

Algoritmos, Listas Sequenciais

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

15 de Maio de 2013

1 Introdução

Uma lista é uma estrutura de dados utilizada na coleção e recuperação eficiente de objetos. Em termos construtivos uma lista é um conjunto de *atributos*, que variam de acordo com o tipo de lista implementada, e *operadores* que em geral são,

- Construção
- Busca
- Inserção
- Remoção

Algumas linguagens de programação fornecem listas como tipos nativos (p.e., Python) enquanto outras inserem este recurso em sua biblioteca padrão (p.e., C++ e Java). Em linguagens como C entretanto deve-se utilizar uma biblioteca não padrão (como a GDSDL¹) ou implementar listas a partir de estruturas nativas mais primitivas.

As implementações mais simples são as de listas *homogêneas*, ou seja, os objetos compartilham o mesmo tipo denominado *tipo base* da lista. Entretanto é possível encontrar implementações *heterogêneas* capazes de manter numa mesma estrutura objetos de tipos diversos não relacionados. Em Python, por exemplo, listas são estruturas nativas e heterogêneas. É importante não confundir listas heterogêneas com *listas genéricas*. Uma lista genérica é uma lista cujo tipo base é definido em tempo de implementação, ou seja, utilizando uma mesma implementação genérica de lista pode-se criar instâncias de listas de números inteiros, de strings, de imagens, de arquivos e etc.

Numa linguagem não orientada a objetos (como C) os atributos de uma lista são representados por elementos primitivos nativos (variáveis escalares, vetores, strings e ponteiros) encapsulados dentro de uma estrutura (*struct*) enquanto os operadores são implementados na forma de funções. Esta estratégia é conhecida como um *tipo de dados abstrato*. Numa linguagem orientada a objetos (como

C++ e Java) uma lista é convenientemente representável por uma classe que encapsula os atributos, em geral privados, e cujos métodos são os operadores. Implementações genéricas de listas (usando *templates* em C++ ou *generics* em Java) são preferenciais pois permitem reuso em contextos diversos.

2 Descrição de Listas Sequenciais

Listas Sequenciais são listas implementadas utilizando-se um vetor como base. A consequência direta deste fato é que listas sequenciais possuem um limite de armazenamento, ou seja, uma vez instanciada uma lista sequencial suportará um número finito de objetos. O limite da quantidade de objetos suportado por uma lista sequencial é denominado de *capacidade* da lista ao passo que a quantidade corrente de objetos armazenados é o *comprimento* da lista. Se o comprimento é zero a lista é dita *vazia*. Se o comprimento se iguala a capacidade a lista é dita *cheia*. *Inserção* é o ato de adição de um novo objeto a uma lista e *remoção* o ato de extração de um objeto existente na lista. Se a lista não estiver cheia diz-se que ela suporta inserção. No caso de lista cheia as tentativas de inserção são *sem sucesso*. Se uma lista está vazia ou não contém um objeto x então a remoção de x é sem sucesso.

Para representar listas em forma algorítmica usaremos um *descriptor*. Um descriptor representa um modelo de associação entre atributos e seus respectivos operadores. O descriptor da tabela-1 será utilizado neste texto para representar listas sequenciais. O vetor M denota o contêiner de armazenamento interno, m denota a capacidade da lista, n representa o comprimento. O operador CRIAR é utilizado para criar uma nova lista, INSERIR para adicionar novos objetos a uma lista dada, REMOVER para extrair objetos existentes e BUSCAR para procurar por um objeto na lista.

¹<http://home.gna.org/gdssl/>

Tabela 1: Descritor de uma lista sequencial

Atributos		
M		Vetor Base
m		Capacidade da lista
n		Comprimento da lista
r		Valor lógico que define se a lista é ou não ordenada

Operador	Entrada	Ação
CRIAR	m, r	Cria uma nova lista sequencial de capacidade máxima m e ordenada quando r é verdadeiro
INSERIR	\mathcal{L}, x	Insere objeto x na lista \mathcal{L}
REMOVER	\mathcal{L}, x	Remove objeto x da lista \mathcal{L} , se presente
BUSCAR	\mathcal{L}, x	Busca objeto x na lista \mathcal{L}

3 Construção de Lista Sequencial

Seja \mathcal{L} uma lista sequencial definida pelo descritor na tabela-1. Então \mathcal{L} possui campos $\mathcal{L}.M$, $\mathcal{L}.m$, $\mathcal{L}.n$ e $\mathcal{L}.r$ que denotam respectivamente vetor base, capacidade, comprimento e auto-ajuste de \mathcal{L} . O construtor de lista do Algoritmo-1 inicializa estes atributos para uma nova lista retornada na saída. O argumento m denota a capacidade da lista e é atribuído diretamente a $\mathcal{L}.m$. O argumento r , que pode ser VERDADEIRO ou FALSO, é atribuído diretamente a $\mathcal{L}.r$ e informa se a lista será ordenada ou não. Uma *lista ordenada* é uma lista cujos objetos se mantêm classificados durante todo tempo de vida da lista. O valor de $\mathcal{L}.n$ é iniciado com zero informando que a lista é inicialmente vazia. A notação $\mathcal{L}.M \leftarrow \text{Alocar}(m)$ é introduzida aqui para representar alocação dinâmica do vetor $\mathcal{L}.M$ com comprimento m .

Algoritmo 1 Construtor de lista sequencial

```

1: Função CRIAR( $m, r$ )
2:    $\mathcal{L}.m \leftarrow m$                                 ▷ Capacidade
3:    $\mathcal{L}.n \leftarrow 0$                                 ▷ Lista inicia vazia
4:    $\mathcal{L}.M \leftarrow \text{Alocar}(m)$                       ▷ Aloca  $m$  células
5:    $\mathcal{L}.r \leftarrow r$                                 ▷ Ordenada?
6:   Retorne  $\mathcal{L}$ 

```

4 Inserção em Lista Sequencial

O Algoritmo-2 implementa a inserção de uma chave x numa lista \mathcal{L} pré-existente. A função INSERIR tenta adicionar x a \mathcal{L} (argumentos de entrada), retorna VERDADEIRO quando a inserção tem sucesso e FALSO do contrário. Note que a lista é repassada como referência pois existe a possibilidade de ser alterada (pela inserção de x).

A inserção se procede como segue. Se a lista estiver cheia (linha-2) o processo se encerra na linha-3 sem sucesso. Do contrário uma nova posição k de inserção é determinada. A primeira hipótese de k é a primeira posição livre em $\mathcal{L}.M$, ou seja, $\mathcal{L}.n + 1$ (linha-5). Caso não seja ordenada (falha do teste da linha-6) a inserção (linha-10) ocorre neste valor de k . Caso $\mathcal{L}.r$ seja VERDADEIRO então $\mathcal{L}.M$ é necessariamente ordenado e é necessário um ajuste de k para manter este critério. Este ajuste é feito pelo laço da linha-7 e representa um deslocamento de todas as chaves de $\mathcal{L}.M$ maiores que x uma posição adiante. Quando o laço encerra o valor de k contém a posição de inserção apropriada. Note que este laço é decrescente (parte da posição $\mathcal{L}.n$ em direção a posição 1) e que os deslocamentos mencionados são nada mais que a cópia das chaves para a posição imediatamente à direita. Note ainda que em ambas situações de inserção o comprimento da lista \mathcal{L} cresce em uma unidade e daí $\mathcal{L}.n$ precisa ser atualizado (linha-11).

Algoritmo 2 Inserção em lista sequencial

```

1: Função INSERIR(ref  $\mathcal{L}, x$ )
2:   Se  $\mathcal{L}.n = \mathcal{L}.m$  então                                ▷ Lista Cheia?
3:     Retorne Falso
4:   senão
5:      $k \leftarrow \mathcal{L}.n + 1$ 
6:     Se  $\mathcal{L}.r$  então                                        ▷ Ordenada?
7:       Enquanto  $k > 1$  e  $\mathcal{L}.M[k - 1] > x$  faça
8:          $\mathcal{L}.M[k] \leftarrow \mathcal{L}.M[k - 1]$ 
9:          $k \leftarrow k - 1$ 
10:     $\mathcal{L}.M[k] \leftarrow x$                                     ▷ Inserção
11:     $\mathcal{L}.n \leftarrow \mathcal{L}.n + 1$                                 ▷ Lista cresce
12:    Retorne Verdadeiro

```

Um ajuste efetuado em uma inserção numa lista ordenada no pior caso tem complexidade $O(n)$ que é um custo substancialmente menor que o de reclassificar a lista pós-inserção (utilizando um algoritmo eficiente seria na ordem de $n \log n$).

5 Busca em Lista Sequencial

Se a lista é não ordenada então o melhor algoritmo de busca é a busca linear. Entretanto se $\mathcal{L}.r$ é VERDADEIRO então $\mathcal{L}.M$ é ordenado e a busca mais eficiente nesta situação é a busca binária. O Algoritmo-3 implementa a busca em lista sequencial levando em conta estas duas possibilidades. A lista alvo é \mathcal{L} e a chave procurada é x , ambos argumentos da função BUSCAR.

Quando $\mathcal{L}.r$ é FALSO (linha-2) ocorre a busca linear. Se a chave buscada é encontrada a função retorna o índice em $\mathcal{L}.M$ da chave encontrada (linha-6). De forma similar se a lista é ordenada ocorre a busca binária (linha-8) e a posição de x em $\mathcal{L}.M$ é retornada na busca com sucesso (linha-13). Na busca sem sucesso BUSCAR retorna zero (linha-19) para indicar que nada foi encontrado (0 é fora da faixa

Algoritmo 3 Busca em lista sequenciais

```

1: Função BUSCAR(ref  $\mathcal{L}$ ,  $x$ )
2:   Se não  $\mathcal{L}.r$  então                                ▷ Busca linear
3:      $k \leftarrow 1$ 
4:     Enquanto  $k \leq \mathcal{L}.n$  faça
5:       Se  $\mathcal{L}.M[k] = x$  então
6:         Retorne  $k$                                 ▷ Busca com sucesso
7:          $k \leftarrow k + 1$ 
8:   senão                                              ▷ Busca binária
9:      $p, q \leftarrow 1, \mathcal{L}.n$ 
10:    Enquanto  $p \leq q$  faça
11:       $k \leftarrow \left\lfloor \frac{p+q}{2} \right\rfloor$ 
12:      Se  $x = \mathcal{L}.M[k]$  então
13:        Retorne  $k$                                 ▷ Busca com sucesso
14:      senão
15:        Se  $x < \mathcal{L}.M[k]$  então
16:           $q \leftarrow k - 1$ 
17:        senão
18:           $p \leftarrow k + 1$ 
19:    Retorne 0                                ▷ Busca sem sucesso

```

1.. $\mathcal{L}.n$). Note que a implementação da busca binária utilizada é iterativa.

6 Remoção em Lista Sequencial

O Algoritmo-4 implementa a remoção em lista sequencial via função REMOVE cujos argumentos de entrada são a lista alvo \mathcal{L} e a chave de remoção x . O algoritmo possui duas partes. Na primeira parte ocorre uma chamada a BUSCAR (linha-2) que determina a posição de x em \mathcal{L} e a armazena em k . Se k recebe um valor não nulo (linha-3) significa que $x \in \mathcal{L}$ e precisa ser removida. A remoção é efetuada pelo laço da linha-5 e se procede como segue. Variando j desde k até $\mathcal{L}.n - 1$ copiam-se todas as chaves em $j + 1$ para a posição j causando um efeito de deslocamento destas chaves à esquerda. Tais deslocamentos tanto proporcionam a sobrescrição de x quanto redução do comprimento de \mathcal{L} (ajustado manualmente na linha-8). O retorno de REMOVE é a posição antiga da chave removida, ou seja, é um valor positivo quando a remoção tem sucesso e zero quando não tem sucesso.

Algoritmo 4 Remoção em busca sequenciais

```

1: Função REMOVE(ref  $\mathcal{L}$ ,  $x$ )
2:    $k \leftarrow \text{BUSCAR}(\mathcal{L}, x)$                                 ▷ Busca por  $x$ 
3:   Se  $k > 0$  então                                ▷ Está presente?
4:      $j \leftarrow k$ 
5:     Enquanto  $j < \mathcal{L}.n$  faça                                ▷ Deslocamento
6:        $\mathcal{L}.M[j] \leftarrow \mathcal{L}.M[j + 1]$ 
7:        $j \leftarrow j + 1$ 
8:      $\mathcal{L}.n \leftarrow \mathcal{L}.n - 1$ 
9:   Retorne  $k$ 

```

7 Outros Aspectos em Listas Sequenciais

Para imprimir as chaves de uma lista sequencial \mathcal{L} podemos usar,

```

1: Para  $k \leftarrow 1$  até  $\mathcal{L}.n$  faça
2:   Escreva  $\mathcal{L}.M[k]$ 

```

Para tanto os atributos $\mathcal{L}.M$ e $\mathcal{L}.n$ precisariam ser visíveis ao usuário. A consequência direta dessa permissividade é a possibilidade de alteração de quaisquer um dos atributos de \mathcal{L} que a rigor só deveriam ser alteradas pelos operadores da lista (isso se agravaria em listas ordenadas pois poderia desconfigurar a ordenação). Uma alternativa seria tornar privado os atributos (se as regras de encapsulamento da linguagem utilizada permitirem) e estender o descritor com operadores auxiliares como os implementados no Algoritmo-5. Estes operadores são descritos a seguir. O operador OBTER retorna a k -ésima chave de um lista sequencial de entrada \mathcal{L} desde que k esteja no intervalo 1 a $\mathcal{L}.n$. Do contrário um erro de acesso é reportado. Os operadores COMPRIMENTO e CAPACIDADE respectivamente devolvem comprimento e capacidade da lista que recebem como entrada.

Algoritmo 5 Operadores complementares de listas sequenciais

```

1: Função OBTER(ref  $\mathcal{L}$ ,  $k$ )                                ▷  $k$ -ésima chave de  $\mathcal{L}$ 
2:   Se  $k > 0$  e  $k \leq \mathcal{L}.n$  então
3:     Retorne  $\mathcal{L}.M[k]$ 
4:   senão
5:     Erro "posição Inválida"

6: Função COMPRIMENTO(ref  $\mathcal{L}$ )
7:   Retorne  $\mathcal{L}.n$ 

8: Função CAPACIDADE(ref  $\mathcal{L}$ )
9:   Retorne  $\mathcal{L}.m$ 

```

Utilizando estes operadores o trecho anterior, que escreve dados de uma lista sequencial, pode ser reescrito como,

```

1: Para  $k \leftarrow 1$  até COMPRIMENTO( $\mathcal{L}$ ) faça
2:   Escreva OBTER( $\mathcal{L}$ ,  $k$ )

```

Em algumas linguagens a operação de desalocação de memória não é automática (como em C e C++) tornando-se necessário realizá-la manualmente quando a memória alocada não for mais necessária. Se uma destas linguagens é utilizada na implementação de listas sequenciais, que alocam dinamicamente $\mathcal{L}.M$, então será necessário um operador de finalização de lista que efetue a desalocação de $\mathcal{L}.M$. A função DESTRUIR no Algoritmo-6 promove a finalização de uma lista sequencial \mathcal{L} passada como entrada. A

notação **Desalocar** $\mathcal{L}.M$ é utilizada para representar desalocação. Os demais atributos recebem zero para desabilitar completamente a lista.

Algoritmo 6 Destrutor de lista sequencial

```

1: Função DESTRUIR(ref  $\mathcal{L}$ )
2:   Desalocar  $\mathcal{L}.M$                                 ▷ Desalocação
3:    $\mathcal{L}.m \leftarrow \mathcal{L}.n \leftarrow 0$ 
  
```

8 Questões

1. Reimplementar a inserção em lista sequencial de forma a não permitir repetição de chaves.
2. Implementar operador REMOVE TODO que receba uma lista sequencial \mathcal{L} e remova todas as aparições de um chave dada x .
3. Implemente um operador de ordenação para uma lista sequencial não ordenada.
4. Listas sequenciais com *sentinela* são listas que possuem uma célula a mais no vetor base ($m+1$) mas que não possui função de armazenamento, ou seja, para o usuário a lista possui ainda a mesma capacidade (m). A função desta célula auxiliar é permitir que a busca em lista sequencial não ordenada (Algoritmo-3, linha-2) efetue ao invés de dois testes (um em **while** e outro em **if**) apenas um. A estratégia é descrita a seguir. Armazena-se uma cópia da chave procurada, x , na posição $n+1$ (sentinela) que sucede a última posição da lista e estará sempre disponível mesmo quando a lista estiver cheia pois se $m = n$ ainda haverá a posição $m+1$ (célula auxiliar). Em seguida efetua-se uma busca *somente* com **while** que neste caso sempre encontrará x . Se neste processo a posição p encontrada for tal que $p \leq \mathcal{L}.n$ então houve busca com sucesso. Entretanto se $p = \mathcal{L}.n + 1$ significa que a busca não teve sucesso. Implemente a busca linear com sentinela em lista não ordenada. Qual a melhoria em ordem de complexidade?
5. Construa versões recursivas para os operadores de inserção, busca e remoção em lista sequencial.
6. Reimplementar a inserção em lista sequencial de forma que seja informada a extremidade de inserção (início ou fim da lista). Note que isso não terá efeito sobre listas ordenadas. Quais as complexidades associadas?
7. Utilizando a implementação genérica de lista apresentada construa uma lista cujos objetos são registros com nome, idade e salário de uma pessoa. Forneça em sua implementação recursos que permitam que o critério de ordenação varie entre *por nome*, *por idade* ou *por salário*.