N_EST DAD_A2 – Texto de Apoio

Site: EAD Mackenzie
Tema: ESTRUTURA DE DADOS {TURMA 03A} 2023/1
Livro: N_EST DAD_A2 – Texto de Apoio

Impresso por: FELIPE BALDIM GUERRA .
Data: sexta, 3 mar 2023, 07:05

Índice

CONCEITUANDO LISTAS LINEARES

A característica fundamental de uma lista linear é o sentido de ordem unidimensional dos elementos que a compõem. Uma ordem que permite dizer com precisão onde a estrutura começa e onde termina. As listas podem crescer ou diminuir de tamanho durante a execução de um programa, de acordo com a necessidade. São exemplos de aplicações com listas:

- Notas de alunos.
- Cadastro de funcionários de uma empresa.
- Dias da semana.
- Lista de voos que irão decolar.
- Lista de arquivos de um diretório.
- Cartas de um baralho etc.

As listas lineares são divididas em dois grupos:

- **Listas Lineares Gerais (sem disciplina de acesso)** – são aquelas que não apresentam nenhuma restrição de acesso, podendo sofrer inserções e remoções em qualquer lugar, inclusive no meio da lista.
- **Listas Particulares (com disciplina de acesso)** – as inserções e remoções de elementos têm uma regra própria.

Nesta aula, estudaremos as Listas Lineares Gerais. A maneira mais simples de se armazenar uma lista como essa é por meio de uso de vetores. A representação por vetor explora a sequencialidade da memória de tal forma que os elementos de uma lista sejam armazenados em endereços contíguos, ou igualmente distanciados um do outro.

VETORES

Um **vetor** (lista-arranjo ou lista linear) de **tamanho N** é uma **coleção** V de N elementos, armazenados de forma **linear**, de tal maneira que se possa acessar seus elementos por meio de um **índice (posição)**. Neste tipo de estrutura, a relação de ordem entre os dados é preservada e tem as seguintes características:

- V[0] é o primeiro elemento da lista.
- V[n-1] é o último elemento da lista.
- Se 1 < k < N-1, L[k] é precedido por L[k-1] e seguido por L[k+1].
- Se N = 0, dizemos que a lista é vazia.
- A inserção de um novo item no meio da lista requer um deslocamento de todos os itens localizados após o ponto de inserção.
- Retirar um item do início da lista requer um deslocamento de itens para preencher o espaço deixado vazio.
- A implementação de vetores é simples para percorrer e acessar e é mais complexa para inserir e remover.

No exemplo abaixo, temos um vetor de tamanho N, com indexação feita a partir do índice 0.

Quais são as operações que podem ser executadas sobre o TAD vetor?

Operação	Significado	Restrição
empty (N)	Cria um vetor vazio com **capacidade** para N elementos.	N ≥ 1
isEmpty()	Verifica se o vetor está vazio, isto é, se não contém elementos.	
size()	Retorna o **tamanho** do vetor, isto é, quantos elementos estão inseridos no vetor.	
get(i)	Retorna o elemento que está armazenado no índice i (posição i).	0 ≤ i ≤ N-1
set(i,E)	Substitui o elemento que está armazenado na posição i por E.	0 ≤ i ≤ N-1
add (i,E)	Insere um novo elemento E no índice i.	0 ≤ i ≤ N-1
remove (i)	Remove o elemento de índice i.	0 ≤ i ≤ N-1
search(E)	Busca o elemento E no vetor e retorna a posição onde foi encontrado ou -1, se não existir no vetor.	

Acompanhe a sequência abaixo, que simula uma Lista Linear com nomes de pessoas:

Operação	Retorno	Status da Lista
empty(5)		[ ] [ ] [ ] [ ] [ ]
add(0,”Pedro”)		**Pedro** [ ] [ ] [ ] [ ]
add (1,”Ana”)		**Pedro** **Ana** [ ] [ ] [ ]
size()	2	
add(5,”Maria”)	Erro!!	
add(3,”Luiza”)		**Pedro** **Ana** **Luiza** [ ] [ ]
set(1,”Carlos”)		**Pedro** **Carlos** **Luiza** [ ] [ ]
add (1,”Sheila”)		**Pedro** [ ] **Carlos** **Luiza** [ ] **Pedro** **Sheila** **Carlos** **Luiza** [ ]
get(3)	Luiza	
add(0,”Milena”)		[ ] **Pedro** **Sheila** **Carlos** **Luiza** **Milena** **Pedro** **Sheila** **Carlos** **Luiza**

add(0,”Paulo”)	Erro!!	
remove(2)		
search(“Carlos”)	2	
search(“luiza”)	-1	
isEmpty()	false	

Há duas estratégias diferentes para se implementar uma Lista Linear:

- **arranjos simples:** nesta estratégia, utiliza-se um vetor de capacidade fixa, que não pode ser alterada durante o ciclo de vida do vetor. Se o tamanho do vetor ultrapassar a capacidade, gera-se uma exceção (erro);
- **arranjos extensíveis:** nesta estratégia, define-se uma capacidade inicial para o vetor e, quando o tamanho deste vetor ultrapassar a capacidade, aloca-se automaticamente mais espaço.

IMPLEMENTAÇÃO UTILIZANDO ARRANJOS SIMPLES – OPERAÇÕES BÁSICAS

A implementação de vetores via arranjos simples utiliza um vetor de qualquer tipo (primitivos ou objetos) e, além da verificação dos índices das operações, verificamos se o tamanho do vetor não ultrapassa sua capacidade nas operações de inserção de elementos. Neste exemplo, estamos criando uma lista linear para armazenar nomes de pessoas:

```
public class Vetor {
    String[] vetor;
    int tamanho;
    int qtde;

    Vetor(int tamanho) {...}

    public boolean isEmpty() {...}

    public int size() {...}

    public String get(int pos) throws Exception {...}

    public void set(int pos, String n) throws Exception {...}

    public void add(int pos, String n) throws Exception {...}

    public void remove(int pos) throws Exception {...}

    public int search(String n) {...}

    public String[] getArray() {...}
}
```

Nós implementaremos agora cada uma dessas operações. Para isso, vamos criar um projeto no NetBeans com o nome ProjLista e adicionar uma classe chamada Vetor com o código acima.

É muito importante que você tente entender cada uma das operações abaixo. Para cada caso, leia atentamente minhas considerações.

Analisaremos, inicialmente, os atributos da classe Vetor:

```
String[] vetor;
int tamanho;
int qtde;
```

Você sabe o objetivo de cada um deles no programa?

- A variável `vetor` é a lista propriamente dita. É nessa variável que armazenaremos o conteúdo de nossa lista; neste caso, será do tipo String, pois armazenaremos nomes de pessoas. Repare que é nesta declaração que você indica o tipo de conteúdo que deseja para sua lista. Por exemplo, se você quiser uma lista de números inteiros, declare `int[] vetor;` ou ainda, se quiser uma lista de objetos do tipo Aluno, declare `Aluno[] vetor;`
- A variável `tamanho` indica a capacidade máxima do vetor, ou seja, quantas posições, no máximo, o vetor suporta (a lista suporta).
- A variável `qtde` indica quantos elementos a lista tem naquele momento. Isso significa que podemos ter uma lista de tamanho = 50 e qtde = 10. Nesse caso, a lista permite até 50 elementos, porém, neste momento, apenas 10 posições estão ocupadas.

Analisaremos agora cada um dos métodos:

MÉTODO CONSTRUTOR

```
Vetor(int tamanho) {
    vetor = new String[tamanho];
    this.tamanho = tamanho;
    this.qtde = 0;
}
```

Este é o método construtor. Assim que Vetor for instanciado, o construtor receberá como parâmetro uma variável que indicará a capacidade máxima da lista. Repare que o vetor é efetivamente criado no neste método e, na sequência, o tamanho da lista é atualizado e a quantidade é "zerada". Isso porque, nesse momento, apesar de termos diversas posições disponíveis, não temos nenhum elemento na lista.

MÉTODO ISEMPTY() – VERIFICA SE A LISTA ESTÁ VAZIA

```
public boolean isEmpty() {
    if (qtde == 0) {
        return true;
    } else {
        return false;
    }
}
```

O objetivo deste método é sinalizar quando a lista está vazia. O atributo que nos indica isso é a `qtde`, pois ela sempre nos indica quantos elementos temos na lista em determinado momento. A sinalização é feita por meio de uma variável booleana que retornará `true` (lista vazia) ou `false` (lista não vazia – atenção: isso não significa que a lista está cheia!).

MÉTODO SIZE() – RETORNA A QUANTIDADE DE ELEMENTOS NA LISTA

```
public int size() {
    return qtde;
}
```

Este método é bastante simples! Ele serve apenas para retornar a quantidade de elementos que existem na lista em determinado momento.

MÉTODO GET() – RETORNA O CONTEÚDO DE UMA DETERMINADA POSIÇÃO DA LISTA

```
public String get(int pos) throws Exception {
    if (isEmpty()) {
        throw new Exception("Lista vazia - não há elemento para recuperar!");
    }

    if (pos < 0 || pos >= qtde) {
        throw new Exception("Índice da Lista é inválido!");
    }
    return (vetor[pos]);
}
```

O objetivo deste método é retornar o conteúdo de uma determinada posição da lista. Essa posição é recebida como parâmetro pela função. Repare que existem algumas condições que invalidam a execução do método:

- Tentar retornar o conteúdo de uma determinada posição se a lista estiver vazia.
- Tentar retornar o conteúdo de uma posição inexistente ou negativa.

Nesses casos, é lançada uma exceção e o código é interrompido. Caso a posição seja válida, ou seja, existe conteúdo na posição indicada, o valor é retornado pela função. Você consegue entender cada um dos elementos da assinatura abaixo?

```
public String get(int pos) throws Exception
```

MÉTODO SET() – ALTERA O CONTEÚDO DE UMA DETERMINADA POSIÇÃO DA LISTA

```
public void set(int pos, String n) throws Exception {
    if (isEmpty()) {
        throw new Exception("Lista vazia - não há elemento para alterar!");
    }

    if (pos < 0 || pos >= qtde) {
        throw new Exception("Índice da Lista é inválido!");
    }
    vetor[pos] = n;
}
```

O objetivo deste método é alterar o conteúdo de uma determinada posição da lista, que já existe. Essa posição é recebida como parâmetro pela função, juntamente com o novo conteúdo. Repare que existem algumas condições que invalidam a execução do método:

- Tentar alterar o conteúdo de uma determinada posição se a lista estiver vazia.
- Tentar alterar o conteúdo de uma posição inexistente ou negativa.

Nesses casos, é lançada uma exceção e o código é interrompido. Caso a posição seja válida, ou seja, existe conteúdo na posição indicada, o valor atual é substituído pelo valor recebido como parâmetro. Você consegue entender cada um dos elementos da assinatura abaixo?

```
public void set(int pos, String n) throws Exception
```

INSERÇÃO, EXCLUSÃO E BUSCA DE ELEMENTOS NA LISTA

MÉTODO ADD() – ADICIONA UM NOVO ELEMENTO NA LISTA

A inserção de um novo item na posição “pos” da lista requer um deslocamento “para direita” de todos os itens localizados a partir do ponto de inserção.

Exemplo: inserir o valor 7 na posição 3

Milena	Pedro	Carlos	Luiza	Ana	Jorge	Oscar							
0	1	2	3	4	5	6	7	8	9	10	11	12	13

A posição 3 do vetor está ocupada pelo nome Luiza. Se eu quero inserir o nome “Miguel” na posição 3, eu preciso, em primeiro lugar, liberar o espaço da posição 3. Para isso, deslocaremos todos os nomes localizados a partir do ponto de inserção uma posição para frente. Lembre-se de que uma das características do vetor é que sempre é preservada a relação de ordem de seus elementos. Veja abaixo como ficará o vetor após o deslocamento dos elementos:

Milena	Pedro	Carlos		Luiza	Ana	Jorge	Oscar						
0	1	2	3	4	5	6	7	8	9	10	11	12	13

E, agora, após a inserção do nome “Miguel”:

Milena	Pedro	Carlos	Miguel	Luiza	Ana	Jorge	Oscar						
0	1	2	3	4	5	6	7	8	9	10	11	12	13

```java
public void add(int pos, String n) throws Exception {
    if (qtde == tamanho) {
        throw new Exception("A Lista está cheia! Impossível inserir!");
    }
    if (pos < 0 || pos > qtde) {
        throw new Exception("Índice da Lista é inválido!");
    }
    // abre espaço no vetor
    for (int j = qtde - 1; j >= pos; j--) {
        vetor[j + 1] = vetor[j];
    }
    vetor[pos] = n;
    qtde++;
}
```

Repare que existem algumas condições que invalidam a execução do método:

- Tentar adicionar um elemento quando a lista já estiver totalmente preenchida.
- Tentar adicionar um elemento em uma posição negativa ou não contígua (por exemplo, a lista tem tamanho=50, a qtde=10 e você tenta inserir um elemento na posição 34).

Nesses casos, é lançada uma exceção e o código é interrompido. Caso a posição seja válida, ou seja, a posição está correta, é então liberado o espaço necessário para inclusão do novo elemento (deslocamento “para a direita” e, na sequência, o elemento é posicionado conforme indicado pela variável pos. Repare que a variável qtde é sempre atualizada nesse método. Você consegue entender cada um dos elementos da assinatura abaixo?

```java
public void add(int pos, String n) throws Exception
```

MÉTODO REMOVE() – REMOVE UM ELEMENTO DA LISTA

Para a operação de remoção de elemento, utilizaremos uma estratégia contrária à da inserção: deslocaremos os elementos do vetor uma casa para a esquerda, cobrindo a posição removida. Acompanhe a sequência abaixo:

Milena	Pedro	Carlos	Miguel	Luiza	Ana	Jorge	Oscar						

0	1	2	3	4	5	6	7	8	9	10	11	12	13

Removeremos do vetor a posição 2, cujo conteúdo é “Carlos”:

Milena	Pedro	Miguel	Luiza	Ana	Jorge	Oscar							
0	1	2	3	4	5	6	7	8	9	10	11	12	13

```
public void remove(int pos) throws Exception {
    if (isEmpty()) {
        throw new Exception("Lista vazia - não há elemento para remover!");
    }
    if (pos < 0 || pos >= qtde) {
        throw new Exception("Índice da Lista é inválido!");
    }
    // movimenta os elementos para a esquerda p/ cobrir posição vaga
    for (int j = pos; j <= qtde - 2; j++) {
        vetor[j] = vetor[j + 1];
    }
    qtde--;
}
```

Repare que existem algumas condições que invalidam a execução do método:

- Tentar remover um elemento quando a lista estiver vazia.
- Tentar remover um elemento em uma posição negativa ou não contígua (por exemplo, a lista tem tamanho=50, a qtde=10 e você tenta remover um elemento na posição 15).

Nesses casos, é lançada uma exceção e o código é interrompido. Caso a posição seja válida, ou seja, a posição está correta, os elementos posicionados a partir do ponto de remoção são deslocados “para esquerda”. Repare que a variável qtde é sempre atualizada nesse método. Você consegue entender cada um dos elementos da assinatura abaixo?

```
public void remove(int pos) throws Exception
```

MÉTODO SEARCH() – BUSCA POR UM ELEMENTO NA LISTA

Esta operação é a busca linear que você já conhece. A partir de um valor enviado como parâmetro, é feita a busca no vetor. Se o elemento existir, a função retorna a posição onde o elemento está armazenado. Se o elemento não existir, a função sinaliza, retornando o valor -1.

```
public int search(String n) {
    for (int i = 0; i < qtde; i++) {
        if (vetor[i].equals(n)) {
            return i;
        }
    }
    return -1;
}
```

ANALISANDO A EFICIÊNCIA DESSES ALGORITMOS

Vamos determinar agora a eficiência desses métodos, a partir da notação Big O?

isEmpty()	0(1)	Tempo Constante – o tempo gasto será sempre o mesmo, independentemente do tamanho da lista.
size()	0(1)	Tempo Constante – o tempo gasto será sempre o mesmo, independentemente do tamanho da lista.
get()	0(1)	Tempo Constante – o tempo gasto será sempre o mesmo, independentemente do tamanho da lista.
set()	0(1)	Tempo Constante – o tempo gasto será sempre o mesmo, independentemente do tamanho da lista.
add()	O(n)	Tempo Linear – Quando n é muito grande (n ➔ ∞) e pos é muito pequeno (0, por exemplo, que é o pior caso de execução deste algoritmo), a quantidade de movimentações necessárias é n-pos+1 ≅ n. Assim, determinamos que o custo desta operação é O(n).
remove()	O(n)	Tempo Linear – Quando n é muito grande (n ➔ ∞) e pos é muito pequeno (0, por exemplo, que é o pior caso de execução deste algoritmo), a quantidade de movimentações necessárias é n-pos+1 ≅ n. Assim, determinamos que o custo desta operação é O(n).

search()	O(n)	Tempo Linear – No melhor caso, a busca é O(1) e, no pior caso, é O(n). Assim, o caso médio é também O(n).

IMPLEMENTAÇÃO UTILIZANDO ARRANJOS EXTENSÍVEIS

Nesta estratégia, define-se uma capacidade inicial para o vetor e, quando o tamanho deste vetor ultrapassar essa capacidade, aloca-se automaticamente mais espaço (dinâmico).

Fazer alocação dinâmica significa criar espaços de memória em tempo de execução. Nas soluções propostas até aqui para os vetores, usamos a alocação estática. Ou seja, quando o vetor criado alcançava sua capacidade máxima, não havia como inserir novos elementos na lista até que pelo menos um elemento fosse removido.

Nesta estratégia, quando o vetor atingir sua capacidade máxima, um novo vetor será criado automaticamente, com um tamanho maior e, em seguida, os dados do vetor que atingiu a capacidade máxima são transferidos para o novo vetor, um a um. Suponha que o vetor A tem capacidade para cinco elementos e ele tem o seguinte conteúdo:

VETOR	Milena	Pedro	Carlos	Miguel	Luiza

Neste momento, com sua capacidade esgotada, cria-se um novo vetor B (com o dobro do tamanho do vetor A, por exemplo) e copiam-se todos os elementos para o novo vetor:

B	Milena	Pedro	Carlos	Miguel	Luiza					

Assim, o dobro do espaço foi reservado e, caso o novo vetor B seja totalmente preenchido, um novo vetor poderá ser criado com 20 elementos e assim por diante. Veja o código referente a esse processo:

```java
public void duplica() {
    if (size() == tamanho) {
        String[] B = new String[vetor.length * 2];
        for (int i = 0; i < vetor.length; i++) {
            B[i] = vetor[i];
        }
        vetor = B;
    }
}
```

O método `duplica()` verifica se o vetor está totalmente preenchido e cria um novo vetor (B) com o dobro do tamanho. Após a criação do novo vetor, todos os elementos da lista são transferidos para B e, ao término desta operação, a lista passa a apontar para o novo vetor (vetor = B). Esta última instrução é importantíssima para garantir a correta continuidade do programa.

Embora este processo de realocação possa parecer lento, o resultado a seguir nos mostra que a complexidade de inserção continua sendo O(n).