Listas encadeadas[1][2]

Listas encadeadas são estruturas flexíveis que permitem inserção e remoção de elementos sem que os demais elementos da lista precisem ser remanejados. Para tanto, cada elemento armazenado em uma lista é por si só um *container* denominado *nó*, que contém um valor e uma ou mais variáveis (ponteiros, referências ou índices) para indicar seus elementos vizinhos. A seguir detalhamos as mais comuns implementações de listas encadeadas, que variam quando ao número de vizinhos armazenados e ao uso de nós artificiais adicionais, denominados *sentinelas*.

1 Lista encadeada simples

Em uma lista encadeada simples, cada nó armazena exclusivamente um ponteiro para o próximo elemento. Assim, partindo do primeiro nó da lista (tradicionalmente denominado cabeça), pode-se acessar qualquer outro elemento de forma iterativa, como mostrado no algoritmo a seguir. No entanto, o acesso aleatório a diferentes posições em tempo constante como no caso de vetores (baseado em aritmética de ponteiros) não é possível. De fato, em uma lista simplesmente encadeada não é possível identificar o antecessor de um elemento, mesmo quando se tem acesso direto a este nó (retornado de uma busca, por exemplo).

Algoritmo 1 Busca em lista encadeada simples

Input: nó *node*, valor *v*

```
1: if (node = null or node.valor = v) then
2:     return node
3: else
4:     return busca(node.próximo)
5: end if
```

Assim como a busca, tanto a inserção como a remoção podem ser implementadas considerando o acesso iterativo aos nós da lista. A inserção em uma lista requer percorrê-la a partir de sua cabeça para identificar o nó antecessor da posição onde deseja-se inserir o novo elemento. A atualização dos ponteiros segue o algoritmo abaixo:

Algoritmo 2 Inserção em lista encadeada simples

Input: nó *antecessor*, nó *novo*

```
1: novo.próximo := antecessor.próximo
2: antecessor.próximo := novo
```

Analogamente, a remoção de um elemento de uma lista requer identificar o nó antecessor do nó ser removido, mas nesse caso apenas o ponteiro do nó antecessor precisa ser atualizado para manter a integridade da lista. Para simplificar a operação de remoção, usa-se uma sentinela como cabeça da lista, e assim o algoritmo abaixo pode ser aplicado a qualquer nó da mesma:

[1] Roteiro de estudo fornecido na disciplina de Estruturas de Dados Básicas 1 (EDB1) da Universidade Federal do Rio Grande do Norte (UFRN).

[2] Autor: Leonardo Bezerra.

2

Algoritmo 3 Remoção em lista encadeada simples

Input: nó *antecessor*, nó *novo*
1: antecessor.próximo := novo.próximo
2: *apagar*(novo)

Todas as operações apresentadas acima para listas encadeadas simples apresentam complexidade de melhor caso constante e de pior caso linear. No entanto, é possível otimizar o custo médio destas operações desde que a lista seja mantida ordenada. Por fim, algumas estruturas especiais que só permitem inserções e remoções nas extremidades da lista apresentam complexidade assintótica de pior caso constante para estas operações, como veremos quando detalharmos pilhas e filas.

2 Lista duplamente encadeada

Em uma lista duplamente encadeada, cada nó armazena um ponteiro para seu antecessor e um ponteiro para o seu sucessor. Assim, ainda que não seja possível acessar um elemento diretamente sem ter que buscá-lo a partir de uma das extremidades da lista, as demais operações tornam-se mais simples. Mais precisamente, tanto a inserção como a remoção podem ser feitas usando apenas o ponteiro (ou a referência) para um nó, já que a partir deste nó pode-se determinar seu antecessor e seu sucessor. Este ganho de eficiência em termos de tempo, no entanto, custa a listas duplamente encadeadas um uso adicional de memória.

3 Exemplos de aplicação

As aplicações mais importantes de listas envolvem contextos onde a ordem dos elementos deve ser preservada após inserções ou remoções, ou em contextos em que a quantidade de elementos é desconhecida *a priori*:

Ordenação: seja construindo uma lista ordenada incrementalmente, usando o algoritmo de ordenação por inserção ou mesmo algoritmos de ordenação linear como o *bucket sort*;

Implementação de outras estruturas: como veremos mais à frente, tabelas de dispersão e filas são tradicionalmente implementadas usando listas;

4 Exercícios sugeridos

3

1. Escreva um procedimento (ou método) para remover chaves duplicadas de uma lista encadeada não-ordenada.
2. Dada uma lista simplesmente encadeada, escreva um procedimento (ou método) para encontrar o n-ésimo elemento, contado do fim da lista para o começo.
3. Dada uma lista simplesmente encadeada, escreva um procedimento (ou método) para remover um nó, dado apenas acesso àquele nó.
4. Considere dois números implementados como listas encadeadas (cada nó, um dígito). Além disso, os números estão representados da direita para a esquerda (o dígito das unidades é o primeiro nó da lista). Escreva um procedimento (ou método) que some os dois números e retorne o resultado usando a mesma representação dos operandos.
5. Dada uma lista circular[4], escreva um procedimento (ou método) que retorne o nó no início do loop.

[3] Fonte: MCDOWELL, Gayle Laakmann. **Cracking the coding interview**. CarrerCup, 2011.
[4] No contexto desta questão, uma lista circular é uma lista onde há um ciclo.