

INF 213 - Lista por contiguidade

Objetivo: praticar alocação dinâmica de memória e entender melhor conceitos de listas por contiguidade.

→ LEMBREM-SE DE USAR PAPEL E CANETA COMO RASCUNHO ANTES DE IMPLEMENTAR <<--

Arquivos fonte e diagramas utilizados nesta aula:

https://drive.google.com/open?id=1JKIwlvn03znBCXmJYFA_1pjOmZtag0sQ

Etapa 1

O arquivo MyVec.h contém a implementação parcial da lista por contiguidade vista em sala de aula. Porém, parte do código foi removido. Sua tarefa consiste em completar o código e testar a classe utilizando o programa TestaMyVec.cpp (a saída esperada se encontra no final do TestaMyVec.cpp).

Etapa 2

Adicione uma função chamada `eraseMatchingElements` a sua classe (essa função não está disponível nos vetores disponibilizados pela STL). Tal função deverá receber um argumento e remover do vetor todos elementos iguais a esse argumento (o argumento deverá ser passado por referência? Referência constante? A função deverá ser constante?). Ao final deve-se retornar quantos elementos foram removidos (a capacidade do vetor não deve ser alterada).

Por exemplo, se o MyVec `v` armazena caracteres e `v=[a,b,c,b,w,z]`, então após a chamada `v.eraseMatchingElements('b')` o valor de `v` deverá ser `[a,c,w,z]` e a chamada da função deverá ter retornado o número 2 (visto que dois caracteres 'b' foram removidos de `v`).

Use o programa `testaFuncoes.cpp` para testar sua função (entenda o funcionamento da entrada desse programa e crie entradas para avaliar o seu programa). Ao usar o programa `testaFuncoes.cpp`, comente a parte dele que testa funções ainda não implementadas.

Obs: sua função não deve alocar mais memória (isso facilitaria bastante a implementação, mas seria menos eficiente). Exercício: como alocar mais memória poderia ajudar?

Exercício (escreva a resposta a esse exercício em comentários por cima da sua implementação da função `eraseMatchingElements`): qual a ordem de complexidade (pior caso) da função `eraseMatchingElements`? É possível melhorar isso?

Etapa 3

Adicione uma função chamada `sortedInsert` que recebe um argumento e insere tal argumento no vetor de modo a mantê-lo ordenado (tal função assume que antes da função ser chamada o vetor já estava ordenado e ela deverá ser o mais eficiente possível (ou seja, deverá usar esse

fato para NÃO precisar reordenar o array a cada inserção usando algum método de ordenação dos vistos em aula)).

Exemplo 1: se o vetor (MyVec) v armazena caracteres e $v=[a,c,w,z]$, então a chamada `v.sortedInsert('b')` deverá transformar v em: `[a,b,c,w,z]`

Exemplo 2: se o vetor v inicialmente estiver vazio (um vetor vazio é considerado ordenado), então as chamadas: `v.sortedInsert('c')` ; `v.sortedInsert('b')` ; `v.sortedInsert('z')` ; `v.sortedInsert('w')` ; farão com que v se torne: `[b,c,w,z]`

Observacao

O programa `testaFuncoes.cpp` pode ser utilizado para realizar testes nas funcoes desenvolvidas nas etapas 2 e 3. O sistema Submittity realizará testes na sua implementacao utilizando `testaFuncoes.cpp` com várias entradas possíveis. Faça vários testes para tentar encontrar possíveis bugs em sua implementação.

→ LEMBRARAM DE USAR PAPEL E CANETA COMO RASCUNHO ANTES DE IMPLEMENTAR ? <<--

Submissao do exercicio

Envie sua implementação da classe MyVec (MyVec.h) pelo sistema submittity (não envie outros arquivos).

A solucao deve ser submetida utilizando o sistema submittity (submittity.dpi.ufv.br). Atualmente a submissao so pode ser realizada dentro da rede da UFV.