Universidade Federal do Ceará - Campus Quixadá
QXD0010 – Estruturas de Dados
Prof. Fabio Dias

Atividade Mãe - Listas Lineares

A implementação da estrutura de dados descrita neste documento deve ser entregue até a meia-noite do dia **08/06/2023** pelo Moodle.

Leia atentamente as instruções abaixo.

Instruções:

- Esta atividade pode ser feito em **dupla** ou **individualmente** e deve ser implementado (preferencialmente) usando a linguagem de programação C++ **(Não aceitarei mais do que dois alunos por atividade)**
- Identifique o seu código-fonte colocando o **nome** e **matrícula** dos integrantes da dupla como comentário no início de seu código.
- Indente corretamente o seu código para facilitar o entendimento. **Trabalhos com códigos maus indentados sofrerão redução na nota.**
- A estrutura de dado devem ser implementadas como TAD. Deve haver um arquivo de cabeçalho (.h) e um arquivo de implementação (.cpp) para a estrutura de dado programada.
- Os programas-fonte devem estar devidamente organizados e documentados.
- Observação: Lembre-se de desalocar os endereços de memória alocados quando os mesmos não forem mais ser usados.
- **Observação: Qualquer indício de plágio resultará em nota ZERO para todos os envolvidos.**

1 Lista Sequencial Dupla

Em sala de aula, vimos duas implementações da lista linear, a lista sequencial e a lista encadeada. Nessas implementações vimos duas operações para adicionarmos elementos na lista, a *push_back* e *push_front*, que adicionam um elemento no final e inicio da lista, respectivamente. A complexidade de tempo dessas funções no caso médio (**quando desconsideramos o tempo da função resize, já que nem sempre ela é executada**) são as seguintes: Na lista sequencial é de $O(1)$ e $O(n)$, respectivamente. A *push_front* tem essa complexidade pela necessidade de realizarmos um deslocamento a direita dos elementos da lista para que possamos inserir o elemento no inicio da lista. Já na lista encadeada, as complexidades são $O(n)$ e $O(1)$, respectivamente. A *push_back* tem essa complexidade pela necessidade de percorrermos a lista toda para chegarmos no último nó.

Gostaríamos de uma implementação de lista linear que forneça complexidade O(1), no caso médio, para as ambas as operações. Pensando nisso, o professor Fábio Dias bolou a estrutura de dados *Lista Sequencial Dupla*.

A lista sequencial vista em sala de aula possui três atributos, um vetor (*m_list*), o tamanho da lista (*m_size*) e capacidade da lista (*m_capacity*). Os elementos são armazenado no vetor *m_list* do início para o final do vetor, ou seja, o primeiro elemento é armazenado no índice 0, o segundo no índice 1 e assim sucessivamente. De forma geral, o próximo elemento é armazenado no índice *m_size*. Logo, o índice do elemento na lista é o mesmo do vetor, ou seja, o elemento de índice 4 da lista está armazenado no índice 4 do vetor.

Na *Lista Sequencial Dupla*, os elementos da lista são armazenados do meio para os extremos no vetor *m_list*, de forma que teremos espaço para inserirmos um elemento tanto no final como no inicio da lista sem a necessidade de realizarmos um deslocamento dos elementos do vetor. Logo, a complexidade dessas funções serão $O(1)$.

Para isso, essa nova ED terá dois novos atributos, dois inteiros *m_head* e *m_tail*. *m_head* e *m_tail* irão indicar qual índice do vetor iremos adicionar o próximo elemento quando a função *push_front* e *push_back* for executada, respectivamente.

Vejamos a configuração inicial dessa lista. *m_capacity* será inicialmente igual a 16 e *m_head* será igual a 7 e *m_tail* igual a 8. Toda vez que *push_front* for executada, o elemento será adicionado no índice *m_head* e, depois ele será decrementado. De forma semelhante, toda vez que *push_back* for executada, o elemento será adicionado no índice *m_tail* e, depois ele será incrementado. Dessa forma, os elementos serão adicionados do meio para os extremos do vetor e a complexidade média das operações *push_front* e *push_back* será $O(1)$.

A Figura 1 ilustra a configuração inicial da Lista Sequencial Dupla.

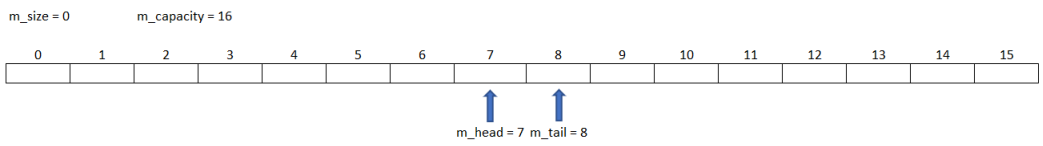

Figura 1: Configuração inicial da Lista Sequencial Dupla.

Suponha que as funções sejam executadas nessa sequencia, *push_back*(78), *push_front*(55), *push_back*(93), *push_back*(20), então o vetor da lista ficaria conforme a Figura 2:

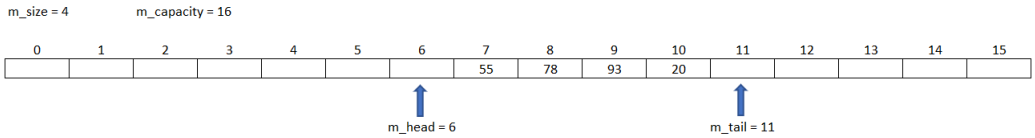

Figura 2: Configuração da Lista Sequencial Dupla após a adição de alguns elementos.

ATENÇÃO: Como citado anteriormente, na lista sequencial normal, o índice do elemento na lista é o mesmo do vetor, ou seja, o elemento de índice 4 na lista está armazenado no índice 4 do vetor. O mesmo não ocorre na lista *Lista Sequencial Dupla*. Tirando como exemplo a Figura 2, o elemento de índice 0 da lista é o 55, que está no índice 7 do vetor. O elemento de índice 3 da lista é o 20, que está no índice 10 do vetor.

Continuaremos a ter as operações de deslocamento e redimensionamento. Elas serão executadas nas seguintes situações.

1.1 Deslocamento

Teremos duas operações de deslocamento, que chamaremos de deslocamento completo e deslocamento parcial. Vejam abaixo a descrição de ambas e quando usa-las.

1.1.1 Deslocamento Completo

A operação de deslocamento completo será executada quando ainda existir espaço disponível no vetor, e quando uma das seguintes situações abaixo ocorrer:

1. Quando a operação de *push_front* for executada e não tiver espaço disponível no início do vetor;

2. Quando a operação de *push_back* for executada e não houver espaço disponível no final do vetor.

Então, a operação de deslocamento deve ser feito para dividir igualmente (se posível) os espaços livres entre o inicio e final do vetor.

Por exemplo, quando $m\_capacity = 16$, $m\_size = 10$, $m\_head = 5$ e $m\_tail = 16$. Veja Figura 3:

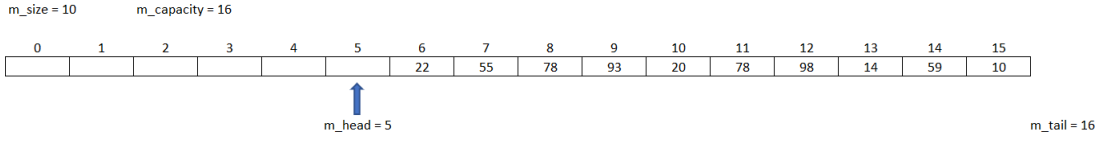

Figura 3: Configuração inicial da Lista Sequencial Dupla.

Nesse caso, como ainda existe espaço na frente do vetor, se a operação *push_front* for executada, então apenas inserirmos sem realizar o deslocamento. Agora se for a operação *push_back*, então devemos executar a operação de deslocamento, já que não existe espaço disponível no final do vetor. Vamos dividir igualmente, **se possível**, os espaços livres. Como temos 6 espaços livres na frente do vetor, iremos deslocar os elementos no vetor em 3 posições para a esquerda, para liberar 3 posições atrás do vetor.

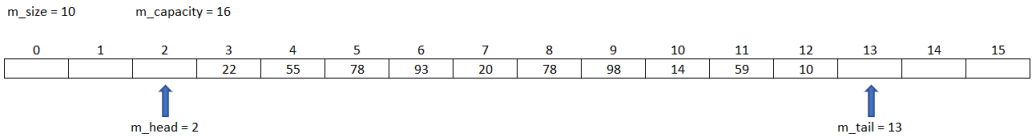

Figura 4: Lista Sequencial Dupla após o deslocamento para esquerda em 3 posições.

Uma situação similar será quando a operação a ser executada for *push_front* e na frente do vetor não tiver espaço disponíveis. Então, se ainda houver espaço disponíveis no vetor, vamos dividi-los e fazer o deslocamento para a direita, para liberar espaço na frente do vetor.

Agora, quando não houver mais espaço disponível, então iremos chamar a operação de redimensionamento.

1.1.2 Deslocamento Parcial

O deslocamento parcial será executada quando um elemento da lista for removido. Nesse caso, desvemos deslocar em uma posição apenas uma parte dos elementos do vetor, os elementos que estão a esquerda ou direita do elemento a ser removido. Observe que dentro do contexto da lista sequencial dupla, podemos deslocar esses elementos tanto para a esquerda quanto para a direita.

O deslocamento será feito na direção que tiver menos espaços disponíveis, ou seja, se na frente do vetor tiver menos espaços disponíveis do que atrás, então iremos deslocar os elementos que estão a esquerda do elemento removido para a direita em uma posição. Caso contrário, se for atrás do vetor que tiver menos espaço, então iremos deslocar os elementos que estão a direita do elemento removido para a esquerda em uma posição.

Por exemplo, imagine que na lista da Figura 2, removemos o elemento 78. Como atrás do vetor temos 5 espaços disponíveis e na frente 7, então iremos fazer o deslocamento dos elementos a direita do 78 em uma posição para a esquerda. Dessa forma, o espaço disponível atrás aumento em uma unidade, conforme podemos vê na Figura 5

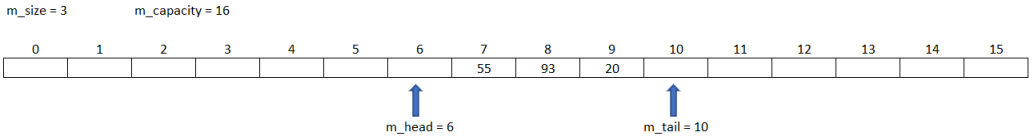

Figura 5: Lista Sequencial Dupla após a remoção do elemento 78 na figura 2.

1.2 Redimensionamento

Devemos alocar um novo vetor, duplicando a capacidade da lista. Depois copiamos os elementos do vetor antigo para o novo, deixando a quantidade iguais (se possível) de espaços disponível na frente e atrás do vetor.

1.3 Problema

Implemente em C++ o Tipo Abstrato de Dados Lista Sequencial Dupla. A sua estrutura de dados deve ser encapsulada por uma classe chamada `DoubleVector`, que

deve suportar as seguintes operações (vocês podem definir outras funções para auxiliar, caso achem necessário):

- `DoubleVector()`: Construtor da classe. Deve iniciar todos os atributos da classe de acordo com a descrição acima da lista.
- `DoubleVector(int n)`: Construtor da classe. Aloca o vetor com capacidade n, deixando espaços disponíveis na frente e atrás iguais.
- `~DoubleVector()`: Destrutor da classe. Libera memória previamente alocada.
- `int size()`: Retorna o numero de elementos na lista.
- `bool empty()`: Retorna `true` se a lista estiver vazia e `false` caso contrário.
- `void push_back(int value)`: Insere o inteiro *value* ao final da lista.
- `int pop_back()`: Remove elemento do final da lista e retorna seu valor.
- `void push_front(int value)`: Insere o inteiro *value* no inicio da lista.
- `int pop_front()`: Remove elemento do inicio da lista e retorna seu valor.
- `int at(int k)`: Retorna o elemento da lista de índice k (Cuidado, é o índice da lista e não do vetor). A função verifica se k esta dentro dos limites de elementos validos. Caso contrário, retorna -1. **Obrigatoriamente deve ser** $O(1)$.
- `void shift()`: Realiza o deslocamento completo, seja para a direita ou esquerda, de acordo a descrição na Seção 1.1.1. Caso desejem, podem dividir em duas funções $left\_shift$ e $right\_shift$.
- `void rezise()`: Realiza o redimensionamento, de acordo a descrição na Seção 1.2.
- `void remove(int k)`: Remove o elemento de índice k da lista (Cuidado, é o índice da lista e não do vetor) de acordo a descrição na Seção 1.1.2. A ordem dos demais elementos da lista devem se manter.
- `void removeAll(int value)`: Remove da lista todas as ocorrências de *value*. A ordem dos demais elementos da lista entre se devem se manter.
- `void print()`: Imprime os elementos da lista.
- `void printReverse()`: Imprime os elementos da lista em ordem reversa.
- `void concat(DoubleVector& lst)`: Concatena a lista atual com a lista `lst` passada por parâmetro. A lista lst não é modificada nessa operação. Os elementos de lst serão adicionados na ordem no final da lista do objeto da função.
- `bool equal(DoubleVector& lst)`: Determina se a lista passada por parâmetro é igual à lista em questão. Duas listas são iguais se elas possuem o mesmo tamanho e o valor do k-ésimo elemento da primeira lista é igual ao k-ésimo elemento da segunda lista.

1.4 O que deve ser submetido

- **Deverá ser submetido:**
  - Os arquivos .h e .cpp da TAD lista sequencial dupla.
  - O arquivo main.cpp com a função principal criando uma lista sequencial dupla e add e removendo alguns elemento, para vê se vocês sabem usar.
  - Os fonte devidamente organizados e documentados.

- Um dos parâmetros utilizados na avaliação da qualidade de uma implementação consiste na constatação da presença ou ausência de comentários. Comente o seu código. Mas também não comente por comentar, forneça bons comentários.

- Não serão aceitos trabalhos submetidos após o prazo final.

- Estou a disposição para quaisquer (+-) dúvidas, exceto dúvidas do tipo: "Não entendi a ED". Essa irei responder com "Até próximo semestre!"