox;
} Hash;
typedef Hash* TADTabelaHash[numEntradas];
void inicializarHash(TADTabelaHash);
int funcaoHashing(int);
void inserirHash(TADTabelaHash, int);
void mostrarHash(TADTabelaHash);
Hash* localizarHash (TADTabelaHash, int);
void excluirHash (TADTabelaHash, int);
void liberarMemoria(TADTabelaHash);
```

Tabelas de hash

Acabamos de estudar como implementar uma tabela hashing aberta e estudaremos agora como implementar uma tabela hashing fechada ou também denominada de tabela hashing com endereçamento aberto.

Neste tipo de implementação, a tabela hash é um vetor com m posições. Todas as chaves são armazenadas na própria tabela sem a necessidade de espaços extras ou ponteiros. Este método é aplicado quando o número de chaves a serem armazenadas é reduzido e as posições vazias na tabela são usadas para o tratamento de colisões.

Tabelas de hash

Quando uma chave x é endereçada na posição $h(x)$ e esta já está ocupada, outras posições vazias na tabela são procuradas para armazenar x . Caso nenhuma seja encontrada, a tabela está totalmente preenchida e x não pode ser armazenada.

Veremos duas maneiras de efetuar a busca por uma posição livre para armazenar x : tentativa linear e tentativa quadrática.

Tabelas de hash fechada – tentativa linear

Na tentativa linear, quando uma chave x deve ser inserida e ocorre uma colisão, a seguinte função é utilizada:

$$h'(x) = (h(x)+j) \text{ mod } m$$

para $1 \leq j \leq m-1$, sendo que $h(x) = x \text{ mod } m$

O objetivo é armazenar a chave no endereço consecutivo $h(x)+1$, $h(x)+2$, ..., até encontrar uma posição vazia.

Tabelas de hash fechada – tentativa linear

A operação de remoção é delicada, não se pode remover de fato uma chave do endereço, pois haveria perda da sequência de tentativas.

Com isso, cada endereço da tabela é marcado como livre (L), ocupado (O) ou removido (R). Livre quando a posição ainda não foi usada, ocupado quando uma chave está armazenada, e removido quando armazena uma chave que já foi removida. Para uma compreensão adequada do processo, analisaremos um exemplo.

Tabelas de hash fechada – tentativa linear

Visando manter um paralelo com o método usado anteriormente também nos utilizaremos de uma tabela com 8 entradas.

Inicialmente temos:

Índice	Situação	Chave
0	L	
1	L	
2	L	
3	L	
4	L	
5	L	
6	L	
7	L	

Inserir chave 16
 $16\%8=0$

Índice	Situação	Chave
0	O	16
1	L	
2	L	
3	L	
4	L	
5	L	
6	L	
7	L	

Tabelas de hash fechada – tentativa linear

Índice	Situação	Chave
0	O	16
1	L	
2	L	
3	L	
4	L	
5	L	
6	L	
7	L	

Inserir chave 23

$$23 \% 8 = 7$$

Índice	Situação	Chave
0	O	16
1	L	
2	L	
3	L	
4	L	
5	L	
6	L	
7	O	23

Tabelas de hash fechada – tentativa linear

Índice	Situação	Chave
0	O	16
1	L	
2	L	
3	L	
4	L	
5	L	
6	L	
7	O	23

Inserir chave 41

$$41 \% 8 = 1$$

Índice	Situação	Chave
0	O	16
1	O	41
2	L	
3	L	
4	L	
5	L	
6	L	
7	O	23

Tabelas de hash fechada – tentativa linear

Índice	Situação	Chave
0	O	16
1	O	41
2	L	
3	L	
4	L	
5	L	
6	L	
7	O	23

Inserir chave 25

$$25\%8=1$$

$$(1+1)\%8=2$$

Índice	Situação	Chave
0	O	16
1	O	41
2	O	25
3	L	
4	L	
5	L	
6	L	
7	O	23

Tabelas de hash fechada – tentativa linear

Índice	Situação	Chave
0	O	16
1	O	41
2	O	25
3	L	
4	L	
5	L	
6	L	
7	O	23

Inserir chave 39

$$39\%8=7$$

$$(7+1)\%8=0$$

$$(7+2)\%8=1$$

$$(7+3)\%8=2$$

$$(7+4)\%8=3$$

Índice	Situação	Chave
0	O	16
1	O	41
2	O	25
3	O	39
4	L	
5	L	
6	L	
7	O	23

Tabelas de hash fechada – tentativa linear

Índice	Situação	Chave
0	O	16
1	O	41
2	O	25
3	O	39
4	L	
5	L	
6	L	
7	O	23

Remover chave 41

$$41 \% 8 = 1$$

Índice	Situação	Chave
0	O	16
1	R	41
2	O	25
3	O	39
4	L	
5	L	
6	L	
7	O	23

Tabelas de hash fechada – tentativa linear

Índice	Situação	Chave
0	O	16
1	R	41
2	O	25
3	O	39
4	L	
5	L	
6	L	
7	O	23

Remover chave 23

$$23 \% 8 = 7$$

Índice	Situação	Chave
0	O	16
1	R	41
2	O	25
3	O	39
4	L	
5	L	
6	L	
7	R	23

Tabelas de hash fechada – tentativa linear

Índice	Situação	Chave
0	O	16
1	R	41
2	O	25
3	O	39
4	L	
5	L	
6	L	
7	R	23

Remover chave 25

$$25\%8=1$$

$$(1+1)\%8=2$$

Índice	Situação	Chave
0	O	16
1	R	41
2	R	25
3	O	39
4	L	
5	L	
6	L	
7	R	23

Tabelas de hash fechada – tentativa linear

Índice	Situação	Chave
0	O	16
1	R	41
2	R	25
3	O	39
4	L	
5	L	
6	L	
7	R	23

Inserir chave 34

$$34 \% 8 = 2$$

Índice	Situação	Chave
0	O	16
1	R	41
2	O	34
3	O	39
4	L	
5	L	
6	L	
7	R	23

Tabelas de hash fechada: Exercício

Com base no que foi apresentado defina a(s) estrutura(s) de dados necessária(s) para a implementação de uma tabela hashing fechada com tentativa linear.

```
#define tam 8  
typedef struct  
{  
    int chave;  
    char livre; /* L = livre, O = ocupado, R =  
removido*/  
}Hash;  
typedef Hash Tabela[tam];
```

Tabelas de hash fechada: Exercício

Agora, implemente a função de inserção de uma chave na tabela hashing fechada em questão.

```
void inserir(Tabela tabela, int n) {
    int i=0;
    int pos = funcaoHashing(n);
    while (i < tam && tabela[(pos+i)%tam].livre!='L'
    && tabela[(pos+i)%tam].livre !='R')
        i = i+1;
    if (i < tam) {
        tabela[(pos+i)%tam].chave = n;
        tabela[(pos+i)%tam].livre = 'O';
    }else
        printf ("\nTabela cheia!"); }
```

Tabelas de hash fechada: Exercício

```
int funcaoHashing(int num)
{
    return num % tam;
}
```

Tabelas de hash fechada: Exercício

Implemente a função de remoção de uma chave na tabela hashing fechada em questão.