MAC–IME–USP

Carlos Eduardo Ferreira
e-mail cef@ime.usp.br

MAC121 - Algoritmos e Estruturas de Dados I

Segundo semestre de 2020

Lista de exercícios– Listas Ligadas e Árvores – entrega: 14 de dezembro

IMPORTANTE: Todos os exercícios são interessantes, e devem ser feitos para aprender a matéria.

Vocês devem entregar apenas os exercícios **5 e 28**

1. Escreva uma função que recebe uma lista ligada ordenada e remove da lista os elementos repetidos, deixando apenas uma cópia de cada elemento.

2. Escreva funções para as seguintes operações:

   (a) verifica se um dado `x` ocorre numa lista ligada;

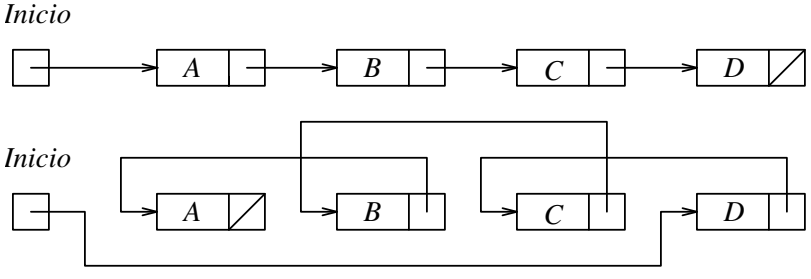
   (b) acrescenta um elemento no fim de uma lista ligada;

   (c) imprima os conteúdos de todos os elementos de uma lista ligada.

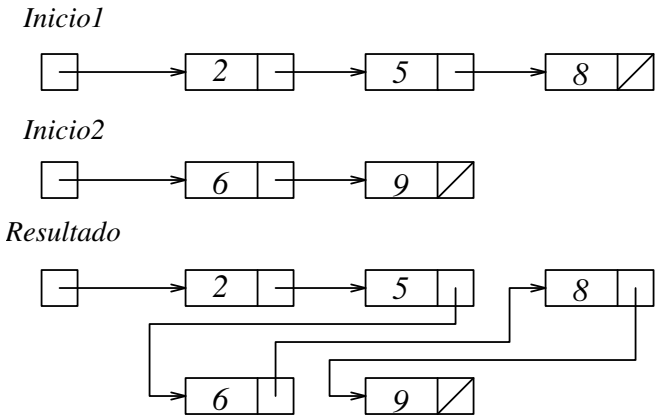
3. Faça uma função que devolve um apontador para o elemento do meio de uma lista ligada (se o número de elementos da lista for par, devolve o $\frac{n}{2}$-ésimo elemento) **sem contar o número de elementos da lista**.

4. Discuta vantagens e desvantagens de vetores (arrays) em relação a listas ligadas. Dê atenção especial às questões de quantidade de memória, velocidade de inserção, remoção e acesso.

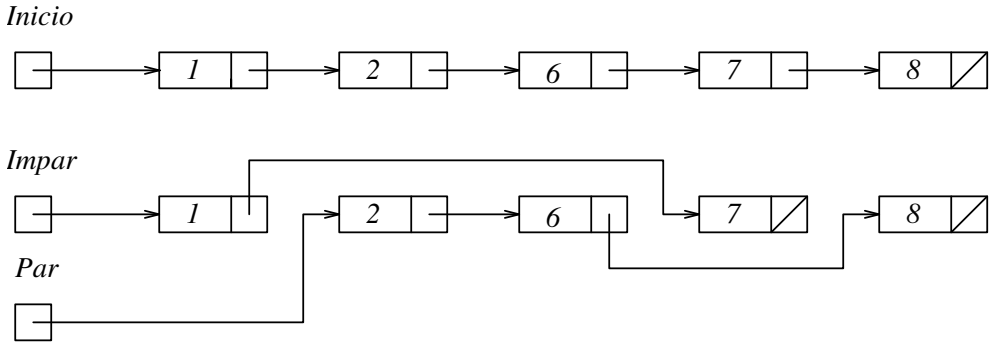
5. Escreva uma função que inverte uma lista ligada dada (o primeiro elemento da nova lista é o último da lista dada, o segundo é o penúltimo da lista dada, e assim por diante. Faça manipulando apenas os apontadores. Exemplo:



6. Escreva uma função para intercalar duas listas ligadas cujas informações estão arranjadas em ordem crescente. Exemplo:



7. Dada uma lista ligada escreva uma função que transforma a lista dada em duas listas ligadas: a primeira contendo os elementos cujos conteúdos são pares e a segunda com os elementos cujos conteúdos são ímpares. Sua função deve manipular somente os apontadores e **não** o conteúdo das células (i.e. não vale ficar copiando os conteúdo de um lado para o outro, só vale alterar os apontadores). Exemplo:



8. Dizemos que uma lista ligada é **circular** se o último elemento aponta para o primeiro. Uma lista ligada circular normalmente é referenciada com um apontador para o último elemento (que aponta, então, para o primeiro). Considere uma lista ligada circular, apontada por `fim`, como descrito acima, e um elemento `x` e faça uma função que remove da lista todas as ocorrências de `x` (cuidado para não entrar em loop...).

9. Dizemos que uma lista é **duplamente ligada** se cada elemento tem um apontador para o próximo elemento da lista e um apontador para o elemento anterior.
   a. Escreva a definição de tipos de uma lista duplamente ligada.
   b. Mostre como fica uma lista duplamente ligada com um único elemento
   c. Como ficaria uma lista duplamente ligada circular?
   d. Escreva funções para inserção e remoção de elementos em uma lista duplamente ligada circular.

10. Dizemos que uma lista ligada (duplamente ligada, circular) tem **cabeça de lista** se ela tem um elemento extra que está presente mesmo quando a lista não tem elementos válidos. Escreva funções de inserção e remoção para:
    a. listas ligadas com cabeça de lista.
    b. listas ligadas circulares com cabeça de lista.
    c. listas duplamente ligadas com cabeça de lista.
    d. listas duplamente ligadas circulares com cabeça de lista.

11. Faça o exercício 8 considerando, agora, uma lista ligada circular com cabeça de lista.

12. Faça uma implementação de pilhas (`empilha`, `desempilha`, `pilhaVazia`, etc) usando:
    a. listas ligadas;
    b. listas ligadas circulares;
    c. listas duplamente ligadas;
    d. listas duplamente ligadas circulares com cabeça de lista.

13. Faça o exercício anterior implementando agora filas.

14. Considere uma implementação de polinômios esparsos usando listas ligadas circulares com cabeça de lista.
    a. Faça uma função que recebe dois polinômios e devolve a soma dos dois;
    b. Faça uma função que recebe um polinômio e devolve o polinômio derivada;
    c. Faça uma função que recebe um polinômio $p(x)$ e um real a e devolve o polinômio obtido do quociente de $p(x)$ por $x - a$ e devolve também um real r, que é o resto desta divisão. (Dica: algoritmo de Briot-Ruffini)

15. Podemos representar uma matriz esparsa $A_{m\times n}$ usando a seguinte estrutura:

```
typedef struct cel {
             int lin, col;
             double coef;
             struct cel *proxNaMesmaLin, *antNaMesmaLin,
                        *proxNaMesmaCol, *antNaMesmaCol;
         } celula;
typedef celula *  apontador;

typedef struct {
          int m, /* número de linhas  */
              n, /* número de colunas */
              r; /* número de elementos não nulos*/
          apontador *vetorDeLinhas, *vetorDeColunas;
        } matrizEsparsa;
```

    Ou seja, cada elemento não nulo da matriz está em uma lista duplamente ligada circular com cabeça de lista dos elementos na mesma linha (cuja cabeça está em `vetorDeLinhas`) e em uma lista ligada circular com cabeça de lista dos elementos na mesma coluna.

a. Desenhe a estrutura correspondente à matriz esparsa $A_{10\times10}$ abaixo:

$$\begin{pmatrix} 0 & 0 & -1 & 0 & 0 & 3 & 0 & 0 & 1 & 13 \\ 7 & 0 & 0 & 0 & 0 & 0 & -8 & 0 & 0 & 0 \\ 0 & 0 & 0 & 4 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2 & 0 & 0 & 0 & -5 & 0 & 0 \\ 0 & -3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 23 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 19 & 0 \\ 0 & 0 & 0 & 0 & 12 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

b. Faça uma função `criaMatrizEsparsa()` que leia m, n e r elementos não nulos de uma matriz esparsa e constroi a estrutura conforme descrito acima.

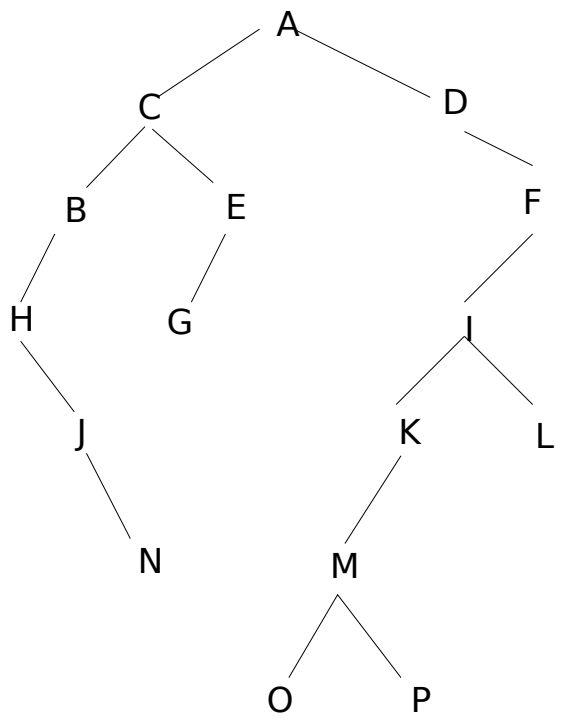
c. Faça uma função que recebe duas matrizes esparsas A e B e devolve a matriz esparsa dada pela soma das matrizes.

16. Faça uma função que conta o número de nós de uma árvore binária.

17. Faça uma função que recebe dois apontadores para nós x e y de uma árvore binária e decide se x é ancestral de y.

18. Refaça o exercício anterior supondo que a implementação de árvore tenha, em cada nó, um ponteiro para o nó pai.

19. Considere a árvore binária abaixo:



Liste os vértices visitados quando a árvore é percorrida em pré-ordem, in-ordem e pós-ordem.

20. Faça uma função que recebe dois vetores com a lista de nós visitados em uma árvore binária quando percorrida em pré-ordem e in-ordem e devolve uma cópia da árvore original.

21. Considere uma implementação de árvore binária em que temos um ponteiro para o pai de cada nó (o pai da raiz é `NULL`). A **profundidade** de um nó é a distância entre o nó e a raiz da árvore. Por exemplo, a profundidade da raiz é 0, dos seus filhos é 1, e assim por diante. Faça uma função que recebe um nó de uma árvore e determina sua profundidade.

22. Escreva uma função que imprima o conteúdo de cada nó de uma árvore binária precedido de um recuo em relação à margem esquerda do papel proporcional à sua profundidade.

23. Faça uma função que recebe um nó de uma árvore binária e devolve um ponteiro para o próximo nó quando a árvore é percorrida em in-ordem. Pode considerar a implementação com o ponteiro para o pai, se você preferir.

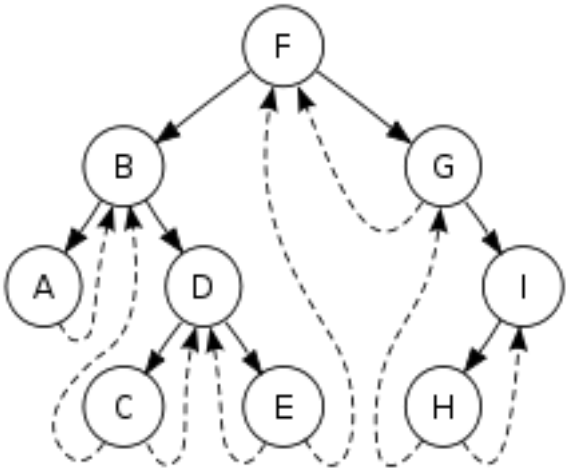
24. Considere uma árvore binária com n elementos. Mostre que $n + 1$ dos ponteiros nos nós da árvore são iguais a `NULL`.

25. Considere a seguinte implementação de árvores binárias:

```
typedef struct cel {
```

```
            elemento info;
            int efio, dfio;
            struct cel * esq;
            struct cel * dir;
        } no;
typedef no * apontador;
```

A ideia é utilizar os apontadores `NULL` (que são muitos, como vimos no último exercício) para apontar, por exemplo, para o sucessor/antecessor do nó em in-ordem. Neste caso, a variável `efio` (resp. `dfio`) teria valor 1 se o apontador `esq` (resp. `dir`) apontar para o antecessor (resp. sucessor) em in-ordem e 0 se for o filho esquerdo (resp. direito) do nó. A figura abaixo (extraída da wikipedia) mostra uma árvore de busca binária com fios (veja *threaded binary tree* na wikipedia).



Faça uma função que recebe uma árvore binária e coloca os fios conforme descrito acima.

26. Ao buscar o elemento 555 em uma árvore de busca binária é possível ter passado pelos seguintes nós? Justifique detalhadamente.
    a. 444, 333, 765, 513, 525, 555;
    b. 500, 700, 680, 515, 600, 535, 571, 566, 550, 555;
    c. 700, 340, 398, 610, 380, 412, 580, 555.

27. Considere a implementação de árvore que tem um apontador para o pai de cada nó. Escreva a função:

    `apontador insere (apontador raiz, elemento x)`

    que recebe uma árvore e devolve um apontador para a raiz da árvore em que `x` é inserido, **atualizando o apontador para o pai**.

28. Suponha que os elementos são inseridos em uma árvore de busca binária inicialmente vazia na seguinte ordem:

    $$50, 30, 70, 20, 40, 60, 80, 15, 25, 35, 45, 36$$

    Desenhe a árvore resultante. Em seguida, remova o nó de conteúdo 30.

29. Faça funções de protótipo

    `int menorQue (apontador raiz, elemento x)`

    `int maiorQue (apontador raiz, elemento x)`

    que devolve 1 se todos os elementos da árvore apontada por raiz forem menores (resp. maiores) que `x`.

30. Faça uma função de protótipo

    `int ehABB (apontador raiz)`

    que recebe um apontador para uma raiz de uma árvore binária e devolve 1 se a árvore é de busca binária e 0 caso contrário. Escreva versões iterativa e recursiva.