Universidade Federal do Rio Grande do Norte
Instituto Metrópole Digital

Linguagem de Programação I • IMD0030
– 1ª Avaliação, 24 de março de 2015 –

A PROVA TEM A DURAÇÃO DE **100 MINUTOS**.
LEIA ATENTAMENTE AS INSTRUÇÕES ABAIXO ANTES DE INICIAR A PROVA

- ⋆ *Esta é uma prova individual e com consulta restrita.*
- ⋆ *Você terá 100 minutos para a realização da prova. As respostas devem ser fornecidas na forma de um arquivo digital compactado contendo seus programas.*

1 Apresentação

Nesta avaliação você deve realizar duas tarefas gerais, descritas a seguir. Você poderá baixar via Sigaa os arquivos que compõem o projeto. O acesso a Internet será suspenso após o download do projeto.

2 Parte I: Implementar classe `Vetor`

Implemente o restante da classe `Vetor` que representa um arranjo unidimensional de inteiros de tamanho n. A classe está definida em `include/vetor.h` e implementada em `src/vetor.cpp`. Esta classe está parcialmente implementada e os métodos que você precisa desenvolver estão marcados com `TODO`. Instruções sobre o funcionamento de cada método foram fornecidas na forma de comentários no arquivo `src/vetor.cpp`.

Em particular você terá a opção de implementar sobrecarga de 3 operadores (`=`, `[]` e `==`) OU métodos equivalente, caso você ainda não entenda bem como funciona sobrecarga de operador. Por exemplo, você pode implementar o método `operator=()` *ou* optar por implementar o método `assign()`. Ambos têm a mesma finalidade, sendo o primeiro por sobrecarga do operador de atribuição '=' do C++. Caso você opte por implementar a sobrecarga de operador, descomente os `defines` correspondentes no início do arquivo `src/vetor.cpp`, visto que o programa principal usa *compilação condicional* para compilar um versão ou outra.

No projeto você vai encontrar um programa `drive_vetor.cpp` cujo objetivo é realizar *testes unitários* com cada método que você deve desenvolver. Para cada teste bem sucedido, o programa indica a pontuação (parcial) correspondente. Esta pontuação é provisória e poderá ser ajustada posteriormente.

Para que seja possível compilar desde o início os métodos que precisam ser criados foram programados na forma de *stubs*. Isso quer fizer que foi adicionado um código mínimo apenas para realizar a compilação e que, obviamente, não corresponde ao código real que deve ser criado por você.

Para compilar a partir do terminal de dentro da pasta raiz `Projeto_Vetor` use:

```
g++ -Wall -std=c++0x drive_vetor.cpp src/vetor.cpp -I include -o drive
```

3 Parte II: Implementar busca pela menor distância

O programa `distancia.cpp` atua como cliente da classe `Vetor` e cria uma coleção (arranjo), `aoBag`, de objetos do tipo `Vetor` e um objeto-alvo, `oTarget`, também do tipo `Vetor`.

Você deve implementar o código da função `searchSmallestDistance()` que deve retornar o índice de um elemento em `aoBag` que possuir e *menor distância* para `aoBag`. A função para calcular a distância entre dois objetos deve ser informada por meio de *ponteiro para função*.

Para compilar a partir do terminal de dentro da pasta raiz `Projeto_Vetor` use:

```
g++ -Wall -std=c++0x distancia.cpp src/vetor.cpp -I include -o distancia
```

Você também deverá implementar duas funções diferentes para cálculo de distâncias, descritas a seguir.

3.1 Distância Euclideana

A distância Euclideana, `euclideanDist()` é a mais tradicional distância Cartesiana, definida como

$$d_e(\mathbf{p}, \mathbf{q}) = \sqrt{\sum_{i=1}^{n}(q_i - p_i)^2}, \tag{1}$$

onde $\mathbf{p} = (p_1, p_2, \ldots, p_i, \ldots, p_n)$ e $\mathbf{q} = (q_1, q_2, \ldots, q_i, \ldots, q_n)$ são vetores, ambos de tamanho n, e p_i e q_i são, respectivamente, seus componentes.

3.2 Distância Manhattan

A distância Manhattan, `manhattanDist()`, também pode ser aplicada sobre o plano Cartesiano, sendo definida como

$$d_m(\mathbf{p}, \mathbf{q}) = \sum_{i-1}^{n} |(p_i - q_i)|, \tag{2}$$

onde $\mathbf{p} = (p_1, p_2, \ldots, p_i, \ldots, p_n)$ e $\mathbf{q} = (q_1, q_2, \ldots, q_i, \ldots, q_n)$ são vetores, ambos de tamanho n, e p_i e q_i são, respectivamente, seus componentes.

3.3 Exemplos

See $\mathbf{p} = (2, 3, 4)$ e $\mathbf{q} = (7, 8, 9)$, temos $\begin{cases} d_e(\mathbf{p}, \mathbf{q}) = \sqrt{(7-2)^2 + (8-3)^2 + (9-4)^2} = 8.66. \\ \text{e} \\ d_m(\mathbf{p}, \mathbf{q}) = |2-7| + |3-8| + |4-9| = 15. \end{cases}$

$\sim$ FIM $\sim$