ESTRUTURA DE DADOS - EXERCÍCIOS

O uso de funções recursivas pode facilitar a implementação de diversos algoritmos. Toda recursão depende de dois elementos: o caso base e o passo recursivo. Dentre as opções a seguir, a que apresenta um passo recursivo é:

- [x] *fat(n)=n*fat(n-1)*
- [ ] *fib(n)=n-1 + n-2*
- [ ] *fat(1)=1*
- [ ] *f(n)=g(n-1)*
- [ ] *par(n)=par(n)*

Respondido em 11/03/2023 15:58:54

Explicação:

O passo recursivo é o elemento que faz o cálculo da função recursiva mover-se em direção ao resultado. Deve envolver a chamada da própria função com um valor diferente de entrada. Isso só acontece na resposta correta: *fat(n)=n*fat(n-1)*, passo recursivo da função de cálculo de fatorial.

fat(1)=1 é o caso base dessa mesma função. *par(n)=par(n)* é uma tautologia, e não uma recursão. As demais respostas são funções que não chamam a si mesmas, não podendo ser passos recursivos.

Uma Lista pode ser implementada de forma contígua ou encadeada. No caso de uma lista ordenada implementada de forma encadeada, as complexidades de pior caso de busca, inserção e remoção são respectivamente:

- [ ] O(log n), O(n) e O(n).
- [x] O(n), O(n) e O(n).
- [ ] O(n), O(1) e O(n).
- [ ] O(1), O(n) e O(n).
- [ ] O(n), O(n) e O(1).

Respondido em 11/03/2023 16:04:04

Explicação:

A busca é O(n) pois no pior caso você terá que percorrer toda a lista sequencialmente até encontrar o último elemento. Já a inserção, no seu pior caso, colocará um elemento no final da lista, uma operação simples, mas a busca para achar a posição correta já é O(n). A remoção de qualquer nó também é uma operação de custo constante, bastando reapontar um ponteiro, mas a busca pelo nó a ser removido também é O(n), o que faz a operação de remoção também possuir complexidade O(n).

O método de ordenação da bolha, ou Bubblesort (BS) tem complexidade de pior caso O(n2) e melhor caso O(n). Suponha que exista um algoritmo de ordenação MS que tem complexidade de melhor caso O(nlog n) e de pior caso O(nlog n). Podemos afirmar que:

- [ ] Para um grande conjunto de entradas variadas de tamanho grande, BS executará em menos tempo que MS, em média.
- [ ] Para uma única entrada de tamanho grande, BS executará em menos tempo que MS.
- [x] Para um grande conjunto de entradas variadas de tamanho grande, MS executará em menos tempo que BS, em média.
- [ ] MS e BS são igualmente eficientes em ordenar elementos, independente da entrada ou seu tamanho.
- [ ] Para uma única entrada de tamanho grande, MS executará em menos tempo que BS.

O uso de funções recursivas pode facilitar a implementação de diversos algoritmos. Toda recursão depende de dois elementos: o caso base e o passo recursivo. Dentre as opções a seguir, a que apresenta um passo recursivo é:

- [ ] *fib(n)=n-1 + n-2*
- [ ] *f(n)=g(n-1)*
- [x] *fat(n)=n*fat(n-1)*
- [ ] *par(n)=par(n)*
- [ ] *fat(1)=1*

Em Python é possível implementar um array utilizando o tipo padrão list. Essa implementação permite o uso das seguintes funções para inserir e remover um elemento, respectivamente:

- [ ] insert, delete/pop.
- [ ] ✔ append, remove/pop.
- [x] ✘ append, pop/delete.
- [ ] impose, remove/destroy.
- [ ] insert, remove/destroy.

Suponha que você está implementando um programa que precisa armazenar dados ordenados em uma estrutura para serem tratados posteriormente, na ordem em que foram recebidos. Haverá uma grande quantidade de recebimentos e tratamento de dados, mas o tamanho esperado da estrutura não deve variar muito. Qual tipo de estrutura de dado é a melhor nessa situação?

- [ ] Lista simplesmente encadeada.
- [ ] Lista em alocação contígua.
- [x] ✔ Fila.
- [ ] Lista duplamente encadeada.
- [ ] Pilha.

	Endereço	Chave	Próximo
	128	5	64
	64	8	32
	32	11	null
L----->	24	3	128

- [ ] O conteúdo armazenado no endereço 32 será apagado.
- [ ] L terá sido apagada.
- [ ] O endereço 24 conterá a chave 5 e próximo 64.
- [x] ✘ O endereço 32 terá seu campo próximo apontando para 24.
- [ ] ✔ A variável L apontará para 128.

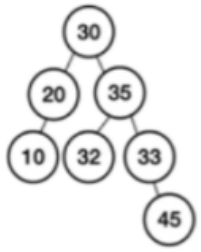
Uma Lista pode ser implementada de forma contígua ou encadeada. No caso de uma lista implementada de forma contígua, as complexidades de pior caso de busca, inserção e remoção são respectivamente:

- [x] ✔ O(n), O(n) e O(n).
- [ ] O(n), O(1) e O(n).
- [ ] O(n), O(n) e O(1).
- [ ] O(1), O(n) e O(n).
- [ ] O(log n), O(n) e O(n).

Marque a opção **correta** acerca da visita *In-Ordem*, usada em percursos de árvores binárias de busca:

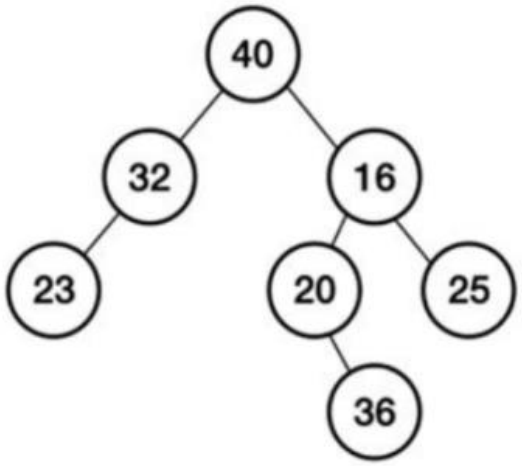
- [ ] O percurso *in-ordem* inicia a visita pela raiz da árvore, depois visita as sub-árvores a esquerda e depois as sub-árvores à direita em ordem simétrica, terminando pela raiz principal.
- [ ] O percurso *in-ordem* inicia a visita pela raiz da árvore, depois visita as sub-árvores a esquerda e depois as sub-árvores à direita em ordem assimétrica.
- [x] ✔ O percurso *in-ordem* inicia a visita nas sub-árvores a esquerda em ordem simétrica, depois visita a raiz da árvore e depois as sub-árvores à direita em ordem simétrica.
- [ ] O percurso *in-ordem* inicia a visita pela raiz da árvore, depois visita as sub-árvores a esquerda e depois as sub-árvores à direita em ordem simétrica.
- [ ] O percurso *in-ordem* inicia a visita nas sub-árvores a direita, depois visita a raiz da árvore e depois as sub-árvores à esquerda em ordem simétrica.

Seja a seguinte árvore binária. Analise as afirmativas e marque a correta.



- [ ] A árvore acima possui 4 nós folhas.
- [x] A figura ilustra uma árvore binária de busca porque para todos os nós vale a seguinte propriedade: os nós contidos na subárvores esquerda são menores que a raiz e os contidos na subárvore direita são maiores.
- [ ] É possível inserir mais um nó filho ao nó 30.
- [ ] Ao buscar pelo nó 45 na árvore acima, um algoritmo de busca em Python irá realizar sempre *O(n)* passos. Isto ocorre uma vez que é necessário analisar todos os nós da árvore para encontrar o nó 45.
- [ ] Supondo que a árvore em questão é uma árvore binária de busca, a inserção de um novo nó com chave 47 pode ser feita em qualquer subárvore vazia.

Considere a seguinte estrutura de árvore. Marque a alternativa correta:



- [x] Pode-se afirmar que o número de passos para buscar um elemento na árvore acima é, no máximo, O(log n).
- [ ] A árvore da figura se trata de árvore binária de busca.
- [ ] Remover o nó 36 irá deixar a árvore com as propriedades de árvore binária de busca.
- [ ] A árvore acima é conhecida como árvore zig zag balanceada.
- [ ] É necessário executar $O(n^2)$ passos para deletar um elemento da árvore acima.

Árvores de busca são organizadas de forma hierárquica, onde cada nó é um objeto que contém informação e cada nó filho é uma subdivisão da informação contida no nó pai. Sobre árvores binárias de busca, marque a opção correta.

- [ ] As operações de busca, inserção e remoção são exatamente O(n log n).
- [ ] Ao se construir uma árvore binária de busca com a sequência 8-13-10-2-5-15-4-7 o número de folhas raiz é igual a 3.
- [ ] A melhor árvore binária que podemos construir tem altura proporcional a n, onde n é a quantidade de nós da árvore.
- [ ] Em toda árvore binária de busca as inserções correm em O(log n) e as remoções em O(n), independentemente da altura da árvore.
- [x] Seja T uma árvore binária completa com n > 0 nós. Então T possui altura mínima e h = 1 + [log n].

Explicação:

As operações de remoção e inserção tem complexidade assintótica proporcional à altura da árvore, assim, se a árvore é balanceada ocorrerá em O(log n). Toda árvore de busca e que está balanceada tem altura proporcional a log n. Logo, a complexidade das operações busca, inserção e remoção são exatamente O(log n). PA melhor árvore binária de busca é a balanceada.Temos 4 folhas.

Em P**ython é possível implem**entar um **ar**ray **uti**lizando o tipo padrão list. Essa implementação permite o uso das seguintes funções para inserir e remover um elemento, respectivamente:

- [x] append, remove/pop.
- [ ] insert, remove/destroy.
- [ ] insert, delete/pop.
- [ ] append, pop/delete.
- [ ] impose, remove/destroy.

Ao usar a biblioteca numpy para criar arra**ys,** existem diversas facilidades que um programador pode utilizar, como funções específicas para somar todos os elementos, encontrar valores mínimo e máximo dos elementos, entre outros.

Entretanto uma desvantagem de usar array da biblioteca numpy é:

- [ ] Não é possível remover elementos do array.
- [ ] Diminuição no tempo de programação.
- [ ] Não é possível adicionar novos elementos ao array.
- [ ] Os índices passam a ser contados a partir de 1.
- [x] Todos os elementos devem ter o mesmo tamanho.

O método de ordenação da bolha, ou Bubblesort tem como melhor caso a entrada já ordenada, que resulta em complexidade O(n). Como seu pior caso, a entrada em ordem invertida, resultando em complexidade $O(n^2)$. Baseado nessas duas afirmações, podemos afirmar que a sua complexidade de caso médio é:

- [ ] O(n)
- [ ] O(nlog n)
- [ ] O(1)
- [ ] O(log n)
- [x] $O(n^2)$

Em uma implementação da estrutura de dados do tipo fila, você possui um espaço de memória contíguo a ela alocada com capacidade para M nós. A variável da fila é F, e duas variáveis guardam os índices do início e final da fila (inicioF e finalF). Em uma implementação otimizada de F, como podemos identificar que a fila está cheia?

- [ ] InicioF = M
- [ ] InicioF==finalF + 1
- [ ] FinalF== M
- [ ] InicioF== finalF
- [x] InicioF==(finalF+1)mod M

Suponha que você está implementando um programa que precisa armazenar dados ordenados em uma estrutura para serem tratados posteriormente, na ordem inversa à que foram recebidos. Haverá uma grande quantidade de recebimentos e tratamento de dados, mas o tamanho esperado da estrutura não deve variar muito. Qual tipo de estrutura de dados é a melhor nessa situação?

- [ ] Lista duplamente encadeada.
- [ ] Lista em alocação contígua.
- [ ] Fila.
- [x] Pilha.
- [ ] Lista simplesmente encadeada.

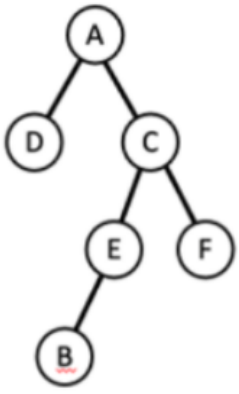
Uma Pilha é uma estrutura de dados que permite o armazenamento de elementos (ou nós) sequencialmente. Sobre as Pilhas é possível afirmar que:

- [x] Permitem inserção ou remoção apenas no seu início.
- [ ] Permitem inserção ou remoção apenas no seu início ou no seu final.
- [ ] Permitem inserção no seu final e remoção apenas no seu início.
- [ ] Permitem inserção ou remoção em qualquer de suas posições.
- [ ] Permitem inserção no seu início e remoção apenas no seu final.

As árvores de busca são estruturas de dados que armazenam elementos de forma hierárquica, permitindo uma busca eficiente em grandes conjuntos de dados. Marque a opção **correta** acerca das estruturas de dados Árvores e Árvores Binárias:

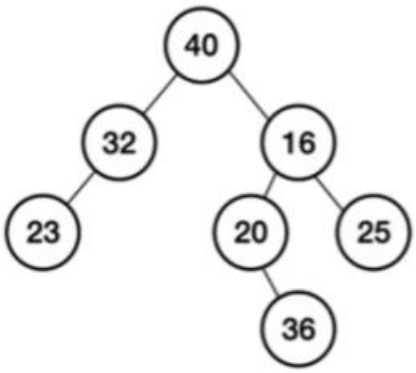
- [ ] Nas Árvores Binárias de Busca cada nó deve ter exatamente 2 filhos.
- [ ] As folhas estão sempre no nível 1 da árvore.
- [x] Ao acessar uma árvore, deve-se acessar pela referência a sua raiz.

Seja a seguinte árvore binária de busca, marque a opção que apresenta o percurso em pré-ordem dessa árvore:



- [ ] D,B,E,F,C,A
- [ ] A,B,C,D,E,F
- [ ] F,C,E,B,A,D
- [x] A,D,C,E,B,F
- [ ] D,A,B,E,C,F

Considere a seguinte estrutura de árvore. Marque a alternativa correta:



- [ ] A árvore acima é conhecida como árvore zig zag balanceada.
- [ ] A árvore da figura se trata de árvore binária de busca.
- [ ] É necessário executar $O(n^2)$ passos para deletar um elemento da árvore acima.
- [ ] Remover o nó 36 irá deixar a árvore com as propriedades de árvore binária de busca.
- [x] Pode-se afirmar que o número de passos para buscar um elemento na árvore acima é, no máximo, O(log n).

Respondido em 02/05/2023 20:52:13

Explicação:

Apesar de que a árvore considerada não é árvore binária de busca, observa-se que os nós estão dispostos de forma balanceada e em níveis, portanto o número de passos para buscar um elemento na árvore acima é, no máximo, O(log n).

Em uma Árvore B, temos que: Cada nó contém no mínimo m registros (m+1 descendentes) e no máximo 2m registros (e 2m+1 descendentes), exceto o nó que é raiz que pode conter entre 1 e 2m registros e todos os nós folhas aparecem no mesmo nível. Sobre Árvores B, é correto afirmar:

- [x] O particionamento de nós em uma Árvore B ocorre quando um registro precisa ser inserido em um nó com 2m registros.
- [ ] O particionamento de nós ocorre quando é necessário diminuir a altura da árvore.
- [ ] O particionamento de nós em uma Árvore B ocorre quando um registro precisa ser buscado em um nó com 2m + 1 registros.
- [ ] O particionamento de nós em uma Árvore B ocorre quando um registro precisa ser inserido em um nó com menos de 2m registros.
- [ ] O particionamento de nós em uma Árvore B ocorre quando a chave do registro a ser inserido contém um valor(conteúdo) intermediário entre os valores das chaves dos registros contidos no mesmo nó.

As operações de busca, remoção e inserção de nós em uma **árvore binária de busca** levam determinado tempo de execução de seus algoritmos. Esses tempos são dados pela alternativa:

- [x] Busca: O(n) / Remoção: O(n) / Inserção: O(n)
- [ ] Busca: O(1) / Remoção: O(log n) / Inserção: O(log n)
- [ ] Busca: O(n) / Remoção: O(log n) / Inserção: O(log n)
- [ ] Busca: O(log n) / Remoção: O(n) / Inserção: O(log n)
- [ ] Busca: O(n) / Remoção: O(n) / Inserção: O(log n)

Respondido em 14/09/2023 21:56:33

Explicação:

No pior caso uma árvore binária de busca com *n* chaves tem *n* níveis. Assim, o pior caso da busca, é buscar o nó mais profundo da árvore que demandará *n* comparações. Como a busca é subrotina da inserção e da remoção, então as três operações terão complexidade de pior caso de O(n).

Dada a seguinte matriz M, inicializada com o código:

M=[[1,2,3,4],[5,6,7,8],[9,10,11,12],[13,14,15,16]]

O código em Python para imprimir cada elemento da coluna iniciada pelo elemento 3 é:

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

- ☐
```
for coluna in M:
    print(coluna)
```
- ☒ ✔
```
for linha in M:
    print(linha[2])
```
- ☐
```
print(M[2])
```
- ☐
```
for linha in M:
    print(linha[3])
```
- ☐
```
for linha in M:
    print(linha)
```

Respondido em 13/09/2023 21:13:44

Explicação:

O laço deve percorrer uma coluna, iterando linha a linha e extraindo dela o seu terceiro elemento, ou seja linha[2]. A resposta correta itera pelas linhas e imprime o elemento [2] de cada uma.

Dentre as respostas erradas, apenas escrever ¿print(linha)¿ imprimirá cada linha como um todo, resultando na impressão de toda a matriz, linha a linha.

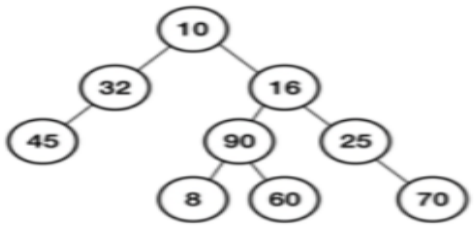
A resposta "print(coluna)" terá o mesmo resultado pois para o código linha e coluna são apenas nomes escolhidos pelo programador. Poderia ser i, aux ou qualquer outra variável escolhida.

Já "print(linha[3])" está com o índice errado, imprimindo os elementos da coluna iniciada por 4. E ¿print(M[2])¿ imprime toda a linha iniciada por 9.

Uma lista circular é uma estrutura de dados contínua, permitindo que seja iterada sobre ela de forma infinita. Uma das suas aplicações em jogos digitais é:

- ☐ Em jogos mobile, para armazenar o número do telefone do jogador.
- ☐ Em jogos competitivos, para garantir que não há scripts ou bots rodando no computador.
- ☒ ✔ Em jogos multijogador em turnos, permitindo ceder o controle a um jogador por vez.
- ☐ Em jogos multijogador para garantir que apenas um dos jogadores jogue todas as vezes.
- ☐ Em jogos de um jogador para armazenar um conjunto fixo de elementos.

Seja a seguinte Árvore Binária. Marque a opção correta:



- ☐ A árvore acima possui raiz de valor 3.
- ☒ ✔ A quantidade de folhas da árvore é 4.
- ☐ A quantidade de nós da árvore é de n ¿ 1, sem considerar o nó raiz.
- ☐ Não é possível inserir nós filhos ao nó 70.
- ☐ É possível inserir mais um filho a esquerda no nó de valor 90.

Explicação:

A quantidade de folhas da árvore é 4, ou seja, são aqueles nós que possuem grau zero.