IMD0029 – ESTRUTURAS DE DADOS BÁSICAS I

PROF. EIJI ADACHI M. BARBOSA

Lista de Exercícios – Lista Encadeada

Obs.: Além das questões nesta lista, vejam também as questões das provas passadas.

Questão 1. Dada uma lista simplesmente encadeada com apenas um ponteiro para o primeiro nó da lista, projete algoritmos **recursivos** para as seguintes funções:

a) `int Max( List L1 ) – Retorna o maior elemento da lista L1`

b) `int Sum( List L1 ) – Retorna a soma dos elementos da lista L1`

c) `int CountMin( List L1, int x ) – Retorna quantos elementos da lista L1 são menores do que x`

Obs.: Caso necessário, pode alterar a assinatura das funções.

Questão 2. Dada uma lista simplesmente encadeada com apenas um ponteiro para o primeiro nó da lista, projete um algoritmo **recursivo** para a seguinte função:

`void Invert(List L1) – Inverte a ordem dos elementos da lista L1`

Ex.: Se a lista de entrada for igual a L1 = {1 --> 5 --> 7 --> 3 --> 4}, após a execução de L1->Invert(), a lista ficará igual a {4 --> 3 --> 7 --> 5 --> 1}.

Obs.: A solução não poderá criar uma lista auxiliar, nem qualquer outra estrutura auxiliar (ex.: array). A solução deverá operar apenas sobre a lista de entrada, alterando sua estrutura.

Questão 3: Dadas duas listas duplamente encadeadas com sentinelas cabeça (Head) e cauda (Tail), projete um algoritmo **iterativo** para a seguinte função:

`List Merge(List L1, List L2) – Retorna uma nova lista contendo todos os elementos de L1 e L2 em ordem crescente. Pode assumir que as listas L1 e L2 já estão ordenadas em ordem decrescente.`

Dica: Esta solução é apenas uma versão do algoritmo merge usado no MergeSort. No MergeSort visto em sala de aula, nós operamos sobre arrays. Para esta solução, deveremos operar sobre listas duplamente encadeadas.

Questão 4: Dada uma lista duplamente encadeada com sentinelas cabeça (Head) e cauda (Tail), projete um algoritmo **iterativo** para a seguinte função:

`void RemoveRepeated(List L1) – Remove os elementos repetidos da lista L1, retornando quantos elementos foram removidos.`

Obs.: Esta função não deverá remover todos os elementos cujos valores se repetem; ele deverá deixar o primeiro elemento na lista, removendo os seguintes. Ex.: A lista { 7 ↔ 2 ↔ 4 ↔ 2 ↔ 3 ↔ 9 ↔ 9 ↔ 3 }, após a execução da função deverá ficar igual a { 7 ↔ 2 ↔ 4 ↔ 3 ↔ 9 }.

Questão 5: Dadas duas listas simplesmente encadeadas com apenas um ponteiro para o primeiro nó da lista, projete algoritmos **iterativos** para as seguintes funções:

a) `List Union( List L1, List L2 )` – Retorna uma nova lista representando a união das listas L1 e L2
b) `List Intersection( List L1, List L2 )` – Retorna uma nova lista representando a interseção das listas L1 e L2
c) `int Complement( List L1, List L2 )` – Retorna uma nova lista representando o complemente de L1 em relação a L2

Questão 6: Muitas aplicações acessam um mesmo dado em curtos períodos de tempo. Para este tipo de aplicação, uma estratégia para otimizar o acesso aos dados comumemnte implementada em listas encadeadas é a estratégia *move-to-front* (MTF). Nesta estratégia, a cada busca realizada com sucesso, o elemento buscado é movido para o início da lista. Dada uma lista duplamente encadeada com sentinelas cabeça (Head) e cauda (Tail), projete um algoritmo **iterativo** para a seguinte função:

`Node SearchMTF(List L1, int k)` – Retorna o nó da lista L1 cujo conteúdo tem valor igual a K. Caso a busca tenha sucesso, o nó deverá ser reposicionado para o início da lista.

Questão 7: Uma outra estratégia para otimizar o acesso aos dados numa lista encadeada é a estratégia da contagem de acessos. Nesta estratégia, mantém-se um atributo extra em cada nó para contar o número de vezes que aquele nó foi acessado. Desta forma, a cada vez que um nó é buscado, o seu atributo que conta o número de acessos é incrementado. Além disso, após cada busca realizada com sucesso na lista, os nós devem ser rearranjados de modo que fiquem ordenados em ordem decrescente em relação ao número de acesso dos nós. Dada uma lista duplamente encadeada com sentinelas cabeça (Head) e cauda (Tail), projete um algoritmo **iterativo** para a seguinte função:

`Node SearchCount(List L1, int k)` – Retorna o nó da lista L1 cujo conteúdo tem valor igual a K. Caso a busca tenha sucesso, o contador do nó deverá ser incrementado, e lista deverá ser reajustada de modo a manter os nós ordenados em ordem decrescente em relação ao contador.

Questão 8: Dada uma lista duplamente encadeada com sentinelas cabeça (Head) e cauda (Tail), projete algoritmos **iterativos** para as seguintes funções:

a) `void BubbleSort( List L1 )` – Ordena a lista L1 em ordem crescente usando uma versão do Bubble Sort.
b) `void InsertionSort( List L1 )` – Ordena a lista L1 em ordem crescente usando uma versão do Insertion Sort.
c) `void SelectionSort( List L1 )` – Ordena a lista L1 em ordem crescente

`usando uma versão do Selection Sort.`

Obs.: Para as funções acima, implemente e use uma função swap(Node n1, Node n2). Tal função deverá de fato trocar os nós de lugar, e não apenas trocar os seus conteúdos.

Dica: existem dois casos distintos para trocar os nós: quando eles são vizinhos, e quando não são vizinhos.

Questão 9: Dada uma lista simplesmente encadeada com apenas um ponteiro para o primeiro nó da lista, projete um algoritmo **iterativo** para a seguinte função:

`bool InsertOrdered(List L1, int x) – Insere o elemento x, mantendo a ordenação crescente da lista.`

Questão 10: Uma lista circular é um tipo de lista encadeada em que o último elemento aponta para o primeiro. No caso de listas duplamente encadeadas circulares, o ponteiro 'anterior' do primeiro elemento aponta para o último elemento. Neste contexto, implemente uma lista duplamente encadeada circular, provendo as seguintes operações:

a) `InsertBegin`
b) `InsertEnd`
c) `Insert(i)`
d) `RemoveBegin`
e) `RemoveEnd`
f) `Remove(i)`
g) `Get(i)`
h) `Search(x)`

Dica: Nos casos de lista circular, é necessário ter cuidado ao iterar sobre a lista para que não se caia num loop infinito.

Questão 11: Determinar se lista encadeada é palíndromo.

Questão 12: Determinar se existe loop numa lista encadeada.

Questão 13: Ordenar uma lista duplamente encadeada que só tem 0s 1s e 2s em tempo O(n).

Questão 14: Remover duplicatas.