CMP1054 - Estrutura de Dados I
Lista de Exercícios - Listas
Max Gontijo de Oliveira

1. Seja `V` o tipo de dado a ser armazenado em uma lista, considere os seguintes métodos públicos de uma implementação de lista.

Método	Descrição
`void add_begin(V v)`	Insere `v` no início da lista.
`void add_end(V v)`	Insere `v` no fim da lista.
`void add_at(V v, int index)`	Insere `v` na posição `index` da lista. Caso `index` seja menor do que zero ou maior do que a quantidade de elementos da lista, deverá ser lançada uma exceção. Observação: nenhum elemento da lista deverá ser perdido. Se `index` for a posição de um elemento já existente, então, `v` deverá ser inserido entre os elementos `index-1` e `index+1`, de modo que `v` passe a figurar como o elemento da posição `index`.
`V set(V v, int index)`	Altera para `v` o valor do elemento localizado na posição `index` e retorna o antigo valor que ocupava essa posição. Caso `index` não corresponda a nenhuma posição existente na lista, deverá ser lançada uma exceção. Observação: a quantidade de elementos da lista não será alterada. Apenas o valor do elementos da posição `index` que passará a ser `v`.
`V get_first()`	Retorna o valor do primeiro elemento da lista. Caso a lista esteja vazia, deverá ser lançada uma exceção.
`V get_last()`	Retorna o valor do último elemento da lista. Caso a lista esteja vazia, deverá ser lançada uma exceção.
`V get(int index)`	Retorna o valor do elemento da lista cujo índice seja `index`. Caso `index` não corresponda a nenhuma posição existente na lista, deverá ser lançada uma exceção.
`V remove_first()`	Remove o primeiro elemento da lista e retorna o valor removido. Caso a lista esteja vazia, deverá ser lançada uma exceção.
`V remove_last()`	Remove o último elemento da lista e retorna o valor removido. Caso a lista esteja vazia, deverá ser lançada uma exceção.
`V remove_at(int index)`	Remove da lista o elemento cujo índice seja `index` e retorna o valor removido. Caso `index` não corresponda a nenhuma posição existente na lista, deverá ser lançada uma exceção.
`int size()`	Retorna a quantidade de elementos da lista.
`bool is_empty()`	Retorna `true` caso a lista esteja vazia ou `false` caso contrário.
`void clear()`	Remove todos os elementos da lista.

(a) Implemente uma lista utilizando vetor com o nome `ListaVetor` que tenha os seguintes atributos privados:
- `int count; // quantidade de elementos da lista`
- `int capacity; // tamanho do vetor apontado por vec`
- `V* vec; // ponteiro para o vetor de elementos da lista`

(b) Implemente uma lista simplesmente encadeada com o nome `ListaSE` que tenha os seguintes atributos privados:
- `int count; // quantidade de elementos da lista`
- `Node* first; // ponteiro para o primeiro nó da lista`
- `Node* last; // ponteiro para o último nó da lista`

Considere que a classe `Node` tenha apenas os seguintes atributos públicos:
- `V v; // valor do nó`
- `Node* next; // endereço do próximo nó`

(c) Implemente uma lista duplamente encadeada com o nome `ListaDE` que tenha os seguintes atributos privados:
- `int count; // quantidade de elementos da lista`
- `Node* first; // ponteiro para o primeiro nó da lista`
- `Node* last; // ponteiro para o último nó da lista`

Considere que a classe `Node` tenha apenas os seguintes atributos públicos:
- `V v; // valor do nó`
- `Node* next; // endereço do próximo nó`
- `Node* prev; // endereço do nó anterior`

(d) Implemente uma lista duplamente encadeada utilizando nós sentinelas com o nome `ListaNosSentinelas` que tenha os seguintes atributos privados:

- `int count; // quantidade de elementos da lista`
- `Node first; // nó sentinela cujo atributo next aponta para o primeiro elemento`
- `Node last; // nó sentinela cujo atributo prev aponta para o último elemento`

Considere que a classe `Node` tenha apenas os seguintes atributos públicos:

- `V v; // valor do nó`
- `Node* next; // endereço do próximo nó`
- `Node* prev; // endereço do nó anterior`

2. Para todas as listas implementadas na questão 1, crie os seguintes métodos públicos. Note que tais métodos não são exatamente pertinentes à uma lista. Os métodos dessa questão tem o aprendizado como propósito maior.

Método	**Descrição**
`void print()`	Imprime na tela (no terminal), o valor de todos os elementos da lista, da primeira à última posição. Pode ser um debaixo do outro, na mesma linha ou com a formatação que você achar melhor.
`void print_desc()`	Imprime na tela (no terminal), o valor de todos os elementos da lista de forma invertida, da última à primeira posição. Pode ser um debaixo do outro, na mesma linha ou com a formatação que você achar melhor.
`void add_sorted(V v)`	Método deve considerar que os elementos da lista estão ordenados e inserir o valor `v` na lista de modo ordenado. Ou seja, se o valor `v` for maior do que o valor do elemento da posição `index` e menor (ou igual) ao valor do elemento da posição `index + 1`, então o valor `v` deverá ser inserido entre esses dois elementos. Assim, o valor `v` seria inserido na posição `index + 1`. Obviamente, se o valor `v` for menor (ou igual) ao valor do primeiro elemento da lista, o valor `v` deverá ser inserido no início da lista; ao passo em que se o valor `v` for maior do que o valor do último elemento da lista, o valor `v` deverá ser inserido no final da lista.
`bool is_sorted()`	Verifica se os elementos da lista estão armazenados de forma ordenada (crescente), retornando `true` caso estejam ou `false` caso não estejam. Caso a lista esteja vazia, o método deverá considerar que a lista está ordenada, retornando o valor `true`.
`bool is_fibonacci()`	Considerando que a lista tenha s elementos (tamanho da lista), esse método deve verificar se os elementos da lista estão armazenados de forma que representem os s primeiros termos de uma série de Fibonacci, retornando `true` caso representem ou `false` caso contrário. Caso a lista esteja vazia, o método deverá retornar `false`. Uma série de Fibonacci tem como seus dois primeiros termos, os valores 0 e 1. Os termos seguintes são criados pela soma dos dois anteriores. Por exemplo: 0, 1, 1, 2, 3, 5, 8, 13 representam os 8 primeiros termos a série de Fibonacci.

`bool contains(LISTA &lst)`	Considerando que `LISTA` seja o tipo da lista em questão onde o método está sendo implementado (`ListaVetor`, `ListaSE`, `ListaDE`, `ListaNoSentinela`), esse método deve retornar `true` caso a lista em questão tenha em seus elementos, todos os elementos da lista `lst`. A ordem dos elementos não deve influenciar. A lista em questão pode ter mais elementos do que a lista `lst`. Por exemplo:
	`ListaDE L1;`
	`ListaDE L2;`
	`ListaDE L3;`
	`L1.add(10);`
	`L1.add(20);`
	`L1.add(30);`
	`L2.add(30);`
	`L2.add(20);`
	`L3.add(10);`
	`L3.add(30);`
	as chamadas `L1.contains(L2);` e `L1.contains(L3);` iriam retornar `true`, enquanto as chamadas `L2.contains(L1);`, `L2.contains(L3);`, `L3.contains(L1);` e `L3.contains(L2);` iriam retornar `false`.

`bool equals(LISTA &lst)`	Considerando que `LISTA` seja o tipo da lista em questão onde o método está sendo implementado (`ListaVetor`, `ListaSE`, `ListaDE`, `ListaNoSentinela`), esse método deve retornar `true` caso a lista em questão tenha exatamente os mesmos elementos da lista `lst`, na mesma ordem.