Pontifícia Universidade Católica de Minas Gerais
Curso de Ciência da Computação
Disciplina: Algoritmos e Estruturas de Dados II
Prof. Felipe Domingos da Cunha / Prof. Max do Val Machado

Trabalho Prático I

1 Regras Básicas

1. Estude bastante cada par de entrada/saída.
2. Todos os programas devem ser desenvolvidos na linguagem Java (exceto quando dito o contrário).
3. Todas as apresentações de trabalho devem ser feitas no Linux usando o editor VIM.
4. Como o combinado não sai caro:
   - As cópias de trabalho serão encaminhadas para o colegiado;
   - O aluno perderá 1 ponto para cada método não comentado ou com comentário inútil.
5. As exceções devem ser perguntadas/discutidas/negociadas com o professor.
6. Na primeira parte deste trabalho, você não pode usar os métodos das classes String, Integer, Double, ... Os únicos métodos permitidos são *char charAt(int)* e *int length()* da classe String.
7. Todas as classes devem ser entregues em um único arquivo. Essa regra será necessária para a submissão de trabalhos no Verde e no identificador de plágio utilizado na disciplina.
8. Use o padrão Java para comentários.
9. Na leitura de um número inteiro, se seu valor for vazio ou não número, ele recebe zero.
10. Nos exercícios de ordenação ou estruturas de dados, se dois registros tiverem a mesma chave de pesquisa, eles serão ordenados pelo atributo nome.
11. O arquivo **readme.txt** contém dicas sobre Linux, VIM e redirecionamento de entrada/saída.
12. Para contar as letras, consoantes e vogais desconsidere acentos e cedilha, ou seja, consideramos somente os caracteres cujos códigos ASCII estiverem entre 'A' e 'Z' ou 'a' e 'z'.
13. Não utilize as classes IO nem Scanner.

14. Para cada exercício: faça (entende-se análise, implementação e comentários), teste (várias vezes) e submeta no Verde. Os exercícios não testados/comentados serão penalizados.

15. A correção será automática através do Verde e de entrevista com o aluno nas aulas de laboratório.

16. A correção sempre será feita na versão do Linux disponível nos laboratórios. Qualquer problema devido ao uso de outras arquiteturas será de responsabilidade do aluno.

17. Fique atento ao Charset dos arquivos de entrada e saída.

2 Dicas sobre Recursividade

1. Um método recursivo pode ser usado como uma estrutura de repetição e um erro básico consiste em confundir tais estruturas com os métodos recursivos. Para evitar esse erro, neste trabalho, não podemos usar estruturas de repetição nos métodos recursivos.

2. Outro erro básico consiste em criar variáveis globais (em Java, atributos) para darem suporte à recursividade. Para evitar esse erro, neste trabalho, não podemos usar variáveis globais.

3. Um terceiro erro básico consiste em passar o valor de retorno como parâmetro. Para evitar esse erro, neste trabalho, não podemos passar parâmetros extras aos do enunciado, exceto um contador para recursividade. O método abaixo contém um exemplo desse erro.

```java
public boolean contarLetrasMinusculas(String s, int i, int resp){
    int resp;
    if(i == s.length()){
        resp = 0;
    }
    else if ((s.charAt(i) >= 'a' && s.charAt(i) <= 'z') == false){
        resp = contarLetrasMinusculas(s, i + 1, resp + 1);
    }
    else {
        resp = contarLetrasMinusculas(s, i + 1, resp);
    }
    return resp;
}
```

4. Quando criamos um método recursivo, normalmente, passamos um contador como parâmetro. Uma forma elegante de implementar essa solução recursiva consiste em criar dois métodos: o recursivo; e outro que chama o recursivo pela primeira vez e inicializa o valor do contador. Neste trabalho, devemos usar essa técnica em todos os métodos recursivos. Veja o exemplo abaixo:

```java

public boolean somenteLetrasMinusculas(String s){
    return somenteLetrasMinusculas(s, 0);
}
private boolean somenteLetrasMinusculas(String s, int i){
    boolean resp;
```

```
7       if(i == s.length()){
8           resp = true;
9       }
10      else if ((s.charAt(i) >= 'a' && s.charAt(i) <= 'z') == false){
11          resp = false;
12      }
13      else {
14          resp = somenteLetrasMinusculas(s, i + 1);
15      }
16      return resp;
17  }
```

5. Uma boa prática de programação é colocar um único *return* em cada método. Por exemplo, o método anterior é mais indicado que a versão abaixo.

```
1
2   public boolean somenteLetrasMinusculas(String s, int i){
3       if(i == s.length()){
4           return true;
5       }
6       else if ((s.charAt(i) >= 'a' && s.charAt(i) <= 'z') == false){
7           return false;
8       }
9       else {
10          return somenteLetrasMinusculas(s, i + 1);
11      }
12  }
```

3 Primeira Parte: Aquecimento

1. **Aquecimento** - Crie um método **iterativo** que receba como parâmetro uma string e retorne seu número de caracteres maiúsculos. Em seguida, teste o método anterior usando redirecionamento de entrada e saída. A entrada padrão é composta por várias linhas sendo que a última contém a palavra FIM[1]. A saída padrão contém um número inteiro para cada linha de entrada.

2. **Palíndromo** - Crie um método **iterativo** que recebe uma string como parâmetro e retorna true se essa é um palíndromo. Na saída padrão, para cada linha de entrada, escreva uma linha de saída com SIM / NÃO indicando se a linha é um palíndromo.

3. **Ciframento de César** - O Imperador Júlio César foi um dos principais nomes do Império Romano. Entre suas contribuições, temos um algoritmo de criptografia chamado "Ciframento de César". Segundo os historiadores, César utilizava esse algoritmo para criptografar as mensagens que enviava aos seus generais durante as batalhas. A ideia básica é um simples deslocamento de caracteres. Assim, por exemplo, se a chave utilizada para criptografar as mensagens for 3, todas as ocorrências do caractere 'a' são substituídas pelo caractere 'd', as do 'b' por 'e', e assim

[1] A entrada padrão dos demais exercícios é da mesma forma.

sucessivamente. Crie um método **iterativo** que recebe uma string como parâmetro e retorna outra contendo a entrada de forma cifrada. Neste exercício, suponha a chave de ciframento três. Na saída padrão, para cada linha de entrada, escreva uma linha com a mensagem criptografada.

4. **Alteração Aleatória** - Crie um método **iterativo** que recebe uma string, sorteia duas letras minúsculas aleatórias (código ASCII $\geq$ 'a' e $\leq$ 'z'), substitui todas as ocorrências da primeira letra na string pela segunda e retorna a string com as alterações efetuadas. Na saída padrão, para cada linha de entrada, execute o método desenvolvido nesta questão e mostre a string retornada como uma linha de saída. Abaixo, observamos um exemplo de entrada supondo que para a primeira linha as letras sorteados foram o 'a' e o 'q'. Para a segunda linha, foram o 'e' e o 'k'.

EXEMPLO DE ENTRADA:	EXEMPLO DE SAÍDA:
o rato roeu a roupa do rei de roma	*o rqto roeu q roupq do rei de romq*
e qwe qwe qwe ewq ewq ewq	*k qwk qwk qwk kwq kwq kwq*
FIM	

A classe Random do Java gera números (ou letras) aleatórios e o exemplo abaixo mostra uma letra minúscula na tela. Em especial, destacamos que: i) *seed* é a semente para geração de números aleatórios; ii) nesta questão, por causa da correção automática, a *seed* será quatro; iii) a disciplina de Estatística e Probabilidade faz uma discussão sobre "aleatório".

```
1 Random gerador = new Random();
2 gerador.setSeed(4);
3 System.out.println((char)('a' + (Math.abs(gerador.nextInt()) % 26)));
```

5. **Álgebra Booleana** - Crie um método **iterativo** que recebe uma string contendo uma expressão booleana e o valor de suas entradas e retorna um booleano indicando se a expressão é verdadeira ou falsa. Cada string de entrada é composta por um número inteiro n indicando o número de entradas da expressão booleana corrente. Em seguida, a string contém n valores binários (um para cada entrada) e a expressão booleana. Na saída padrão, para cada linha de entrada, escreva uma linha de saída com SIM / NÃO indicando se a expressão corrente é verdadeira ou falsa.

6. **Is** - Crie um método **iterativo** que recebe uma string e retorna true se a mesma é composta somente por vogais. Crie outro método **iterativo** que recebe uma string e retorna true se a mesma é composta somente por consoantes. Crie um terceiro método **iterativo** que recebe uma string e retorna true se a mesma corresponde a um número inteiro. Crie um quarto método **iterativo** que recebe uma string e retorna true se a mesma corresponde a um número real. Na saída padrão, para cada linha de entrada, escreva outra de saída da seguinte forma X1 X2 X3 X4 onde cada Xi é um booleano indicando se a é entrada é: composta somente por vogais (X1); composta somente somente por consoantes (X2); um número inteiro (X3); um número real (X4). Se Xi for verdadeiro, seu valor será SIM, caso contrário, NÃO.

7. **Palíndromo em C** - Na linguagem C, crie um método **iterativo** que recebe uma string como parâmetro e retorna true se essa é um palíndromo. Na saída padrão, para cada linha de entrada, escreva uma linha de saída com SIM / NÃO indicando se a linha é um palíndromo.

   Observação: Uma linha de entrada pode ter caracteres não letras.

8. **Leitura de Página HTTP** - Leia duas strings sendo que a primeira é o nome de uma página web e a segunda, seu endereço. Por exemplo, "Pontifícia Universidade Católica de Minas Gerais" e "www.pucminas.br". Em seguida, mostre na tela o número de vogais (sem e com acento), consoantes e dos padrões "$< br >$" e "$< table >$" que aparecem no código dessa página. A entrada padrão é composta por várias linhas. Cada uma contém várias strings sendo que a primeira é um endereço web e as demais o nome dessa página web. A última linha da entrada padrão contém a palavra "FIM". A saída padrão contém várias linhas sendo que cada uma apresenta o número de ocorrência (valor x_i entre parênteses) de cada caractere ou string solicitado. Cada linha de saída será da seguinte forma: a(x_1) e(x_2) i(x_3) o(x_4) u(x_5) á(x_6) é(x_7) í(x_8) ó(x_9) ú(x_{10}) à(x_{11}) è(x_{12}) ì(x_{13}) ò(x_{14}) ù(x_{15}) ã(x_{16}) õ(x_{17}) â(x_{19}) ê(x_{19}) î(x_{20}) ô(x_{21}) û(x_{22}) consoante(x_{23}) $< br >$(x_{24}) $< table >$(x_{25}) nomepágina(x_{26}).

9. **Arquivo em Java**: Faça um programa que leia um número inteiro n indicando o número de valores reais que devem ser lidos e salvos sequencialmente em um arquivo texto. Após a leitura dos valores, devemos fechar o arquivo. Em seguida, reabri-lo e fazer a leitura de trás para frente usando os métodos getFilePointer e seek da classe RandomAccessFile e mostre todos os valores lidos na tela. Nessa questão, você não pode usar, arrays, strings ou qualquer estrutura de dados. A entrada padrão é composta por um número inteiro n e mais n números reais. A saída padrão corresponde a n números reais mostrados um por linha de saída.

10. **Arquivo em C**: Refaça a questão anterior na linguagem C.

11. **RECURSIVO - Aquecimento** - Refaça a questão **Aquecimento** de forma recursiva.

12. **RECURSIVO - Palíndromo** - Refaça a questão **Palíndromo** de forma recursiva.

13. **RECURSIVO - Ciframento de César** - Refaça a questão **Ciframento de César** de forma recursiva.

14. **RECURSIVO - Alteração Aleatória** - Refaça a questão **Alteração Aleatória** de forma recursiva.

15. **RECURSIVO - Álgebra Booleana** - Refaça a questão **Álgebra Booleana** de forma recursiva.

16. **RECURSIVO - Is** - Refaça a questão **Is** de forma recursiva.

17. **RECURSIVO - Palíndromo em C** - Refaça a questão **Palíndromo em C** de forma recursiva.

4 Segunda Parte: Estruturas Sequenciais, Pesquisa e Ordenação

A Proclamação da República no Brasil aconteceu no dia 15 de novembro de 1889 e vigora até os dias atuais. O imperador Dom Pedro II foi deposto e o comando ficou a cargo do Marechal Deodoro da Fonseca. A difusão dos ideais republicanos remonta ao período colonial, como durante a Inconfidência Mineira e a Conjuração Baiana, no final do século XVIII. Apesar dos ideais e das revoltas buscarem a superação da monarquia, apenas no final do século XIX, com o fim do escravismo, as elites agrárias do país aceitaram organizar o Estado brasileiro nos moldes republicanos. O fato de a República nascer como uma aceitação das elites e ter sido realizada através da espada do exército brasileiro conformou um caráter autoritário e excludente do Estado brasileiro, garantindo os privilégios das classes dominantes e a negação de direitos às classes exploradas durante muito tempo. A participação do exército na vida política nacional foi também uma constante da história republicana do país (brasilescola.uol.com.br/historiab/brasil-republica2.htm, em 3/1/2019).

O arquivo **presidente.zip** contém um conjunto de páginas web com informações sobre os presidentes do Brasil. Esse conjunto de páginas foi baixado da Wikipédia em 3/1/2019 e **sofreu algumas adaptações** para ser utilizado neste e nos próximos trabalhos práticos. Tal arquivo deve ser copiado para a pasta /tmp/. Quando reiniciamos o Linux, ele normalmente apaga os arquivos existentes na pasta /tmp/.

Implemente os itens pedidos a seguir.

1. **Classe**: Crie uma classe *Presidente* seguindo todas as regras apresentadas no slide unidade01g_conceitosBasicos_introducaoOO.pdf. Sua classe terá os atributos privados id (int), nome (String), idade (int), dataNascimento (LocalDateTime), localNascimento (String), inicioMandato (LocalDateTime), fimMandato (LocalDateTime), dataMorte (LocalDateTime), localMorte (String), antecessor (String), sucessor (String), vice (String), pagina (String) e paginaTam (long). O atributo *id* corresponde à ordem em que a pessoa foi presidente; *pagina*, ao nome da página html; *paginaTam*, ao tamanho da página html. Sua classe também terá pelo menos dois construtores, e os métodos *gets*, *sets*, *clone*, *imprimir* e *ler*. O método *imprimir* mostra os atributos do restrito (ver cada linha da saída padrão) e o *ler* lê os atributos de um registro.

   A entrada padrão é composta por várias linhas e cada uma contém uma string indicando a página html a ser lida. A última linha da entrada contém a palavra FIM. A saída padrão também contém várias linhas, uma para cada registro contido em uma linha da entrada padrão.

2. **Lista com Alocação Sequencial**: Crie uma Lista de registros baseada na de inteiros vista na sala de aula. Sua lista deve conter todos os atributos e métodos existentes na lista de inteiros,

contudo, adaptados para a classe *Presidente*. Lembre-se que, na verdade, temos uma lista de ponteiros (ou referências) e cada um deles aponta para um registo. Neste exercício, faremos inserções, remoções e mostraremos os elementos de nossa lista.

Os métodos de inserir e remover devem operar conforme descrito a seguir, respeitando parâmetros e retornos. Primeiro, o *void inserirInicio(Presidente presidente)* insere um registro na primeira posição da Lista e remaneja os demais. Segundo, o *void inserir(Presidente presidente, int posição)* insere um registro na posição p da Lista, onde $p < n$ e n é o número de registros cadastrados. Em seguida, esse método remaneja os demais registros. O *void inserirFim(Presidente presidente)* insere um registro na última posição da Lista. O *Presidente removerInicio()* remove e retorna o primeiro registro cadastrado na Lista e remaneja os demais. O *Presidente remover(int posição)* remove e retorna o registro cadastrado na p-ésima posição da Lista e remaneja os demais. O *Presidente removerFim()* remove e retorna o último registro cadastrado na Lista.

A entrada padrão é composta por duas partes. A primeira é igual a entrada da primeira questão. As demais linhas correspondem a segunda parte. A primeira linha da segunda parte tem um número inteiro n indicando a quantidade de registros a serem inseridos/removidos. Nas próximas n linhas, tem-se n comandos de inserção/remoção a serem processados neste exercício. Cada uma dessas linhas tem uma palavra de comando: II inserir no início, I* inserir em qualquer posição, IF inserir no fim, RI remover no início, R* remover em qualquer posição e RF remover no fim. No caso dos comandos de inserir, temos também um número inteiro indicando a linha (no arquivo censo.dat) do registro a ser inserido. No caso dos comandos de "em qualquer posição", temos também a posição. No Inserir, a posição fica imediatamente após a palavra de comando. A saída padrão tem uma linha para cada registro removido sendo que essa informação será constituída pela palavra "(R)" e o atributo **nome**. No final, a saída mostra os atributos relativos a cada registro cadastrado na lista após as operações de inserção e remoção.

3. **Pilha com Alocação Sequencial**: Crie uma Pilha de registros baseada na pilha de inteiros vista na sala de aula. Neste exercício, faremos inserções, remoções e mostraremos os elementos de nossa pilha. A entrada e a saída padrão serão como as da questão anterior, contudo, teremos apenas os comandos I para inserir na pilha (empilhar) e R para remover (desempilhar).

4. **Fila Circular com Alocação Sequencial**: Crie uma classe *Fila Circular* de *Presidente*. Essa fila deve ter tamanho cinco. Em seguida, faça um programa que leia vários registros e insira seus atributos na fila. Quando o programa tiver que inserir um registro e a fila estiver cheia, antes, ele deve fazer uma remoção. A entrada padrão será igual à da questão anterior. A saída padrão será um número inteiro para cada registro inserido na fila. Esse número corresponde à média **arredondada** da idade dos presidentes contidas na fila após cada inserção. Além disso, para cada registro removido da fila, a saída padrão também apresenta a palavra "(R)" e alguns

atributos desse registro. Por último, a saída padrão mostra os registros existentes na fila seguindo o padrão da questão anterior.

5. **Lista com Alocação Sequencial em C**: Refaça a questão "Lista com Alocação Sequencial" na linguagem C.

6. **Pesquisa Sequencial**: Faça a inserção de alguns registros no final de uma Lista e, em seguida, faça algumas pesquisas sequenciais. A chave primária de pesquisa será o atributo **nome**. A entrada padrão é composta por duas partes onde a primeira é igual a entrada da primeira questão. As demais linhas correspondem a segunda parte. A segunda parte é composta por várias linhas. Cada uma possui um elemento que deve ser pesquisado na Lista. A última linha terá a palavra FIM. A saída padrão será composta por várias linhas contendo as palavras SIM/NÃO para indicar se existe cada um dos elementos pesquisados. Além disso, crie um arquivo de log na pasta corrente com o nome matrícula_sequencial.txt com uma única linha contendo sua matrícula, tempo de execução do seu algoritmo e número de comparações. Todas as informações do arquivo de log devem ser separadas por uma tabulação '\t'.

7. **Pesquisa Binária**: Repita a questão anterior, contudo, usando a Pesquisa Binária. A entrada e a saída padrão serão iguais às da questão anterior. O nome do arquivo de log será matrícula_binaria.txt. A entrada desta questão está ordenada.

8. **Ordenação por Seleção**: Na classe Lista, implemente o algoritmo de ordenação por seleção considerando que a chave de pesquisa é o atributo **id**. A entrada e a saída padrão são iguais às da primeira questão, contudo, a saída corresponde aos registros ordenados. Além disso, crie um arquivo de log na pasta corrente com o nome matrícula_selecao.txt com uma única linha contendo sua matrícula, número de comparações (entre elementos do *array*), número de movimentações (entre elementos do *array*) e o tempo de execução do algoritmo de ordenação. Todas as informações do arquivo de log devem ser separadas por uma tabulação '\t'.

9. **Ordenação por Seleção Recursiva**: Repita a questão anterior, contudo, usando a Seleção Recursiva. A entrada e a saída padrão serão iguais às da questão anterior. O nome do arquivo de log será matrícula_selecaoRecursiva.txt.

10. **Ordenação por Inserção**: Repita a questão de Ordenação por Seleção, contudo, usando o algoritmo de Inserção, fazendo com que a chave de pesquisa seja o atributo **paginaTam**. O nome do arquivo de log será matrícula_insercao.txt.

11. **Shellsort**: Repita a questão de Ordenação por Seleção, contudo, usando o algoritmo Shellsort, fazendo com que a chave de pesquisa seja o atributo **pagina**. O nome do arquivo de log será matrícula_shellsort.txt.

12. **Heapsort**: Repita a questão de Ordenação por Seleção, contudo, usando o algoritmo Heapsort, fazendo com que a chave de pesquisa seja o atributo **dataNascimento**. O nome do arquivo de log será matrícula_heapsort.txt.

13. **Quicksort**: Repita a questão de Ordenação por Seleção, contudo, usando o algoritmo Quicksort, fazendo com que a chave de pesquisa seja o atributo **inicioMandato**. O nome do arquivo de log será matrícula_quicksort.txt.

14. **Counting Sort**: Repita a questão de Ordenação por Seleção, contudo, usando o algoritmo Counting Sort, fazendo com que a chave de pesquisa seja o atributo calculado **idade**. O nome do arquivo de log será matrícula_countingsort.txt.

15. **Bolha**: Repita a questão de Ordenação por Seleção, contudo, usando o algoritmo da Bolha, fazendo com que a chave de pesquisa seja o atributo **localNascimento**. O nome do arquivo de log será matrícula_bolha.txt.

16. **Mergesort**: Repita a questão de Ordenação por Seleção, contudo, usando o algoritmo Mergesort, fazendo com que a chave de pesquisa seja o atributo **fimMandato**. O nome do arquivo de log será matrícula_mergesort.txt.

17. **Radixsort**: Repita a questão de Ordenação por Seleção, contudo, usando o algoritmo Radixsort, fazendo com que a chave de pesquisa seja o atributo **id**. O nome do arquivo de log será matrícula_radixsort.txt.

18. **Quicksort em C**: Refaça a questão "Quicksort" na linguagem C.

19. **Ordenação PARCIAL por Seleção**: Refaça a Questão "Ordenação por Seleção" considerando a ordenação parcial com k igual a 10.

20. **Ordenação PARCIAL por Inserção**: Refaça a Questão "Ordenação por Inserção" considerando a ordenação parcial com k igual a 10.

21. **Heapsort PARCIAL**: Refaça a Questão "Heapsort" considerando a ordenação parcial com k igual a 10.

22. **Quicksort PARCIAL**: Refaça a Questão "Quicksort" considerando a ordenação parcial com k igual a 10.