Hash - Codificação

5. Códigos para tabela Hash

A codificação da estrutura de dados hash pode ser implementada de maneira relativamente simples, se usarmos as estruturas já implementadas de listas para gerenciar as colisões.

As operações sobre uma tabela hash são:

1. A criação de um hash vazio (criaHash)
2. A inserção de um elemento no hash (insereHash)
3. A busca de um elemento no hash (buscaHash)
4. A retirada de um elemento do hash (retiraHash)

Opcionalmente pode-se fazer uma operação para imprimir os elementos do hash (imprimeHash)

Para nosso exemplo, vamos usar os códigos que já temos de listas. Seguem válidas as mesmas definições de tipos de listas que usamos, e as rotinas já implementadas. Para isso, copiaremos o código das listas para dentro do mesmo código do hash. Vamos aqui destacar o que precisamos fazer além dos códigos de listas já implementadas.

A tabela hash nessa forma de implementação é um vetor de listas. Para definir o tamanho do vetor, foi definida uma constante TAM_HASH.

A declaração do hash h, no programa principal é realizada pelo comando

```
Lista h[TAM_HASH];
```

O tamanho do hash escolhido vai depender da quantidade de dados no problema a ser tratado. Para esse exemplo didático, vamos usar um tamanho do hash de 10 posições. É um vetor de 10 listas.

5.1. A função hash.

Uma parte importante da estrutura de dados hash é a função hash, que obtém o índice do vetor (posição na tabela de espalhamento) para uma dada chave. A boa distribuição dessa função é importante para o bom desempenho da estrutura. Para nosso exemplo, vamos usar uma função que pega o resto da divisão da chave pelo número de posições do vetor hash. Isso pode ser feito pelo código a seguir:

```
int hash(int chave)
{
   return (chave % TAM_HASH);
}
```

5.2. A criação de um hash vazio (criaHash)

Para criar um hash vazio, é preciso criar uma lista vazia para cada posição do hash. Isso pode ser feito pelo código:

```
void criaHash(Lista *h)
{
   int i;
   for (i=0; i<TAM_HASH; i++)
      h[i]=criaLista();
}
```

5.3. A inserção de um elemento no hash (insereHash)

Para inserir um elemento no hash é preciso descobrir em qual posição do vetor (qual lista) ele deve ser inserido, e chamar a função insereLista para aquela posição do vetor.

```
void insereHash(Lista *h, int valor)
{
   int i;
   i = hash(valor);
   h[i] = insereLista(h[i],valor);
}
```

5.4. A busca de um elemento no hash (buscaHash)

Para buscar um elemento no hash é preciso descobrir em qual posição do vetor (qual lista) ele deve ser estar, e chamar a função buscaLista para aquela posição do vetor. Se o elemento não estiver nessa lista, não estará em nenhuma outra.

```
Lista buscaHash(Lista *h, int valor)
{
   int i;
   i = hash(valor);
   return(buscaLista(h[i],valor));
}
```

Observe que a função busca retorna a posição do elemento, se ele existir no hash, ou NULL caso ele não exista no hash.

5.5. A retirada de um elemento do hash (retiraHash)

Para retirar um elemento do hash é preciso descobrir em qual posição do vetor (qual lista) ele deve estar, e chamar a função retiraLista para aquela posição do vetor.

```
void retiraHash(Lista *h, int valor)
{
   int i;
   i = hash(valor);
   h[i] = retiraLista(h[i],valor);
}
```

5.6. A impressão dos elementos do hash (imprimeHash)

Para imprimir todos os elementos do hash é preciso imprimir o conteúdo de todas as listas que compõem o hash. Isso pode ser feito com o código abaixo.

```
void imprimeHash(Lista *h)
{
   int i;

   printf("\nImprimento as listas do hash h");
   for (i=0; i<TAM_HASH; i++)
   {
      printf("\nh[%d]: ",i);
      imprimeLista(h[i]);
   }
   printf("\n");
}
```