UNIRIO UNIVERSIDADE FEDERAL DO ESTADO DO RIO DE JANEIRO | CCET CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA | EIA ESCOLA DE INFORMÁTICA APLICADA | BACHARELADO EM SISTEMAS DE INFORMAÇÃO

Disciplina: Estruturas de Dados
2024.1 — Lista de exercícios 1

Parte dos exercícios extraída ou adaptada da bibliografia da disciplina. Implemente as soluções em um projeto Java dedicado aos exercícios. Quando o exercício se referir a outras classes e/ou funções, importe os pacotes necessários no código. Mesmo quando não for fornecida uma função `Main`, escreva a `Main` que teste as funcionalidades que você implementou, procurando os casos limite. Esta prática ajuda a garantir que o código funciona como você (ou o exercício) espera.

Tipos abstratos de dados

Questão 1

Considere o tipo abstrato de dados `Contador`, contendo as seguintes funcionalidades:

interface Contador		
	`Contador( String id )`	cria um novo `Contador` com identificador `id`
`void`	`incrementa()`	incrementa o valor do `Contador`
`int`	`valor()`	quantidade de incrementos já feitos no `Contador`
`String`	`toString()`	representação em string do `Contador`

(a) Crie uma interface Java que represente este tipo abstrato de dados;

(b) Crie uma classe que implemente esta interface e execute suas funcionalidades;

(c) Utilizando esta classe, crie uma função `moedas`. Esta função deve receber um valor `int` como entrada, lançar uma quantidade de moedas[1] igual ao valor recebido, e imprimir em console a quantidade de caras e coroas.

Questão 2

Considere o tipo abstrato de dados `Lista`, contendo as seguintes funcionalidades:

interface Lista	
`Lista( int cap )`	cria uma nova `Lista` de inteiros com capacidade para `cap` elementos. As posições de uma lista com capacidade `cap` são numeradas 0, 1, ..., `cap-1`
`void set( int val, int pos )`	armazena o valor `val` na `pos`-ésima posição da `Lista`. Caso a `Lista` tenha até `pos` elementos, uma `ArrayIndexOutOfBoundsException` deve ser gerada
`int get( int pos )`	retorna o valor armazenado na `pos`-ésima posição da `Lista`. Caso a `Lista` tenha até `pos` elementos, uma `ArrayIndexOutOfBoundsException` deve ser gerada

[1] Utilize a classe `java.util.Random` para criar um gerador de números aleatórios, e o método `nextBoolean()` para gerar uma variável Booleana aleatória. Um valor `true` (resp. `false`) representa uma moeda lançada que caiu cara (resp. coroa).

(a) Crie uma interface Java que represente este tipo abstrato de dados.

(b) Crie três classes que implementem esta interface e executem suas funcionalidades, com as seguintes propriedades.

ATENÇÃO: no desenvolvimento destas três classes, você **não pode utilizar estruturas de dados auxiliares** da biblioteca padrão do Java, incluindo `ArrayList`, ou de outras bibliotecas.

1. A classe `ListaSimples` deve armazenar os números da lista em um array de `cap` posições;
2. A classe `ListaExpansivel` deve armazenar os números da mesma forma que a classe `ListaSimples`, mas possuir um método `expandir( int novaCap )`, que expande a capacidade atual da lista para `novaCap`[2];
3. A classe `ListaDinamica` deve possuir as mesmas funcionalidades gerais da classe `ListaExpansível`, mas a implementação do método `set` deve expandir a capacidade do vetor se necessário.

Questão 3

Uma interface importante da biblioteca padrão do Java é a interface `Comparable<T>`. Ela especifica a interface a ser implementada por todas as classes que podem ter seus objetos *ordenados*, ou seja, que possuam uma regra consistente[3] que diga, entre dois objetos, quem vem antes de quem.

A ideia é simples: o método `Comparable<T>::compareTo(T o)` deve comparar dois objetos (`this` e `o`) e retornar um número inteiro que indica quem vem antes: se `this` vier antes de `o`, o retorno deve ser um número negativo (e.g. -1); se `o` vier antes de `this`, o retorno deve ser um número negativo (e.g. +1); e se `this` e `o` forem equivalentes (ou seja, tanto faz quem vem antes), o retorno deve ser 0.

(a) Crie uma classe `Racional`, que represente números racionais e implemente[4] a interface `Comparable<Racional>`, contendo os seguintes métodos:

class Racional	
`Racional( int num, int den )`	cria um `Racional` com valor `num/dem`
`String toString()`	retorna a representação do número em string, na fração original `num/dem`
`int compareTo( Racional o )`	compara o `Racional` com `o` nos termos da interface `Comparable<Racional>`

(b) Na função `Main`, crie um vetor contendo 20 objetos do tipo `Racionais`, com valores variados (entre negativos e positivos) e utilize a função `Arrays::sort` para ordenar este vetor.

Filas e pilhas

Para as questões a seguir, quando necessário, utilize como base o código desenvolvido em aula e disponível através da turma da disciplina no Google Classroom.

[2] Dica: para criar esta funcionalidade, crie um novo array com comprimento `novaCap`, transfira os valores armazenados no array antigo para o novo, e substitua o array antigo pelo novo.

[3] Esta regra consistente é chamada *ordenação total*. Para mais detalhes, veja: `https://docs.oracle.com/javase/8/docs/api/java/lang/Comparable.html`.

[4] Dica: para comparar dois números racionais, primeiro exclua o caso em que eles tenham sinais diferentes (ou algum dos dois seja zero). Depois disso, se os dois números forem positivos, `a/b` vem antes de (é menor que) `c/d` se $ad < bc$; se os dois forem negativos, `-a/b` vem antes de `-c/d` se $ad > bc$.

Questão 4

Utilizando Generics, modifique a interface `Fila` e a classe `FilaVetor` para que possam armazenar objetos de um tipo genérico `T`.

Atenção: por limitação da linguagem Java, *arrays genéricos são proibidos*[5]: dado um tipo genérico `Tipo`, não podemos criar um vetor de objetos `Tipo` com uma linha de código como:

```
Tipo [] vetor = new Tipo[142857],
```

que seria a sintaxe ideal. Em vez disso, precisamos usar fazer um *typecast* como:

```
Tipo [] vetor = (Tipo []) new Object[142857]
```

para obter esse resultado, ao custo de um *warning* do compilador.[6]

Questão 5

A implementação feita em sala de `FilaVetor` deixa posições livres no início da fila conforme a fila vai sendo utilizada. Modifique a implementação da classe para que, após utilizar as posições ao final do vetor `conteudo`, a fila continue sendo preenchida a partir das posições no início do vetor.[7][8]

Questão 6

Utilizando Generics, modifique a interface `Pilha` e a classe `PilhaVetor` para que possam armazenar objetos de um tipo genérico `T` . (Veja a nota na questão 4.)

Questão 7

Imagine que um cliente utilizou uma `Pilha` para realizar uma série de operações `push` e `pop`. Não sabemos a sequência exata, mas sabemos que as chamadas a `push` inseriram os números de 0 a 9, nesta ordem.

(a) Se imprimirmos os valores retornados pelas chamadas a `pop`, quais destas sequências de valores podem ocorrer?

- `4 3 2 1 0 9 8 7 6 5`
- `4 3 2 1 0 5 6 7 8 9`
- `1 4 7 9 8 6 5 3 0 2`
- `4 6 8 7 5 3 2 9 0 1`
- `1 2 3 4 5 6 9 8 7 0`
- `2 1 4 3 6 5 8 7 9 0`
- `2 5 6 7 4 8 9 3 1 0`
- `0 4 6 5 3 8 1 7 2 9`

(b) Para os casos que podem ocorrer, qual é a sequência de chamadas que obtém esses resultados? (Exemplo: dez `push`, seguidos de dez `pop`.)

Questão 8

As implementações feitas em sala de `FilaVetor` e `PilhaVetor` fixam a capacidade máxima da estrutura de dados igual a uma constante `CAP`.

(a) Modifique estas classes para que o método `push` lance uma `LimitExceededException` caso a pilha/fila já esteja cheia.[9]

(b) Modifique estas classes para que `CAP` seja utilizada apenas como capacidade inicial e, caso uma chamada a `push` vá exceder a capacidade, o comprimento do vetor `conteudo`

[5] Para mais detalhes, recomendo esta leitura.

[6] Código com *typecast* não é seguro contra erros de *runtime*, e é importante restringir MUITO as formas de acesso a vetores criados assim. Mas essa construção é necessária, e até as bibliotecas padrão do Java a utilizam.

[7] Dica 1: Todos os métodos, exceto `get()`, terão que ser adaptados, e a adaptação é semelhante em todos.

[8] Dica 2: utilize o operador `%` (resto de divisão).

[9] Dica: como fazer para diferenciar os casos de pilha/fila cheia e vazia? É possível fazer isso olhando apenas para as posições do topo da pilha e do início e final da fila?

seja dobrado. (A sua implementação das classes `ListaExpansivel` e `ListaDinamica` na questão 2 podem servir de inspiração.)

(c) [DESAFIO] Modifique estas classe para que o comprimento de `conteudo` caia pela metade se a ocupação atual (após uma chamada a `pop`) seja igual a 25% da capacidade. (Esta é uma forma de economizar espaço em memória.)

Questão 9

Nas implementações feitas em sala de `FilaVetor` e `PilhaVetor`, uma chamada a `pop` não "apaga" o objeto do vetor `conteudo`. Em vez disso, apenas utilizamos os valores de `t_pos` (caso `PilhaVetor`) e `f_pos` (caso `FilaVetor`) para indicar que aquela posição de `conteudo` não é mais uma posição ocupada, e o valor armazenado lá é ignorado até ser sobreposto.

Podemos "apagar" esse conteúdo armazenando `null` antes de atualizar o valor de `t_pos`/`f_pos`. Considerando as funcionalidades do Java, qual é a vantagem desta estratégia?

Listas encadeadas

Para as questões a seguir, quando necessário, utilize como base o código desenvolvido em aula e disponível através da turma da disciplina no Google Classroom. Os métodos solicitados devem pertencer à classe `ListaEncadeada`, exceto quando dito o contrário, e as funções não podem acessar a estrutura interna das listas encadeadas.

Questão 10

Considere que `no` é um nó em uma lista encadeada. Se `no.prox` não é nulo, qual é o efeito da seguinte linha de código?

```
no.prox = no.prox.prox
```

Questão 11

Considere que, em certo ponto da execução de um método qualquer, a variável `Node meuNo` contém uma referência a um certo nó de uma lista encadeada. É possível, a partir desta referência, remover `meuNo` da lista encadeada? Se não, qual deve ser a "estratégia" para remover `meuNo`? Explique sua resposta.

Questão 12

Utilizando Generics, modifique a classe `ListaEncadeada` para que possa armazenar objetos de um tipo genérico `T`.

Questão 13

Implemente dois métodos, um iterativo e um recursivo, que percorram uma lista encadeada, imprimindo os valores armazenados nela em ordem.

Questão 14

Implemente um método na classe `ListaEncadeada` que receba um inteiro `k` e exclua o elemento na `k`-ésima posição (contando a partir de 0) de uma lista encadeada (e lance uma exceção adequada caso a lista tenha `k` elementos ou menos).

Questão 15

Implemente um método recursivo que imprima os valores armazenados em uma lista encadeada, em ordem reversa.

Questão 16

Crie uma classe `ListaEncadeadaOrdenada`, contendo a mesma estrutura interna da classe `ListaEncadeada` e um método adicional `inserir`, que recebe um número inteiro n e insere este número na lista. Os inteiros armazenados na lista devem sempre estar em ordem crescente, e `inserir` deve respeitar esta ordem.

Questão 17

Implemente um método que modifique uma lista encadeada para que ela passe a conter os mesmos valores em ordem inversa. (Faça uma implementação iterativa e uma recursiva.)

Questão 18

Implemente uma função que receba duas listas encadeadas ordenadas e crie uma nova lista encadeada, também ordenada. As listas originais devem ter seu conteúdo mantido.

Questão 19

Implemente uma função que receba duas listas encadeadas ordenadas e crie uma nova lista encadeada, também ordenada. As listas originais devem ter seu conteúdo removido.

Questão 20

Implemente classes para representar listas circulares, lista duplamente encadeadas, e listas circulares duplamente encadeadas. Em cada uma destas classes, implemente um método para inserir um novo elemento na k-ésima posição da lista (com k menor ou igual ao comprimento da lista).

Complexidade, busca e ordenação

Questão 21

Utilizando limites, determine se as afirmações abaixo são verdadeiras ou falsas.

(a) $3n^4 + 6n^2 + 9n = O(n^4)$

(b) $5n^3 + 2n^2 \log n = O(n^3)$

(c) $291n \log n + n^2 = O(n \log n)$

(d) $2^n + n^5 = O(n^5)$

(e) $2^n = O(3^n)$

(f) $2^{2 \log n} = O(n^2)$

Questão 22

Considere as afirmações a seguir:

- $460n^3 = O(n^5)$
- $460n^3 = O(n^4)$
- $460n^3 = O(n^3)$
- $460n^3 = O(n^2)$

Quais destas são verdadeiras? Se mais de uma for verdadeira, qual afirmação é mais forte? Justifique suas respostas.

Questão 23

Simplifique as funções abaixo utilizando a notação O:

(a) $f(n) = 5n + 2 \log n + 9$

(b) $g(n) = 2n^3 + 7n^2 + 10n^4$

(c) $h(n) = 4n^2 \log n + 3n + 500$

(d) $x(t) = e^t + \log t + 1$

Questão 24

Determine a complexidade de tempo, em função de n, dos trechos de código a seguir.

(a)
```
int tot = 0;
for (int i = n; i > 0; i /= 2)
    for(int j = 0; j < i; j++)
        tot++;
```

(b)
```
int tot = 0;
for (int i = 1; i < n; i *= 2)
    for (int j = 0; j < i; j++)
        tot++;
```

(c)
```
int tot = 0;
for (int i = 1; i < n; i *= 2)
    for (int j = 0; j < n; j++)
        tot++;
```

(d)
```
int tot = 0;
for (int i = 0; i < n; i++)
  for (int j = 1; j < i; j *= 2)
        tot++;
```

Questão 25 .....

Determine a complexidade de tempo do algoritmo que você implementou na questão 15. Implemente um método que realize a mesma tarefa, mas utilize uma solução iterativa, e determine sua complexidade de tempo.

Questão 26 .....

Projete um algoritmo que verifique se, em um vetor de inteiros, existem três consecutivos que somam zero, e determine sua complexidade.

Questão 27 .....

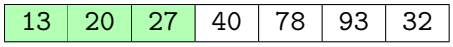
Considere a sua classe `ListaEncadeadaOrdenada` da questão 16. Projete um algoritmo, que busque um inteiro n na lista, retornando sua posição na lista caso ele exista, ou -1 caso não exista. Implemente este algoritmo como um método desta classe, determine a complexidade de tempo deste algoritmo, e compare com a complexidade da busca binária em um vetor ordenado.

Questão 28 .....

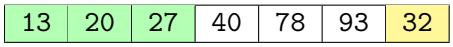
Implemente os algoritmos de ordenação *insertion sort*, *bubble sort*, e *merge sort*.

Questão 29 .....

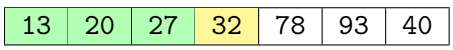
O algoritmo de ordenação conhecido como *selection sort* é baseado na seguinte estratégia. Dividimos nosso vetor em duas partes: uma seção esquerda, que contém os menores elementos existentes no vetor em ordem crescente, e uma seção direita, que contém o restante dos elementos vetor. A cada rodada, queremos expandir a seção esquerda em uma posição, e para isso, buscamos o menor elemento da seção direita e o trocamos de posição com o primeiro elemento desta seção.



Exemplo de uma rodada do *selection sort*. Após três rodadas do algoritmo, os três menores elementos estão em ordem na seção esquerda do vetor; a seção direita contém os demais.



Durante a quarta rodada, o algoritmo identifica o menor elemento da seção direita e troca este elemento com o primeiro da seção direita.



Com isso, agora a seção direita pode ser expandida em uma posição. Isto conclui essa rodada.

Elabore o algoritmo *selection sort* em pseudo-código, implemente-o, e determine sua complexidade de tempo e de espaço.

Questão 30 .....

"[O algoritmo] gnome sort *é baseado na técnica usada pelo gnomo de jardim (*tuinkabouter*) holandês tradicional. Eis como um gnomo de jardim ordena uma fila de vasos de flores.*

Basicamente, ele olha para o vaso à sua frente e o anterior; se eles estão na ordem certa, ele anda um vaso à frente, senão, ele troca os vasos e anda um vaso para trás. Condições de fronteira: se não existe um vaso anterior, ele dá um passo à frente; se não existe um vaso à frente dele, ele termina." (Dick Grune)

Elabore o algoritmo *gnome sort* em pseudo-código, implemente-o, e determine sua complexidade de tempo e de espaço.

Questão 31 .....

Projete um algoritmo que conte quantos pares de números iguais existem em um vetor, e avalie sua complexidade. Por exemplo, no vetor

```
[ 3 9 1 3 9 3 ]
```

existem quatro pares de números iguais, nas posições 0/3 (`3`), 0/5 (`3`), 1/4 (`9`), e 3/5 (`3`).

Questão 32 .....

Considere um vetor ordenado de números inteiros distintos.

(a) Uma *subsequência* de um vetor é uma sequência de números que aparecem consecutivamente neste vetor. Por exemplo, dado o vetor

```
[ 12 37 51 86 94 ]
```

a sequência `37 51 86` é subsequência deste vetor, mas a sequência `12 51 94` não é.

Elabore um algoritmo que, dado um vetor de inteiros, imprime todas as subsequências deste vetor, e determine sua complexidade em função do comprimento do vetor de entrada.

(b) O *k-sufixo* de uma subsequência consiste nos últimos k números desta subsequência. (Subsequências com comprimento menor do que k não possuem k-sufixo.)

Elabore um algoritmo que, dado um vetor ordenado de inteiros distintos, produza e imprima todos os possíveis k-sufixos distintos de suas subsequências, e determine sua complexidade em função de k e do comprimento do vetor de entrada. (Não é necessário imprimir as subsequências originais, apenas os k-sufixos.)

(c) Um *subconjunto* de um vetor é uma sequência de números que aparecem neste vetor, consecutivamente ou não. Por exemplo, dado o vetor

```
[ 12 37 51 86 94 ]
```

as sequências `37 51 86` e `12 51 94` são subconjuntos deste vetor, mas a sequência `51 86 95` não é.

Elabore um algoritmo que, dado um vetor de inteiros, imprime todos os subconjuntos deste vetor, e determine sua complexidade em função do comprimento do vetor de entrada.

Aplicações: estruturas lineares e recursão

Nos exercícios a seguir, para os algoritmos que exigirem pilhas ou filas, implemente-os utilizando os ADTs `Pilha` e/ou `Fila`, e não as classes que as implementam como `PilhaVetor` e `FilaVetor`. Desta forma, desacoplamos nosso código da implementação dos ADTs.

Questão 33 .....

Escreva uma função que receba uma string de caracteres e retorne o seu reverso (com as caracteres de trás pra frente). Você não pode acessar o conteúdo da string recebida exceto acessando posições individuais ou através de um `for` pelos seus caracteres.

Variante: Escreva uma função que receba uma frase na forma de uma string de caracteres e retorna a frase obtida revertendo cada palavra (mas mantendo a sequência de palavras). Além da restrição anterior, você não pode construir um array intermediário de palavras na sua solução.

Questão 34

Expressões aritméticas podem ser escritas na notação *pós-fixada*. Nesta notação, um operador é imediatamente precedido de todos os seus operandos, em ordem. Por exemplo: considere a expressão

```
2 3 + 5 *
```

em notação pós fixada. Os operandos do operador `+` são `2` e `3`, e os operandos do operador `*` são `5` e `2 3 +`. Assim, essa expressão é equivalente, na notação infixa tradicional, à expressão `(2 + 3) * 5`.

(a) Converta a expressão `(5 - 2) * (5 - 3) + 5` para a notação pós-fixada.

(b) Implemente um método que receba uma expressão aritmética em notação pós-fixada e calcule seu valor. Considere que a expressão somente contém os operadores `+`, `-`, `*`, e `/`, com seus significados habituais em programação. (Para simplificar sua implementação, você pode considerar que a expressão de entrada está no formato de um vetor, com cada número ou operador armazenado em posições consecutivas do vetor como uma string.)

(c) Implemente um método que receba uma expressão aritmética em notação pós-fixada e retorne uma expressão equivalente em notação pré-fixada. Por exemplo, se a expressão `2 3 + 5 *` for recebida, o método deverá retornar a expressão `* + 2 3 5`. (Seu método não deve inverter a ordem dos argumentos mesmo que o operador seja comutativo, como `+` ou `*`.)

Questão 35

Sistemas de arquivos normalmente organizam os arquivos presentes através de *diretórios*, que são estruturas que armazenam arquivos e, potencialmente, outros diretórios. Por exemplo, esta é um trecho da estrutura básica de um sistema Unix:

```
/
    usr/
        bin/
            gcc
            grep
            less
            sudo
        include/
            <diretório vazio>
        sbin/
            authserver/
            envvars
    var/
        log/
```

Neste exemplo, a raiz do sistema de arquivos possui dois diretórios (`usr` e `var`), que por sua vez armazenam outros diretórios. Um diretório pode armazenar somente arquivos (como o diretório `bin`), somente diretórios (como a raiz), tanto arquivos quanto diretórios (como `sbin`) ou estar vazio (como `authserver` e `log`).

(a) Crie classes `Arquivo` e `Diretorio` para representar arquivos e diretórios em um programa. Qual deve ser a relação estas classes?

(b) Implemente uma função que receba um diretório e imprima a estrutura dele, no formato apresentado acima. Diretórios são identificados por um caracter '`/`' no final do

nome, enquanto arquivos não possuem esse caracter. Depois de um diretório, deve ser impresso o seu conteúdo, tanto arquivos quanto diretórios, com 4 espaços adicionais de identação.

Questão 36 .....

Escreva uma função que receba uma string de parênteses, colchetes, e chaves, e determine se esta sequência está bem-balanceada, isto é, se cada operador fechando possui um correspondente abrindo com um aninhamento adequado. Por exemplo, sua função deve retornar `true` para a sequência `[()]{}{[()()]()}` e `false` para as sequências `([]`, `{[}]` e `][`.

Questão 37 .....

[DESAFIO] *Circuito de corrida.* O último item deste problema foi apresentado pelo prof. Jayme Szwarcfiter na primeira aula de Algoritmos e Estruturas de Dados na COPPE/UFRJ em 2010.

Considere o seguinte cenário: Ao longo de um circuito fechado (como um autódromo), se encontram diversas bombas de combustível, cada um contendo uma certa quantidade de combustível. Um carro de corrida, localizado em uma destas bombas e com tanque inicialmente vazio, começará a percorrer o circuito, abastecendo seu tanque em todas as bombas que encontrar com todo o combustível que estiver em cada bomba. Sabemos que, se juntarmos o conteúdo total de todas as bombas, isto é suficiente para que o carro dê exatamente uma volta no circuito. (Para simplificar o problema, considere que o consumo de combustível do carro é constante ao longo do percurso, então para percorrer p.ex. 25% do circuito, o carro precisa utilizar 25% do combustível disponível.)

(a) Determine, dadas as localizações das bombas, o conteúdo de cada uma delas, e a localização inicial do carro, projete um algoritmo que determine se o carro conseguirá completar o circuito ou se o tanque ficará vazio em algum momento (a famosa "pane seca"). Realize a modelagem do problema e a especificação da entrada de dados que seu algoritmo precisa receber.

(b) Mostre que, dadas as localizações das bombas e o conteúdo de cada uma delas, existe (ao menos) uma localização inicial para o carro que evita a pane seca. (Bônus: mostre que, se existir mais de uma localização inicial que evita a pane seca, então todas evitam a pane seca, e projete um algoritmo para decidir entre os dois casos.)