Recursividade

Marco A L Barbosa malbarbo.pro.br

Algumas considerações:

- Não utilize arranjos (a menos que seja pedido no exercício)
- Não utilize laços de repetição
- Use Lista, Natural e Natural1 como definidos em aula
- Lembre-se de descrever o que o algoritmo faz, isto é, descreva o relacionamento da entrada e da saída
- Faça exemplos concretos de entradas e saídas, isto vai te ajudar a projetar o algoritmo
- Não se preocupe que a soluções pareçam iguais, a ideia é aprender recursão por repetição!
- Você pode resolver os exercícios usando pseudocódigo, mas também é interessante fazer em uma linguagem de programação
- Talvez você não tenha tempo de resolver todos os exercícios, então escolha com sabedoria quais resolver
- E por fim, **Confie na recursão natural!**

Listas

1. Projete um algoritmo que receba como entrada uma Lista `lst` e conte quantos elementos `lst` tem.
2. Projete um algoritmo que receba como entrada uma Lista `lst` e verifique se todos os elementos de `lst` são pares.
3. Projete um algoritmo que receba como entrada uma Lista `lst` e um elemento `a` e conte quantas vezes `a` aparece em `lst`.
4. Projete um algoritmo que receba como entrada uma Lista `lst` e um elemento `a` e devolva uma Lista que é como `lst` mas sem as ocorrências dos valores maiores que `a`.
5. Projete um algoritmo que receba como entrada uma Lista `lst` e um elemento `a` e devolva uma Lista que é como `lst` mas com `a` no final.
6. Projete um algoritmo que receba como entrada uma Lista `lst` e devolva uma Lista com os mesmos elementos de `lst` mas em ordem reversa. Dica: combine o resultado da recursão natural com o primeiro elemento usando a função que insere no final.
7. Projete um algoritmo que receba como entrada duas Listas, `lst1` e `lst2`, e devolva uma nova lista contendo os elementos de `lst1` seguidos dos elementos de `lst2`. Dica: faça a recursão pensado em `lst1` e considere `lst2` como uma valor atômico (como o parâmetro `a` dos primeiros exercícios).
8. Projete um algoritmo que receba como entrada uma Lista `lst` e devolva uma Lista que é como `lst` mas com apenas uma ocorrência dos elementos repetidos consecutivos. Por exemplos, se `lst` for $\langle 3, 3, 3, 1, 5, 5, 1, 1, 1\rangle$ a saída deve ser $\langle 3, 1, 5, 1\rangle$.
9. Projete um algoritmo que verifique se os elementos de uma Lista estão em ordem não decrescente.
10. Defina um novo tipo chamado Lista1, que é como Lista, mas que não pode ser vazia, isto é, tem que ter pelo menos um elemento.

11. Projete um algoritmo que encontre o valor máximo de uma Lista1. (Em Python você pode verificar se um valor é de um tipo específico usando `isinstance`, por exemplo, para verificar se `a` é do tipo `int`, use `isinstance(a, int)`)

12. Projete um algoritmo que encontre o valor máximo de uma Lista não vazia, semelhante ao exercício anterior, mas agora a entrada é uma Lista e não uma Lista1. (Dica: assuma que a lista não pode ser `None` e mude o caso base).

13. Projete um algoritmo que receba como entrada uma Lista `lst` de números em ordem não decrescente e um número `n` e devolva uma Lista com os elementos de `lst` e com `n` em ordem não decrescente (ou seja, a função insere `n` na lista ordenada produzindo uma nova lista).

14. Projete um algoritmo que receba como entrada uma Lista de números e devolva uma lista com os mesmos valores de entrada mas em ordem não decrescente. (Aplique a ideia que estamos utilizando, não tente implementar um método de ordenação específico). Dica: combine o resultado da recursão natural com o primeiro elemento usando a função que insere ordenado.

Árvores

15. Faça a definição de um tipo para representar uma árvore binária. Uma árvore binária ou é vazia, ou é um nó com um valor inteiro e uma árvore binária esquerda e uma árvore binária direita.

16. Projete um algoritmo que receba como entrada uma árvore binária e conte a quantidade de nós na árvore.

17. Projete um algoritmo que receba como entrada uma árvore binária e um valor e verifique se o valor aparece na árvore (observe que estamos falando apenas de árvores binárias e não de árvores binárias de busca, ou seja, não existe nenhum restrição sobre como os valores estão espelhados pela árvore).

18. Uma árvore binária é de busca se cada nó x da árvore tem as seguintes propriedades:

    - O valor armazenado em x é maior os igual à todos os valores armazenados nos nós da árvore a esquerda; e
    - O valor armazenado em x é menor ou igual à todos os valores armazenados nos nós da árvore a direita

    Projete um algoritmo que receba como entrada uma árvore binária de busca e um valor e verifique se o valor aparece na árvore.

19. Projete um algoritmo que verifique se um árvore binária é de busca.

Naturais

20. Projete um algoritmo que receba como entrada um número qualquer a e um número natural n e calcule o valor a^n (use recursividade!).

21. Projete um algoritmo que receba como entrada dois números Natural1, `n` e `x`, e devolva uma Lista com os divisores de `x` que são menores ou iguais a `n` (não se preocupe com a ordem dos valores na resposta).

22. Um número Natural1 é perfeito se a soma dos seu divisores, exceto ele mesmo, é igual a ele. Por exemplo, o número 6 é perfeito porque $6 = 1 + 2 + 3$. Projete um algoritmo que verifique se um número Natural1 é perfeito. (Essa função não é recursiva! Use a função que produz a lista de divisores e a função que soma os valores de uma lista para implementar essa função)

23. Projete um algoritmo que receba como entrada um número Natural1 `n` e devolva uma lista com os número perfeitos menores ou iguais que `n` (não se preocupe com a ordem dos valores na resposta). Dica: use a função definida no exercício anterior para decidir se um número é perfeito.

24. Projete um algoritmo que receba como entrada dois números Natural1, `n` e `x` e devolva a quantidade de divisores de `x` que são menores ou iguais a `n`. Por exemplo, se a entrada for $x = 12$ e $n = 10$, a resposta tem que ser 5, porque 12 tem 5 divisores menores ou iguais a 10, que são $\langle 1, 2, 3, 4, 6\rangle$.

25. Um número Natural1 é primeiro se ele tem exatamente dois divisores distintos: 1 e ele mesmo. Projete um algoritmo que verifique se um número Natural1 é primo. (Essa função não é recursiva! Use a função que conta os divisores para implementar esse função)

26. Projete um algoritmo que receba como entrada um número Natural1 `n` e devolva uma lista com os números primos menores ou iguais a `n` (não se preocupe com a ordem dos valores na resposta). Dica: use a função definida no exercício anterior para decidir se um número é primo).

27. Projete uma função que receba como entrada um arranjo de números A (indexado a partir de 0) e um número Natural $n \leq A.\mathit{length}$ e calcule a soma $A[0] + A[1] + \cdots + A[n-1]$.

28. Projete uma função que receba como entrada um arranjo de números A (indexado a partir de 0) e dois números Naturais a e b, onde $a \leq b \leq A.\mathit{length}$, e calcule a soma $A[a] + A[a+1] + \cdots + A[b-1]$.

29. Projete uma função que receba como entrada um arranjo de números A (indexado a partir de 0) e dois números Naturais a e b, onde $a \leq b < A.\mathit{length}$, e verifique se o subarranjo $A[a], A[a+1], A[a+2], \cdots, A[b]$, é palíndromo, ou seja, os valores são os mesmos quando lidos direita para a esquerda ou da esquerda para a direita.

Desafios

30. Projete um algoritmo que recebe como entrada uma Lista `lst` e num número Natural `n` e produz uma nova lista com os primeiros `n` elementos de `lst` (você pode assumir que a quantidade de elementos de `lst` é menor ou igual a `n`). Dicas: faça o caso base pensado em `n` e decomponha `lst` e `n` na chamada recursiva.

31. Projete um algoritmo que recebe como entrada uma Lista `lst` e num número Natural `n` e produz uma nova lista sem os primeiros `n` elementos de `lst` (você pode assumir que `n` é menor ou igual a quantidade de elementos de `lst`). Dicas: faça o caso base pensado em `n` e decomponha `lst` e `n` na chamada recursiva.

32. Projete um algoritmo que receba como entrada duas Listas com os elementos em ordem não decrescente e produza uma nova lista com os mesmo elementos das duas listas de entrada em ordem não decrescente, ou seja, o algoritmo faz a intercalação das duas listas de entrada.

33. Projete um algoritmo que receba como entrada uma Lista de números e devolva uma lista com os mesmos valores de entrada mas em ordem não decrescente. Use a seguinte estratégia:

    - Se a lista for vazia, ordene ela trivialmente;
    - Senão, decomponha a lista de entrada em duas listas, `lst1` e `lst2` de maneira que a diferença da quantidade de elementos em cada uma das listas seja no máximo 1 (use os algoritmos dos dois primeiros desafios). Em seguida ordene cada uma das listas recursivamente e em seguinte intercale as listas ordenadas (usando o algoritmos do terceiro desafio).