Algoritmos, Listas Encadeadas

Prof. Ricardo Reis
Universidade Federal do Ceará
Sistemas de Informação
Estruturas de Dados

21 de novembro de 2012

1 Introdução

Listas Encadeadas são listas formadas por elementos estruturais denominados *nódos* que se conectam entre si serialmente e que possuem o valor de uma chave da lista e de uma referência para o nódo seguinte na mesma lista. O primeiro nódo da lista, chamado *raiz* da lista, possui a primeira chave e uma referência para o segundo nódo que por sua vez possui a segunda chave e uma referência para o terceiro nódo e assim sucessivamente. Cada nódo mantém não mais que uma referência. O último nódo da lista possui uma referência para λ (referência nula) que indica final de lista. O número de nódos de uma lista encadeada corresponde ao seu *comprimento*. Teoricamente a capacidade de uma lista encadeada é ilimitada.

O esquema da Figura-1 representa uma lista encadeada. Cada nódo corresponde a um retângulo cuja parte à esquerda contém a chave (x_1, x_2, x_3, $\cdots$) e a parte à direita (em tom escuro) contém a referência ao nódo seguinte. A referência ao nódo raiz é indicada em separado por um quadrado isolado que *não* corresponde a um nódo mas que indica onde está o nódo raiz.

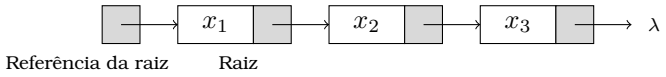

Figura 1: Esquema de lista encadeada

Uma lista encadeada requer em geral dois descritores em sua definição, um para descrição de nódos e outro para descrição da lista propriamente dita. A tabela-1 representa o descritor de um nódo. O atributo $chave$ representa o valor de chave registrado no nódo. O campo $prox$ traz a referência do nódo que o sucede ou λ caso se trate do último nódo.

Tabela 1: Descritor de nódo de uma lista encadeada

Atributo	**Descrição**
$chave$	Chave do nódo
$prox$	Referência para o nódo seguinte

A tabela-2 corresponde ao descritor de uma lista encadeada propriamente dita. A referência $raiz$ é utilizada para apontar para a raiz da lista [1]. O comprimento da lista é registrado por n. Numa lista encadeada vazia $raiz$ e n valem respectivamente λ e zero. Note que *não* são diretamente mantidas todas as referências a nódos da lista, mas apenas ao primeiro nódo e desta forma para acessar uma chave k em específico é necessário realizar uma *caça ao tesouro*, ou seja, partir do primeiro nódo (raiz) e avançar sucessivamente ao nódo seguinte (através de $prox$) até que a chave do nódo visitado seja k.

O operadores mais comuns para listas encadeadas também são descritos na tabela-2. O construtor CRIAR se distingue da versão sequencial por dispensar o valor de capacidade (ilimitada, neste caso). A inserção pode ocorrer em ambas extremidades da estrutura sendo o operador INSERIRNOINICIO especializado em inserções no início da estrutura e INSERIRNOFINAL especializado em inserções no final da estrutura. O destrutor, DESTRUIR, desaloca todos os nódos que compõem a lista encadeada naturalmente destruindo-a.

Tabela 2: Descritor de uma lista encadeada

Atributo	**Descrição**
$raiz$	Referência ao nódo raiz da lista
n	Número de nódos da lista

Operador	**Entrada**	**Ação**
CRIAR		Cria uma nova lista encadeada
INSERIRNOINICIO	$\mathfrak{L}$, x	Insere objeto x na extremidade inicial da lista $\mathfrak{L}$.
INSERIRNOFINAL	$\mathfrak{L}$, x	Insere objeto x na extremidade final da lista $\mathfrak{L}$.
REMOVER	$\mathfrak{L}$, x	Remove objeto x da lista $\mathfrak{L}$, quando x pertence a $\mathfrak{L}$.
BUSCAR	$\mathfrak{L}$, x	Busca objeto x na lista $\mathfrak{L}$
DESTRUIR	$\mathfrak{L}$	Destrói uma lista encadeada

[1] Note que usamos $raiz$ para representar a referência ao primeiro nódo e não o primeiro nódo em si

2 Construtor em Lista Encadeada

Um construtor de lista encadeada é trivial e é implementado pela função CRIAR, Algoritmo-1, que cria uma nova lista encadeada $\mathfrak{L}$ e a retorna como saída. Como a lista deve iniciar vazia então os parâmetros $raiz$ e n são configurados respectivamente para λ e 0.

Algoritmo 1 Construtor de lista encadeada

```
1: Função CRIAR
2:     L.raiz ← λ
3:     L.n ← 0
4:     Retorne L
```

3 Inserção em Lista Encadeada

Os nódos que compõem uma lista encadeada são em geral alocados dinamicamente. Como trataremos com mais de uma possibilidade de inserção então lançaremos mão do operador adicional NOVONODO (Algoritmo-2) que cria dinamicamente um novo nódo, com valor de chave desejado e campo $prox$ nulo (λ). O comando **Alocar** indica que o nódo $\mathcal{N}$ é alocado dinamicamente.

Algoritmo 2 Novo nódo de uma lista encadeada

```
1: Função NOVONODO(x)
2:     Alocar(N)                    ▷ Aloca novo nódo
3:     N.chave ← x
4:     N.prox ← λ
5:     Retorne N
```

A inserção no início de uma lista encadeada é implementada no Algoritmo-3. Consiste da criação de um novo nódo e de sua posterior promoção à nova raiz da estrutura. Esta promoção é feita em duas etapas. Na primeira faz-se o novo nódo apontar (via $prox$) para a raiz da lista (linha-3) e na segunda fa-ze a referência à raiz apontar para o novo nódo (linha-4). A primeira etapa garante a conectividade da estrutura e a segunda define o novo nódo como nova raiz.

Algoritmo 3 Inserção no início de uma lista encadeada

```
1: Função INSERIRNOINICIO(L, x)
2:     N ← NOVONODO(x)              ▷ N é o novo nódo
3:     N.prox ← L.prox
4:     L.raiz ← N
5:     L.n ← L.n + 1                ▷ Aumenta um nódo
```

Note que mesmo em caso de lista vazia o Algoritmo-3 ainda funciona, isso porque a conexão (segunda etapa) se faz à λ o que é coerente haja vista que numa lista que passa a ter um único nódo, tal nóso é naturalmente ao mesmo tempo o primeiro e o último da estrutura. A Figura-2 ilustra como ocorre uma inserção do início de uma lista encadeada.

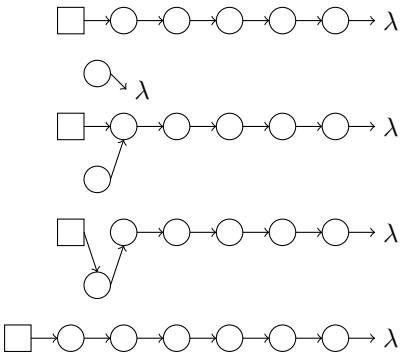

Figura 2: Inserção no início de uma lista encadeada (não ordenada)

A inserção no final de uma lista encadeada é implementada no Algoritmo-4. Nesta situação há duas hipóteses a serem tratadas: lista vazia e não vazia. Em caso de lista vazia ($\mathfrak{L}.n = 0$) invoca-se o operador INSERIRNOINICIO já implementado pois neste caso em paricular a inserção no início é equivalente a no final (linha-3). Em caso de lista não vazia busca-se o último nódo da estrutura ao qual se conecta o novo nódo. A busca é realizada por um laço **Enquanto** (linha-6) que faz uma referência p auxiliar progressivamente saltar para o nódo seguinte (via $prox$, linha-7) enquanto este existir (não nulo). Note que na partida p toma uma cópia da referência raiz. No final do laço, p aponta para o último nódo da lista. Na linha-8 um novo nódo é construído (por chamada a NOVONODO) e conectado a lista através de p (fazer o $prox$ de p apontar para $\mathcal{N}$ equivale a apontar o último nódo de $\mathfrak{L}$ para $\mathcal{N}$).

Algoritmo 4 Inserção no final de uma lista encadeada

```
1: Função INSERIRNOFINAL(L, x)
2:     Se L.n = 0 então
3:         INSERIRNOINICIO(L, x)
4:     senão
5:         p ← L.raiz               ▷ p aponta para raiz
6:         Enquanto p.prox ≠ λ faça
7:             p ← p.prox
8:         p.prox ← NOVONODO(x)
9:         L.n ← L.n + 1            ▷ Lista aumenta um nódo
```

A Figura-3 ilustra a inserção no final de uma lista encadeada.

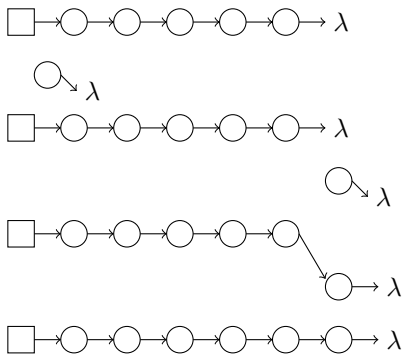

Figura 3: Inserção no final de uma lista encadeada (não ordenada)

4 Busca em Lista Encadeada

É natural imaginar que uma busca em lista encadeada ocorra no sentido de determinar a referência ao nódo que contenha a chave procurada x. Entretanto essa abordagem impossibilita, por exemplo, a remoção do nódo encontrado pela busca haja vista não ser mais conhecido a referência ao nódo anterior necessária para restabelecer a integridade da lista. Isso é o mesmo que dizer que não é possível retroagir em $O(1)$ para nódos anteriores numa lista encadeada pois cada nódo só conhece o nódo adiante. Por essa razão construímos um operador de busca com as seguintes propriedades (considere x a chave procurada),

1. Se a lista estiver vazia ou x estiver no nódo raiz então o operador retornará λ.
2. Se x estiver na lista num nódo diferente do nódo raiz então o operador retornará uma referência para o *nódo pai* daquele que contém x (nódo pai de um dado nódo p é o nódo imediatamente anterior a p).
3. Se x *não* estiver na lista então,
   (a) Caso a lista seja não-ordenada então o operador retornará uma referência para o último nódo da lista.
   (b) Caso a lista seja ordenada então se x for menor que a chave do nódo raiz então o operador retornará λ e do contrário retornará a referência ao último nódo cuja chave não extrapola x (note que se x for maior que todas as chaves existentes na lista essa referência será do último nódo).

Estes comportamentos são implementados pela função BUSCAR do Algoritmo-5. Entre as linhas 2 e 4 resolve-se a condição 1. Um laço principal (linha-5) efetua a caça ao tesouro (o avanço ocorre na linha-9), mas sempre testando o nódo adiante ao visitado ($p.prox \neq \lambda$). y denota a chave do nódo visitado (linha-6). O teste da linha-7 é responsável pela parada do laço e por essa razão contém duas partes lógicas sendo a primeira para controle de listas não ordenadas (**não** $\mathfrak{L}.r$ **e** $y = x$) e a segunda para listas ordenadas ($\mathfrak{L}.r$ **e** $y \geq x$). Note que $y = x$ na primeira parte torna qualquer nódo da lista candidato a conter x ao passo que $y \geq x$ suspende a busca assim que o valor x é extrapolado (de fato, estando a lista ordenada, não há sentido algum buscar adiante da primeira chave que extrapola x). Se este teste obtém sucesso então o laço é suspenso (linha-8) e o fluxo de execução vai para a linlefind.ret onde o último valor de p é retornado. Isso significa dizer que quando x é encontrado num nódo diferente da raiz, BUSCAR retorna a referência ao nódo anterior àquele que contém x.

Algoritmo 5 Busca em lista encadeada

```
1:  Função BUSCAR(ref 𝔏, x)
2:      p ← 𝔏.raiz                          ▷ Primeira hipótese
3:      Se p = λ ou p.chave = x então ▷ Lista vazia ou
        x na raiz
4:          Retorne λ
5:      Enquanto p.prox ≠ λ faça  ▷ Caça ao tesouro
6:          y ← p.prox.chave
7:          Se (não 𝔏.r e y = x) ou (𝔏.r e y ≥ x) então

8:              Pare                          ▷ Para o laço
9:          p ← p.prox                        ▷ Avanço
10:     Retorne p
```

5 Remoção em Lista Encadeada

Algoritmo 6 Remoção em lista encadeada

```
1:  Função REMOVER(ref 𝔏, x)
2:      p ← BUSCAR(𝔏, x)                ▷ Procura da chave
3:      Se p = λ então
4:          Se 𝔏.raiz ≠ λ então
5:              q ← 𝔏.raiz
6:              𝔏.raiz = q.prox
7:          senão
8:              Retorne Falso
9:      senão
10:         q ← p.prox
11:         p.prox ← q.prox
12:     Desalocar q
13:     Retorne Verdadeiro
```

A especialização de BUSCAR permite que outras tarefas que necessitem de busca sejam implementadas mais facilmente. Um exemplo é a remoção de nódos. Veja Algoritmo-6 cuja função REMOVER remove uma chave x de uma lista $\mathfrak{L}$ quando presente. Neste algoritmo a chamada a BUSCAR na linha-2 retorna para p a informação necessária a remoção, ou seja, se p receber λ significará que ou x não está na

lista ou que está no nódo raiz. Do contrário significará que x está na lista e que p contém a referência para o nódo imediatamente anterior (pai).

A remoção da raiz ocorre da linha-5 a linha-6 onde apenas efetua-se uma reconexão como a ilustrada na Figura-4. A remoção de um nódo interno (não-raiz) ocorre de forma similar (veja Figura-5) e ocorre entre a linha-10 e a linha-11. Neste último caso a referência p está diretamente envolvida. Já a referência q, em ambos os casos, é utilizada para manter o nódo removido e ao final ser aplicado na desalocação do mesmo (linha-12). Em linguagens com desalocação automática (como em Java e Python) o uso de q é desnecessário.

A função REMOVER retorna **`Verdadeiro`** quando a remoção obtém sucesso e **`Falso`** do contrário.

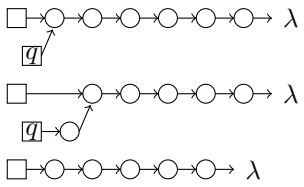

Figura 4: Remoção do nódo raiz

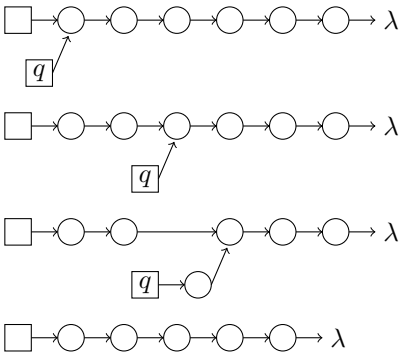

Figura 5: Remoção de nódo interno

6 Destrução de Lista Encadeada

A função DESTRUIR, Algoritmo-7, implementa uma forma simples de eliminar completamente uma lista encadeada. A ideia é extrair e eliminar continuamente o nódo raiz até que a lista esteja vazia. A referência auxiliar q é utilizada em cada etapa para manter a *antiga* raiz e depois desalocá-la (linha-5). Note que $\mathfrak{L}.n$ precisa receber zero manualmente para indicar que mais uma vez a lista tem comprimento nulo.

Algoritmo 7 Destruição de lista encadeada

```
1: Função DESTRUIR(ref 𝔏)
2:     Enquanto 𝔏.raiz ≠ λ faça
3:         q ← 𝔏.raiz                    ▷ referência auxiliar
4:         𝔏.raiz ← 𝔏.raiz.prox           ▷ Avanço raiz
5:         Desalocar(q)                  ▷ Elimina raiz antiga
6:     𝔏.n ← 0
```

7 Lista Encadeada Ordenada

Uma lista encadeada ordenada é aquela cujas chaves se mantêm ordenadas durante todo ciclo de vida da estrutura. Para atingir este comportamento o operador de inserção precisa ser reimplementado e o operador de busca readaptado. Os demais devem se manter da mesma forma. Trataremos estas duas modificações a seguir.

7.1 Inserção em Lista Encadeada Ordenada

A imposição de um critério de ordenação implica desabilitar INSERIRNOINICIO e INSERIRNOFINAL pois novas chaves que surgem para serem inseridas possuem um destino fixo em algum ponto da estrutura e logo estes dois últimos operadores perdem o sentido. Na pratica implementações de listas encadeadas ordenadas e não ordenadas são construídas separadamente.

O Algoritmo-8 implementa a inserção em lista encadeada ordenada. Este algoritmo trata três situações: (i) lista vazia, (ii) lista não vazia sendo a chave a ser inserida menor ou igual a chave da raiz e (iii) lista não vazia sendo a chave de inserção maior que a chave da raiz. Os casos (i) e (ii) são tratados quando o teste da linha-3 obtém sucesso sendo resolvidos exatamente da mesma forma que a inserção no início de uma lista encadeada não ordenada (linha-4 e linha-5). Se este teste falha o caso (iii) é resolvido. Equivale a um processo semelhante à inserção em final de lista encadeada não ordenada, ou seja, uma referência auxiliar p e um laço cooperam no intuito de identificar uma posição de inserção. Entretanto neste caso o teste de **`Enquanto`** também verifica se $x < p.prox.chave$ suspendendo naturalmente numa posição que precederá o nódo a ser inserido ($\mathcal{N}$). Note que, de forma similar a inserção no início de listas encadeadas não ordenadas, duas reconexões devem ser realizadas de forma a introduzir o novo nódo na lista linha-10 e linha-11).

A Figura-6 ilustra a inserção em lista encadeada ordenada.

Algoritmo 8 Inserção em lista encadeada ordenada

```
1: Função InserirEmListaOrdenada(Ł, x)
2:     N ← NovoNodo(x)
3:     Se Ł.n = 0 ou x ≤ Ł.raiz.chave então
4:         N.prox ← Ł.raiz
5:         Ł.raiz ← N
6:     senão
7:         p ← Ł.raiz
8:         Enquanto p.prox ≠ λ e x < p.prox.chave faça
9:             p ← p.prox
10:        N.prox ← p.prox
11:        p.prox ← N
12:    Ł.n ← Ł.n + 1                ▷ Lista aumenta de tamanho
```

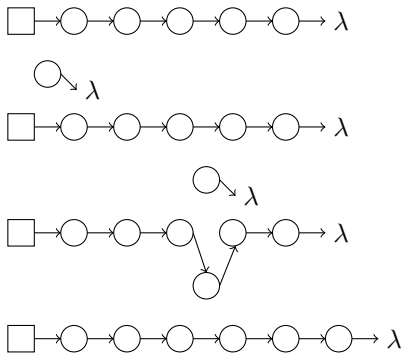

Figura 6: Inserção em lista encadeada ordenada

8 Questões

1. Defina alocação sequencial e alocação por encadeamento. Quais as vantagens e desvantagens de cada uma?
2. Use Buscar para criar uma implementação mais compacta para Inserir.
3. Expandir o operador de remoção de forma a eliminar todas as aparições da chave x.
4. Descrever e implementar listas encadeadas não-ordenadas sem repetição de chaves.
5. Descrever e implementar listas encadeadas ordenadas sem repetição de chaves.
6. Denomina-se *nódo cabeça* a um nódo que fica na posição inicial da lista mas cuja chave não integra a estrutura, ou seja, tais listas, quando vazias, ainda possuem o nódo cabeça. De fato quando criadas já recebem este nódo especial que se mantém durante toda vida útil da lista e não pode ser acessado, editado ou removido pelo usuário. A função de um nódo cabeça é reduzir o tamanho dos códigos dos operadores da lista. Isso ocorre porque a presença deste nódo dispensa alguns testes que alongam a codificação. Identifique as vantagens de implementação que o nódo cabeça trás e em seguida descreva e implemente listas com nódo cabeça.
7. Implementar a substituição de chaves em listas encadeadas. Todas as ocorrências de uma dada chave deverão ser substituídas por outro valor de chave de entrada.
8. Reimplementar a inserção em listas encadeadas sem repetição de chave de forma que cada nó mantenha um contador da quantidade de vezes que sua chave foi tentada ser inclusa (naturalmente esse valor é 1 na primeira inserção e incrementa da unidade nas inserções sem sucesso futuras).
9. Construir função que execute a intercalação de duas listas encadeadas não ordenadas de entrada numa nova lista de saída. Os nós excedentes de uma das listas, se houverem, deverão ser também incluídos na saída na ordem que se sucedem.
10. Construir função que receba duas listas ordenadas e retorne uma nova lista ordenada que é a fusão das duas de entrada. Como fazer isso em $O(n)$? Que muda entre as implementações sequencial e encadeada?
11. Implementar a ordenação de uma lista encadeada utilizando o método bolha.
12. Implementar algoritmo de cópia de lista encadeada.
13. Seja um arquivo de texto formado por nomes de pessoas (apenas o primeiro nome) com possível repetição. Implementar programa em C que receba este arquivo e gere um novo arquivo contendo os mesmos nomes do arquivo de entrada, mas em ordem alfabética (*ordem lexicográfica*) e sem repetição.
14. A frequência de uma palavra num texto é igual ao total de vezes que a palavra aparece neste texto. Construir programa que receba um arquivo de texto e gere um novo arquivo com a frequência de suas palavras. O leiaute deste arquivo de saída é de duas colunas onde a primeira contém as palavras que aparecem pelo menos uma vez no texto (disposta em ordem alfabética) e a segunda coluna o valor da frequencia da palavra correspondente na primeira coluna.
15. Considere que polinômios de ordem quaisquer sejam representados por listas encadeadas de seus coeficientes. Implementar funções para determinar,
    (a) o valor de $P(x_0)$ onde x_0 é um valor de entrada.
    (b) $P(x) + Q(x)$
    (c) $P(x) \times Q(x)$

    onde $P(x)$ e $Q(x)$ denotam polinômios distintos.
16. Seja L a lista encadeada de nós $l_1, l_2, l_3 \ldots, l_n$. Percorrendo L uma única vez construir algoritmos que gerem as seguintes novas listas,
    (a) $l_2, l_3, l_4 \ldots, l_n, l_1$.
    (b) $l_n, l_{n-1}, l_{n-2} \ldots l_2, l_1$.
    (c) $l_1 + l_n, l_2 + l_{n-1} \ldots l_{n/2} + l_{n/2+1}$, onde n é par.
17. Uma associação interessante entre listas sequenciais e encadeadas é a de uma nova estrutura de dados conhecida como *arraylist*. Esta estrutura utiliza encadeamento, mas seus nós são substituídos por listas sequenciais, ou seja, trata-se uma lista encadeada cujos nós são listas sequenciais. Implementar essa estrutura incluindo construção, destruição, inserção e busca, no caso onde a lista é não-ordenada.
18. Que implicações a exigência de ordenação trazem para a implementação de uma *arraylist* (veja questão 17)? Modifique a implementação original para permitir esse comportamento.