Algoritmos, Listas Sequenciais

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

21 de Novembro de 2012

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 GDSL [1]) 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 *descritor*. Um descritor representa um modelo de associação entre atributos e seus respectivos operadores. O descritor 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/gdsl/

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	$\mathfrak{L}$, x	Insere objeto x na lista $\mathfrak{L}$
REMOVER	$\mathfrak{L}$, x	Remove objeto x da lista $\mathfrak{L}$, se presente
BUSCAR	$\mathfrak{L}$, x	Busca objeto x na lista $\mathfrak{L}$

3 Construção de Lista Sequencial

Seja $\mathfrak{L}$ uma lista sequencial definida pelo descritor na tabela-1. Então $\mathfrak{L}$ possui campos $\mathfrak{L}.M$, $\mathfrak{L}.m$, $\mathfrak{L}.n$ e $\mathfrak{L}.r$ que denotam respectivamente vetor base, capacidade, comprimento e auto-ajuste de $\mathfrak{L}$. 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 $\mathfrak{L}.m$. O argumento r, que pode ser VERDADEIRO ou FALSO, é atribuído diretamente a $\mathfrak{L}.r$ 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 $\mathfrak{L}.n$ é iniciado com zero informando que a lista é inicialmente vazia. A notação $\mathfrak{L}.M \leftarrow$ **Alocar**(m) é introduzida aqui para representar alocação dinâmica do vetor $\mathfrak{L}.M$ com comprimento m.

Algoritmo 1 Construtor de lista sequencial

```
1: Função CRIAR(m, r)
2:     L.m ← m                          ▷ Capacidade
3:     L.n ← 0                          ▷ Lista inicia vazia
4:     L.M ← Alocar(m)                  ▷ Aloca m células
5:     L.r ← r                          ▷ Ordenada?
6:     Retorne L
```

4 Inserção em Lista Sequencial

O Algoritmo-2 implementa a inserção de uma chave x numa lista $\mathfrak{L}$ pré-existente. A função INSERIR tenta adicionar x a $\mathfrak{L}$ (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 $\mathfrak{L}.M$, ou seja, $\mathfrak{L}.n + 1$ (linha-5). Caso não seja ordenada (falha do teste da linha-6) a inserção (linha-10) ocorre neste valor de k. Caso $ls.r$ seja VERDADEIRO então $\mathfrak{L}.M$ 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 $\mathfrak{L}.M$ 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 $\mathfrak{L}.n$ 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 $\mathfrak{L}$ cresce em uma unidade e daí $\mathfrak{L}.n$ precisa ser atualizado (linha-11).

Algoritmo 2 Inserção em lista sequencial

```
1: Função INSERIR(ref L, x)
2:     Se L.n = L.m então                ▷ Lista Cheia?
3:         Retorne Falso
4:     senão
5:         k ← L.n + 1
6:         Se L.r então                  ▷ Ordenada?
7:             Enquanto k > 1 e L.M[k − 1] > x faça
8:                 L.M[k] ← L.M[k − 1]
9:                 k ← k − 1
10:        L.M[k] ← x                    ▷ Inserção
11:        L.n ← 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 $\mathfrak{L}.r$ é VERDADEIRO então $\mathfrak{L}.M$ 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 $\mathfrak{L}$ e a chave procurada é x, ambos argumentos da função BUSCAR.

Quando $\mathfrak{L}.r$ é FALSO (linha-2) ocorre a busca linear. Se a chave buscada é encontrada a função retorna o índice em $\mathfrak{L}.M$ 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 $\mathfrak{L}.M$ 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 Ł, x)
2:     Se não Ł.r então                          ▷ Busca linear
3:         k ← 1
4:         Enquanto k ≤ Ł.n faça
5:             Se Ł.M[k] = x então
6:                 Retorne k                     ▷ Busca com sucesso
7:             k ← k + 1
8:     senão                                     ▷ Busca binária
9:         p, q ← 1, Ł.n
10:        Enquanto p ≤ q faça
11:            k ← ⌊(p + q)/2⌋
12:            Se x = Ł.M[k] então
13:                Retorne k                     ▷ Busca com sucesso
14:            senão
15:                Se x < Ł.M[k] então
16:                    q ← k − 1
17:                senão
18:                    p ← k + 1
19:    Retorne 0                                 ▷ Busca sem sucesso
```

$1..\text{Ł}.n$). 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 REMOVER cujos argumentos de entrada são a lista alvo Ł 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 Ł e a armazena em k. Se k recebe um valor não nulo (linha-3) significa que $x \in$ Ł e precisa ser removida. A remoção é efetuada pelo laço da linha-5 e se procede como segue. Variando j desde k até $\text{Ł}.n - 1$ 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 Ł (ajustado manualmente na linha-8). O retorno de REMOVER 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 REMOVER(ref Ł, x)
2:     k ← BUSCAR(Ł, x)                          ▷ Busca por x
3:     Se k > 0 então                            ▷ Está presente?
4:         j ← k
5:         Enquanto j < Ł.n faça                 ▷ Deslocamento
6:             Ł.M[j] ← Ł.M[j + 1]
7:             j ← j + 1
8:         Ł.n ← Ł.n − 1
9:     Retorne k
```

7 Outros Aspectos em Listas Sequenciais

Para imprimir as chaves de uma lista sequencial Ł podemos usar,

```
1: Para k ← 1 até Ł.n faça
2:     Escreva Ł.M[k]
```

Para tanto os atributos $\text{Ł}.M$ e $\text{Ł}.n$ precisariam ser visíveis ao usuário. A consequência direta dessa permissividade é a possibilidade de alteração de quaisquer um dos atributos de Ł 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 Ł desde que k esteja no intervalo 1 a $\text{Ł}.n$. 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 Ł, k)        ▷ k-ésima chave de Ł
2:     Se k > 0 e k ≤ Ł, n então
3:         Retorne Ł.M[k]
4:     senão
5:         Erro "posição Inválida"

6: Função COMPRIMENTO(ref Ł)
7:     Retorne Ł.n

8: Função CAPACIDADE(ref Ł)
9:     Retorne Ł.m
```

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

```
1: Para k ← 1 até COMPRIMENTO(Ł) faça
2:     Escreva OBTER(Ł, 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 $\text{Ł}.M$, então será necessário um operador de finalização de lista que efetue a desalocação de $\text{Ł}.M$. A função DESTRUIR no Algoritmo-6 promove a finalização de uma lista sequencial Ł passada como entrada. A

notação **Desalocar** $\mathfrak{L}.M$ 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 L)
2:     Desalocar L.M                    ▷ Desalocação
3:     L.m ← L.n ← 0
```

8 Questões

1. Reimplementar a inserção em lista sequencial de forma a não permitir repetição de chaves.
2. Implementar operador REMOVETUDO que receba uma lista sequencial $\mathfrak{L}$ 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 $p \leq \mathfrak{L}.n$ então houve busca com sucesso. Entretanto se $p = \mathfrak{L}.n + 1$ 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*.