# Eficiencia de listas (e exercicios)

Prof. Salles Magalhaes

# Revisao de iteradores

- Exercicios:
  - Indique as principais diferencas e semelhancas entre um iterador e um apontador?
  - Quais as vantagens de iteradores?
  - Como iteradores fornecem acesso uniforme a diferentes tipos de containers, um codigo que usa iteradores de vetores dinamicos pode ser facilmente adaptado para processar listas encadeadas. A afirmacao anterior e’ verdadeira ou falsa?

# Revisao de iteradores

- Exercicios: criar funcoes genericas (não membro):
  - Funcao apagaPosPares(v), para apagar todos elementos em posicoes pares (posicoes 0,2,4,6….) de uma lista e de um vetor dinamico. (você tera que implementar um método *erase* na classe MyVec…)
  - Funcao ehPalindrome que, dado um container (MyVec ou MyList2) armazenando chars, retorna true se a ordem deles representar uma palindrome e false caso contrario. Qual sera a ordem de complexidade da funcao para cada container?
  - Funcao “soma” que, dados dois iteradores it1 e it2 para um mesmo container de inteiros (sabemos que após incrementar it1 algumas vezes eventualmente chegaremos a it2), soma todos elementos que estiverem no intervalo [it1,it2).
  - Funcao “removeMatchingElements(v,value)” que, dado um container v e um valor value, remove todos elementos de v com valor igual a value. Sua funcao devera utilizar iteradores. Qual a ordem de complexidade para diferentes tipos de containers?

# Validade de iteradores

- Após inserir um elemento?
  - Vector:
    - Iteradores antes da posição para vector continuam validos (EXCETO se a capacidade for alterada), outros iteradores podem apontar para elementos diferentes.
  - Lista:
    - apenas conectividade pode mudar
- Após remover um elemento:
  - Vector:
    - Todos iteradores após a posição sao invalidados
    - Na verdade, eles provavelmente apontarao para posicoes após a removida (mas sem garantia)
  - Lista:
    - continuam validos (apenas conectividade muda)

# Eficiencia de listas

Qual a eficiencia (ordem de complexidade) de cada uma das operacoes na tabela seguinte?

Obs:

- Nem todas operacoes (mesmo algumas eficientes) foram implementadas nas nossas estruturas de dados.
- Muitas vezes operacoes não eficientes para determinadas estruturas de dados não sao implementadas (por exemplo, nas bibliotecas). Nesse caso, indique a eficiencia que seria obtida se elas fossem implementadas.

| | Lista por contiguidade | Lista simplesmente encadeada | Lista duplamente encadeada |
|---|---|---|---|
| size() | | | |
| push_back | | | |
| empty() | | | |
| pop_back | | | |
| push_front | | | |
| pop_front | | | |
| insert | | | |
| erase | | | |
| find | | | |
| eraseMatchingElements/remove | | | |

| | Lista por contiguidade | Lista simplesmente encadeada | Lista duplamente encadeada |
|---|---|---|---|
| size() | O(1) | **O(1)** ou O(n) (STL) | **O(1)** ou O(n) (STL) |
| push_back | O(1) | O(1) | O(1) |
| empty() | O(1) | O(1) | O(1) |
| pop_back | O(1) | O(1) | O(1) |
| push_front | O(n) | O(1) | O(1) |
| pop_front | O(n) | O(1) | O(1) |
| insert | O(n) | O(n) -- se inserir na pos. exata | O(1) |
| erase | O(n) | O(n) -- se remover a pos. exata | O(1) |
| find | O(n) | O(n) | O(n) |
| eraseMatchingElements/remove | O(n) ou O(n^2) (impl. simples) | O(n) | O(n) |

| | Lista por contiguidade | Lista simplesmente encadeada | Lista duplamente encadeada |
|---|---|---|---|
| Proximo elemento (iterator) | | | |
| Elemento anterior (iterator) | | | |
| Pular k elementos com iterador | | | |
| Pular -k (voltar) elementos com iterador | | | |
| Derreferenciar iterador | | | |
| Comparar dois iteradores (==) | | | |
| Comparar dois iteradores (<) | | | |
| Distancia entre dois iteradores | | | |
| Iterador “no meio” de dois iteradores | | | |
| Atribuir um iterador a outro | | | |
| Trocar dois elementos de posição (swap) | | | |

| | Lista por contiguidade | Lista simplesmente encadeada | Lista duplamente encadeada |
|---|---|---|---|
| Proximo elemento (iterator) | O(1) | O(1) | O(1) |
| Elemento anterior (iterator) | O(1) | O(1) | O(1) |
| Pular k elementos com iterador | O(1) | O(k) | O(k) |
| Pular -k (voltar) elementos com iterador | O(1) | O(n) | O(k) |
| Derreferenciar iterador | O(1) | O(1) | O(1) |
| Comparar dois iteradores (==) | O(1) | O(1) | O(1) |
| Comparar dois iteradores (<) | O(1) | O(n) | O(n) |
| Distancia entre dois iteradores | O(1) | O(n) | O(n) |
| Iterador “no meio” de dois iteradores | O(1) | O(n) | O(n) |
| Atribuir um iterador a outro | O(1) | O(1) | O(1) |
| Trocar dois elementos de posição (swap) | O(1) | O(1) | O(1) |

# Eficiencia de outras operacoes

- Busca binaria
- Operacao “splice”
- Operacao “unique”: remove elementos iguais consecutivos (exceto primeiro de cada grupo)
- Operacao “reverse”

# Eficiencia de outras operacoes

- Operacao “splice”
- Disponivel, por exemplo, nas listas disponiveis na STL/C++.
- Funcao membro: void splice (iterator position, list& x, iterator first, iterator last):
    - Move os elementos (da lista x) que estiverem no intervalo [first,last) para a posição “position” da lista.
    - Exemplo (supondo que temos um “splice” na classe MyList2) ilustrando o comportamento desejado na proxima pagina.

# Eficiencia de outras operacoes

```
MyList2<int> lista1;
…. // lista1 = {10,50}
MyList2<int> lista2;
…. // lista2 = {1,2,5,8}
MyList2<int>::iterator it1 … // it1 aponta para 50 (lista 1)
MyList2<int>::iterator it2first,it2last; //it2first aponta para 2, it2last aponta para 8

lista1.splice(it1,lista2,it2first,it2last);
for(int i:lista1)
        cout << i << " "; //imprime 10 2 5 50
cout << endl;
for(int i:lista2)
        cout << i << " "; //imprime 1 8
cout << endl;
```

# Eficiencia de outras operacoes

- Qual seria a ordem de complexidade da funcao splice para um vetor dinamico?
- E para uma lista (duplamente) encadeada?
- Tente implementar essa funcao na nossa classe MyList2. (para praticar apontadores e entender melhor a implementacao de listas, não use iteradores nem reuse funcoes ja implementadas)