Dredd - Juiz Online

Minutos Restantes: 306

Usuário: Gabriel Nathan Almeida Silva

Notas:
Q1: ?
Q2: ?
Q3: ?
Q4: ?
Q5: 80
Q6: ?
Q7: ?
Q8: ?
Q9: ?
Q10: 36
Q11: ?
Q12: ?
Total: 10

REO3 - ABB, AVL e Rubro Negra

Prova Aberta Até: 13/07/2020 23:59:59

Número Máximo de Tentativas: 4

Atenuação da Nota por Tentativa: 0%

Instruções para a prova: A prova é individual. Desligue seu celular. Não converse com os colegas. Não fique olhando para a tela dos colegas.

Questão 1: Árvore ABB - Nó sem pai

Em algumas implementações de árvores binárias de busca, os nós têm encadeamento duplo (para baixo e para cima) enquanto que em outras eles tem encadeamento simples (só para baixo). A primeira estratégia simplifica a implementação de alguns métodos enquanto a segunda economiza memória.

Complete o código fornecido para produzir uma implementação **com encadeamento simples**. A implementação deve possuir as operações de inserção e busca. Se quiser, implemente uma operação de escrever a árvore para facilitar a depuração do programa. Ela não será usada na avaliação do programa.

A função principal foi feita supondo o uso de manipulação de exceções. Você pode alterar essa estratégia, desde que a operação de escrita da mensagem de erro continue na função principal e continue sendo realizada na saída padrão.

Entradas:

1. i - para **inserir** uma informação na árvore. A letra i deve ser seguida da chave (número inteiro) e valor (palavra).
2. b - para **buscar** uma informação na árvore. A letra b deve ser seguida da chave a ser buscada.
3. e - para escrever o conteúdo da árvore, em ordem.
4. f - para finalizar a execução do programa.

Saídas:

A operação de inserção não produz saída. A operação de busca escreve o valor associado à chave. Caso a informação buscada não esteja na árvore, a função principal escreve "INEXISTENTE" no lugar do valor. A operação de escrever não tem validade na avaliação.

Exemplo de Entrada:

```
i 4 quatro
i 2 dois
i 8 oito
i 6 seis
b 2
b 6
b 7
f
```

Exemplo de Saída:

```
dois
seis
INEXISTENTE
```

Peso: 1

Nova Resposta:

Selecione o arquivo com o código fonte do programa que resolve o problema para enviá-lo.

Escolher arquivo | Nenhum arquivo selecionado | Enviar Resposta

Questão 2: Implementação de uma Árvore Binária de Busca (ABB) sem nó

Existem muitas formas de implementar uma estrutura de dados qualquer. A linguagem de programação usada é um dos fatores que mais influência as características de implementação. As linguagens que escondem ponteiros e usam *coleta de lixo* para gerenciar o uso da memória favorecem a implementação de árvores sem a classe auxiliar "nó".

Linguagens sem *coleta de lixo* não favorecem esse tipo de implementação, mas isso não quer dizer que não podemos fazer uma.

Implemente uma ABB sem usar uma classe auxiliar. Para agilizar seu desenvolvimento, use este programa que já tem atributos e métodos projetados, além de um programa com interface para inserir, buscar, remover e escrever a árvore. Os métodos auxiliares no código são sugestão de implementação. O programa lê comandos numéricos e seus argumentos para as várias operações possíveis.

Entradas:

Cada comando é um número inteiro identificando o comando seguido dos parâmetros necessários para executar o comando, se houverem. Os códigos de comandos são:

- O número 0 para encerrar a execução do programa.
- O número 1 para inserir chave (número inteiro) e valor (número inteiro) na árvore

- O número 2 para remover dado da árvore, seguido da chave (número inteiro) que deve ser removida.
- O número 3 para buscar na árvore, seguido da chave consultada. Este comando produz uma saída que é o valor associado.
- O número 4 para escrever todas chaves e seus respectivos valores num formato de texto com parênteses.
- O número 5 para escrever os nós da árvore nível a nível.

Saídas:

Cada comando tem sua saída específica. Veja os exemplos abaixo.

Exemplo de entrada e saída juntos:

```
1 5 50
1 3 30
1 2 20
4
(5/50 (3/30 (2/20 () ()) ()) ())
1 6 60
5
[5/50]
[3/30][6/60]
[2/20][][][]
[][]
2 5
5
[6/60]
[3/30][]
[2/20][]
[][]
2 4
Impossível remover. A chave não existe.
2 3
4
(6/60 (2/20 () ()) ())
2 2
2 6
5
[]
0
```


Peso: 1

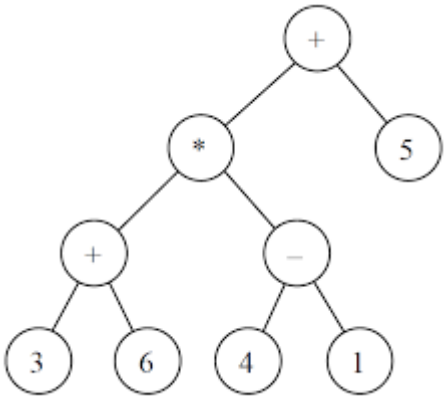
Nova Resposta:

Selecione o arquivo com o código fonte do programa que resolve o problema para enviá-lo.

Escolher arquivo | Nenhum arquivo selecionado | Enviar Resposta

Questão 3: Árvores Binárias - Resolução de Expressões em Notação Prefixa

Em várias áreas da computação, árvores são utilizadas para resolver vários problemas e aplicadas de forma eficiente para estruturação de "registros" seguindo critérios específicos. Entre as várias aplicações de árvores binárias, encontram-se as árvores de expressões. Em uma árvore de expressões, expressões lógicas ou aritméticas podem ser representadas sem ambiguidade e sem a necessidade de uso de parênteses. Assim, a imagem apresentada ilustra, sem ambiguidade e sem parênteses, a expressão `((3+6)*(4-1))+5`. Computacionalmente, isso é importante, principalmente no desenvolvimento de compiladores e interpretadores e está relacionado ao uso de notações prefixas ou posfixas de expressões.



Em uma representação prefixa, também chamada de notação polonesa, as expressões são representadas com operadores vindo antes dos operandos. Por exemplo, a expressão `(a+b)` seria representada como `+ a b`. O termo notação polonesa deve-se à sua criação pelo estudioso de lógica polonês Jan Łukasiewicz. Perceba que a notação polonesa corresponde justamente ao percorrimento prefixo da árvore de expressões. Perceba também, que na árvore de expressões, os operadores são nós intermediários e os valores ou variáveis numéricas são nós folhas.

Seu objetivo é escrever um programa que aceite uma expressão aritmética escrita em notação prefixa e construir a árvore de expressão associada, efetuando o cálculo da expressão usando o percorrimento adequado à tarefa. **Dica:** use `stoi()` para converter strings para valores inteiros.

Entradas:

1. Expressão aritmética em notação polonesa, composta pelos caracteres: +, -, *, /, ^, em que ^ indica potenciação e os outros operadores indicam as mesmas operações presentes em C++. Considere que a expressão informada sempre será válida.

Saídas:

1. Resultado da operação após manipulação da árvore. Em casos de operações inválidas, como divisão por zero, deverá ser escrito **invalido** na saída padrão.

Exemplo de Entrada:

```
+ * + 3 6 - 4 1 5
```

Exemplo de Saída:

```
32
```

Exemplo de Entrada:

```
- * 2 / 8 4 3
```

Exemplo de Saída:

```
1
```

Exemplo de Entrada:

```
+ ^ 3 2 / * 2 2 - 4 3
```

Exemplo de Saída:

```
13
```

Exemplo de Entrada:

```
+ ^ 3 2 / * 2 2 - 4 4
```

Exemplo de Saída:

```
invalido
```

Peso: 1

Nova Resposta:

Selecione o arquivo com o código fonte do programa que resolve o problema para enviá-lo.

Escolher arquivo Nenhum arquivo selecionado Enviar Resposta

Questão 4: Árvore Binária com Contadores

Implemente a árvore binária com contadores, de forma que ela permita elementos repetidos, mas que as repetições sejam armazenadas no próprio nó. Ou seja, cada nó conterá, além do valor, um contador para informar quantos elementos daquele valor estão armazenados. Ao inserir um elemento na árvore, caso já exista um nó com esse valor, então o contador é incrementado. O processo de remoção, por sua vez, irá decrementar o contador, só removendo o nó quando o valor for totalmente excluído da árvore. Você deverá implementar os métodos de inserção, remoção e de percorrimento. A função de percorrimento

deverá imprimir, logo após o valor, a quantidade de elementos armazenados naquele nó. Por exemplo, a representação 1(4)/0 indica que o valor 1 possui 4 elementos armazenados e encontra-se no nível 0 da árvore.

Entradas:

1. 10 valores a serem inseridos na árvore
2. 5 valores a serem removidas da árvore
3. 10 valores a serem inseridos na árvore
4. 5 valores a serem removidas da árvore

Saídas:

1. Percorrimento da árvore após 10 inserções
2. Percorrimento da árvore após 5 remoções
3. Percorrimento da árvore após 10 inserções
4. Percorrimento da árvore após 5 remoções

Exemplo de Entrada:

```
1 1 1 1 2 2 2 3 3 3
1 1 2 2 2
4 4 5 5 5 2 2 6 6 6
5 5 6 1 1
```

Exemplo de Saída:

```
1(4)/0 2(3)/1 3(3)/2
1(2)/0 3(3)/1
1(2)/0 2(2)/2 3(3)/1 4(2)/2 5(3)/3 6(3)/4
2(2)/1 3(3)/0 4(2)/1 5(1)/2 6(2)/3
```

Exemplo de Entrada:

```
1 1 1 1 2 2 2 3 3 3
1 1 2 2 3
1 1 1 1 1 2 2 2 2 2
3 3 1 1 1
```

Exemplo de Saída:

```
1(4)/0 2(3)/1 3(3)/2
1(2)/0 2(1)/1 3(2)/2
1(7)/0 2(6)/1 3(2)/2
1(4)/0 2(6)/1
```

Peso: 1

Nova Resposta:

Selecione o arquivo com o código fonte do programa que resolve o problema para enviá-lo.

Escolher arquivo | Nenhum arquivo selecionado | Enviar Resposta

Minutos Restantes: 306

Usuário: Gabriel Nathan Almeida Silva

Notas:
Q1: ?
Q2: ?
Q3: ?
Q4: ?
Q5: 80
Q6: ?
Q7: ?
Q8: ?
Q9: ?
Q10: 36
Q11: ?
Q12: ?
Total: 10

Questão 5: Árvore binária (ABB) - Contagem de nós

Faça um programa que calcula a quantidade de nós em uma árvore binária. Seu programa também deve exibir a diferença de nós da sub-árvore esquerda com a sub-árvore direita da raiz (esquerda menos direita). A leitura dos valores deve ser feita até que o valor -1 seja lido (ele representará que a entrada acabou).

A quantidade de nós numa árvore vazia é zero.

Não é permitido usar atributos que indiquem quantidade de nós na árvore e nem no nó.

Entradas:

1. Valores (inteiros) a inserir na árvore.

Saídas:

1. Quantidade de nós da árvore.
2. Diferença de nós da sub-árvore esquerda com a sub-árvore direita da raiz.

Exemplo de Entrada:

```
45 30 10 69 5 187 100 50 25 2 -1
```

Exemplo de Saída:

```
10 1
```

Peso: 1

Última tentativa realizada em: 12/07/2020 23:19:33

Tentativas: 2 de 4

Nota (0 a 100): 80

Status ou Justificativa de Nota: O programa não resolve todas as instâncias do problema.

Ver Código da Última Tentativa

Nova Resposta:

Selecione o arquivo com o código fonte do programa que resolve o problema para enviá-lo.

Escolher arquivo Nenhum arquivo selecionado Enviar Resposta

Minutos Restantes: 306

Usuário: Gabriel Nathan Almeida Silva

Notas:
Q1: ?
Q2: ?
Q3: ?
Q4: ?
Q5: 80
Q6: ?
Q7: ?
Q8: ?
Q9: ?
Q10: 36
Q11: ?
Q12: ?
Total: 10

Questão 6: Árvore ABB - Eficiência da busca

A árvore binária de busca serve para reduzir os passos necessários na recuperação de informações de um conjunto grande. Para investigar essa eficiência você vai implementar uma árvore com as operações de inserção e busca, de tal forma que seja possível saber o número de nós visitados durante uma busca na árvore.

Este código já tem um começo de implementação, em que a função principal já está pronta, e também está implementado um tipo de dados que o programa vai usar para armazenar na árvore. Esse tipo consiste de uma chave (número inteiro) e um valor (texto). Você pode alterar a função principal se decidir que não deseja usar manipulação de exceções, porém o programa deverá continuar funcionando do mesmo jeito.

Entradas:

O programa consiste em um laço com as opções de inserção, busca, escrita em ordem e encerramento. A operação de escrita não é importante na nota. Se você decidir implementar, adicione o comando para escrever na função principal. As opções são identificados por letras seguidas de parâmetros, a saber:

1. i - para **inserir** uma informação na árvore. A letra i deve ser seguida da chave (número inteiro) e valor (palavra).
2. b - para **buscar** uma informação na árvore. A letra b deve ser seguida da chave a ser buscada.
3. e - para escrever o conteúdo da árvore, em ordem.
4. f - para finalizar a execução do programa.

Saídas:

A operação de inserção não produz saída. A operação de busca escreve o valor associado à chave, seguida do número de nós visitados na árvore durante a busca. Caso a informação busca não esteja na árvore, a função principal escreve "INEXISTENTE" no lugar do valor. A operação de escrever escreve todos os valores da árvore e **não tem influência na avaliação**.

Exemplo de Entrada:

```
i 2 dois
i 5 cinco
i 6 seis
i 4 quatro
i 1 um
i 3 tres
b 1
b 3
b 7
b 2
f
```

Exemplo de Saída:

```
um 2
tres 4
INEXISTENTE 3
dois 1
```

Peso: 1

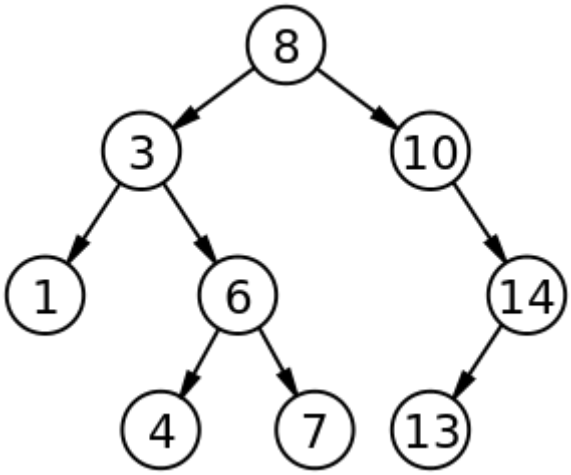
Nova Resposta:

Selecione o arquivo com o código fonte do programa que resolve o problema para enviá-lo.

Escolher arquivo | Nenhum arquivo selecionado | Enviar Resposta

Questão 7: Encontre o sucessor em uma Árvore Binária de Busca (ABB)

Implemente um método para encontrar o sucessor de uma chave numa árvore binária. Suponha que a chave pertence à árvore. É importante que haja um mínimo de eficiência: buscar a chave (já pronto) e depois caminhar até o sucessor, sem processar outros nós.



Use esta implementação que já tem pronto método para escrever a árvore, além de uma interface que ativa os métodos implementados. A interface não deve ser alterada.

Antes de implementar, planeje. Note que um nó que não tem filho à direita tem um sucessor que não está numa subárvore. Na árvore da figura, por exemplo, o sucessor de 7 está na raiz.

Entradas:

O programa lê comandos e os executa. Os sucessores são encontrados chamando repetidamente o método que encontra um sucessor. Basta usar a interface providenciada. Os comandos são:

- f - para finalizar a execução do programa
- i - para inserir uma chave, seguido da chave (inteiro)
- e - para escrever o conteúdo da árvore, nível a nível para facilitar o entendimento da estrutura
- s - para escrever todos os sucessores de uma chave, seguido da chave (inteiro)

Saídas:

Cada comando produz uma saída específica. Apenas mantenha as saídas já implementadas.

Exemplo de entrada e saída juntos:

```
i 300
i 20
i 430
i 12
i 360
i 451
s 20
300 360 430 451
f
```

Peso: 1

Nova Resposta:

Selecione o arquivo com o código fonte do programa que resolve o problema para enviá-lo.

Escolher arquivo Nenhum arquivo selecionado Enviar Resposta

Questão 8: Contagem dos nós folha de uma Árvore Binária de Busca (ABB)

A partir do código fornecido, implemente os métodos que calculam a quantidade de nós folhas em uma árvore binária. Não é necessário manter o projeto do programa exemplo. Métodos auxiliares podem ser criados. Métodos da contagem de folhas que não forem usados devem ser apagados.

A quantidade de folhas numa árvore vazia é zero.

Entradas:

1. Número de valores a serem lidos.
2. Valores (inteiros) a inserir na árvore.

Saídas:

- Quantidade de nós folha.

Exemplo de Entrada:

```
8
30 15 50 41 10 60 32 45
```

Exemplo de Saída:

```
4
```

Peso: 1

Nova Resposta:

Selecione o arquivo com o código fonte do programa que resolve o problema para enviá-lo.

Escolher arquivo Nenhum arquivo selecionado Enviar Resposta

Questão 9: Estrutura de dados AVL recursiva com números - Biblioteca

Uma biblioteca comunitária decidiu reorganizar seus livros e renovar seu sistema de consulta, facilitando a busca por títulos. Para isso, cada livro será registrado com seu nome, localização e quantidade de itens disponíveis para empréstimo. A cada empréstimo ou devolução, a quantidade de itens disponíveis para empréstimo é atualizada. Utilizando o código fornecido, implemente um árvore AVL com as funções de inserir e remover elementos. A idea principal do problema é que implementação da remoção seja utilizando recursão, não sendo necessário o uso da função transplanta. A estratégia de remoção de elementos que não estejam em nohs folhas é a substituição pelo elemento posterior (sucessor).

O código fornecido já tem o menu de entrada e o método de impressão. **Não** altere essas funções. O caminho mais fácil para a solução do exercício é não alterar a assinaturas das funções, dessa forma, não resultará em conflito com o código fornecido. O dado a ser armazenado na árvore é composto por uma **chave** do tipo *inteiro*, o **nome do livro** do tipo *string*, a **localização** do tipo *string* e a **quantidade de itens** disponíveis do tipo **inteiro**. Não existem livros diferentes com a mesma chave.

Entradas:

1. i - para **inserir** elemento : deve ler chave [inteiro] , nome do livro [string], localização [string] e quantidade disponível [inteiro]
2. r - para **remover** elemento : deve ler chave [inteiro]
3. b - para **buscar** elemento : deve ler chave [inteiro]
4. l - para fazer o **levantamento** de livros em uma dada localização: deve ler uma string
5. e - para **imprimir** árvore
6. f - finalizar

Saídas:

1. i - inserir elemento : A operação de inserção **não** produz saída
2. r - remover elemento : A operação de remoção **não** produz saída
3. b - buscar elemento : A operação de busca escreve a frase: "Elemento buscado: ([chave],[nome],[local],[quantidade])", em que [chave], [nome], [local] e [quantidade] devem ser

substituídos pela respectiva chave, nome do livro, localização e quantidade disponível."

4. l - levantamento por localização: Imprime: "Levantamento do local [local]: [quantidade]", em que [local] é o local pesquisado e [quantidade] é a quantidade de elementos na dada localização.
5. e - imprimir árvore : imprime a árvore seguindo o padrão de formatação tree
6. f - finalizar : finaliza o programa

Exemplo de Entrada:

```
i 5 Memorias_Postumas_de_Bras_Cubas C6E01 3
i 7 Capitaes_de_Areia C4E02 5
i 9 Luciola C6E05 2
i 6 Dom_Casmurro C3E01 4
i 11 Quincas_Borba C3E01 3
i 42 O_Guia_do_Mochileiro_das_Galaxias C7E42 10
i 43 O_Restaurante_no_Fim_do_Universo C7E42 8
i 44 A_Vida_o_Universo_e_Tudo_Mais C7E42 7
i 45 Ate_mais_e_Obrigado_pelos_Peixes C7E42 6
i 46 Praticamente_Inofensiva C7E42 6
e
l C7E42
l C3E01
f
```

Exemplo de Saída:

```
(7,Capitaes_de_Areia)
 ├e─(5,Memorias_Postumas_de_Bras_Cubas)
 │   └d─(6,Dom_Casmurro)
 └d─(43,O_Restaurante_no_Fim_do_Universo)
     ├e─(11,Quincas_Borba)
     │   ├e─(9,Luciola)
     │   └d─(42,O_Guia_do_Mochileiro_das_Galaxias)
     └d─(45,Ate_mais_e_Obrigado_pelos_Peixes)
         ├e─(44,A_Vida_o_Universo_e_Tudo_Mais)
         └d─(46,Praticamente_Inofensiva)
Levantamento do local C7E42: 37
Levantamento do local C3E01: 7
```

Exemplo de Entrada 2:

```
i 5 Memorias_Postumas_de_Bras_Cubas C6E01 3
i 7 Capitaes_de_Areia C4E02 5
i 9 Luciola C6E05 2
i 6 Dom_Casmurro C3E01 4
i 14 Pedro_e_o_Lobo C6E05 8
l C6E05
b 9
r 9
r 9
b 9
e
f
```

Exemplo de Saída 2:

```
Levantamento do local C6E05: 10
Elemento buscado: (9,Luciola,C6E05,2)
Erro na remoção: chave não encontrada!
Erro na busca: elemento não encontrado!
(7,Capitaes_de_Areia)
 ├e─(5,Memorias_Postumas_de_Bras_Cubas)
 │   └d─(6,Dom_Casmurro)
 └d─(14,Pedro_e_o_Lobo)
```

Observação: para baixar o arquivo, clique com o botão direito do mouse, depois clique em "salvar link como" e escolha uma pasta do seu computador. Não tente abrir o código no navegador e copiar o texto. Isso resultará em mudança na codificação do texto de UTF-8 para ISO, deformando a saída formatada da árvore.

Peso: 1

Nova Resposta:

Selecione o arquivo com o código fonte do programa que resolve o problema para enviá-lo.

Escolher arquivo | Nenhum arquivo selecionado | Enviar Resposta

Questão 10: Estrutura de dados AVL recursiva com números - PeguePague

Um estabelecimento comercial decidiu disponibilizar diversos pontos de consulta para que os próprios clientes consultem a disponibilidade de um produto, sirvam-se da quantidade desejada e dirijam-se ao caixa para o pagamento. Para que essa estratégia seja bem sucedida, ele o contratou para desenvolver o sistema de busca ao produtos, e, para maior eficiência, você optou pelo uso de uma árvore binária de busca balanceada - AVL. Utilizando o código fornecido implemente um árvore AVL com as funções de inserir e remover elementos. A ideia principal do problema é que implementação da remoção seja utilizando recursão, não sendo necessário o uso da função transplanta. A estratégia de remoção de elementos que não estejam em nós folhas é a substituição pelo elemento posterior (sucessor).

O código fornecido já tem o menu de entrada e o método de impressão. **Não** altere essas funções. O caminho mais fácil para a solução do exercício é não alterar a assinaturas das funções, dessa forma, não resultará em conflito com o código fornecido. O dado a ser armazenado na árvore é composto por uma **chave** do tipo *inteiro*, o **nome do produto** do tipo *string*, a **marca do produto** do tipo **string** e a **quantidade disponível** do tipo **inteiro**.

Entradas:

1. i - para **inserir** elemento : deve ler chave [inteiro] , nome do produto [string], marca [string] e quantidade disponível [inteiro]
2. r - para **remover** elemento : deve ler chave [inteiro]

3. b - para **buscar** elemento : deve ler chave [inteiro]
4. l - para fazer o **levantamento** da quantidade de produtos de uma dada marca: deve ler uma string
5. e - para **imprimir** árvore
6. f - finalizar

Saídas:

1. i - inserir elemento : A operação de inserção **não** produz saída
2. r - remover elemento : A operação de remoção **não** produz saída
3. b - buscar elemento : A operação de busca escreve a frase: "Elemento buscado: ([chave],[nome],[marca],[quantidade])", em que [chave], [nome], [local] e [quantidade] devem ser substituídos pela respectiva chave, nome do produto, marca e quantidade disponível."
4. l - levantamento por marca: Imprime: "Levantamento da marca [marca]: [quantidade]", em que [marca] é a marca pesquisada e [quantidade] é a quantidade de produtos dessa marca.
5. e - imprimir árvore : imprime a árvore seguindo o padrão de formatação tree
6. f - finalizar : finaliza o programa

Exemplo de Entrada:

```
i 100 chocolate SrCacau 30
i 187 sal MarSalgado 15
i 210 sorvete LeiGelato 28
i 130 biscoito CreckCrack 111
i 304 azeite MarSalgado 21
i 214 capuccino SrCacau 12
i 402 sardinha MarSalgado 21
l MarSalgado
l SrCacau
e
f
```

Exemplo de Saída:

```
Levantamento da marca MarSalgado: 57
Levantamento da marca SrCacau: 42
(187,sal)
 ├e─(100,chocolate)
 │   └d─(130,biscoito)
 └d─(214,capuccino)
     ├e─(210,sorvete)
     └d─(304,azeite)
         └d─(402,sardinha)
```

Exemplo de Entrada 2:

```
i 100 chocolate SrCacau 30
i 187 sal MarSalgado 15
i 210 sorvete LeiGelato 28
i 130 biscoito CreckCrack 111
i 187 sal MarSalgado 15
i 304 azeite MarSalgado 21
i 214 capuccino SrCacau 12
```

```
i 402 sardinha MarSalgado 21
i 87 pacoquinha DaDinda 13
i 55 azeitonas Filistao 24
i 13 macarrao Filistao 12
i 803 lasanha Filistao 21
i 255 mocaccinho SrCacau 13
b 210
r 210
r 210
b 210
l SrCacau
l Filistao
e
f
```

Exemplo de Saída 2:

```
Erro na inserção: chave já existente!
Elemento buscado: (210,sorvete,LeiGelato,28)
Erro na remoção: chave não encontrada!
Erro na busca: elemento não encontrado!
Levantamento da marca SrCacau: 55
Levantamento da marca Filistao: 57
(187,sal)
 ├e─(100,chocolate)
 │    ├e─(55,azeitonas)
 │    │    ├e─(13,macarrao)
 │    │    └d─(87,pacoquinha)
 │    └d─(130,biscoito)
 └d─(304,azeite)
      ├e─(214,capuccino)
      │    └d─(255,mocaccinho)
      └d─(402,sardinha)
           └d─(803,lasanha)
```

Observação: para baixar o arquivo, clique com o botão direito do mouse, depois clique em "salvar link como" e escolha uma pasta do seu computador. Não tente abrir o código no navegador e copiar o texto. Isso resultará em mudança na codificação do texto de UTF-8 para ISO, deformando a saída formatada da árvore.

Peso: 1

Última tentativa realizada em: 13/07/2020 18:52:22

Tentativas: 1 de 4

Nota (0 a 100): 36

Status ou Justificativa de Nota: O programa não resolve todas as instâncias do problema. O programa pode acessar posições indevidas da memória. A quantidade de dados escritos pelo programa é diferente da quantidade de dados esperados.

Ver Código da Última Tentativa

Minutos Restantes: 306

Usuário: Gabriel Nathan Almeida Silva

Notas:
Q1: ?
Q2: ?
Q3: ?
Q4: ?
Q5: 80
Q6: ?
Q7: ?
Q8: ?
Q9: ?
Q10: 36
Q11: ?
Q12: ?
Total: 10

Nova Resposta:
Selecione o arquivo com o código fonte do programa que resolve o problema para enviá-lo.

Escolher arquivo | Nenhum arquivo selecionado | Enviar Resposta

Minutos Restantes: 306

Usuário: Gabriel Nathan Almeida Silva

Notas:
Q1: ?
Q2: ?
Q3: ?
Q4: ?
Q5: 80
Q6: ?
Q7: ?
Q8: ?
Q9: ?
Q10: 36
Q11: ?
Q12: ?
Total: 10

Questão 11: Estrutura de dados AVL recursiva com números - Checagem CPF

Uma empresa está tentando se adaptar a rotina de distanciamento social vigente. Uma das medidas é restringir a entrada de funcionários ao espaço físico da empresa de acordo com o número do CPF. Em dias pares, somente funcionários com CPF finalizando em número par pode entrar, em casos de dias ímpares, somente funcionários com CPF finalizando em número ímpar. Para agilizar o controle de acesso, você deve criar um sistema para fazer buscas rápidas na base, para isso, utilizando árvores binárias de buscas balanceadas.

Utilizando o código fornecido, implemente uma árvore AVL com as funções de inserir, remover e buscar elementos. A idea principal do problema é que implementação da remoção seja utilizando recursão, não sendo necessário o uso da função transplanta. A estratégia de remoção de elementos, em casos que precise de substituição, é a substituição pelo elemento posterior (sucessor). Além disso, você deve desenvolver uma função, que, ao receber a chave de acesso do funcionário e o dia do mês, retorna se o funcionário está autorizado a entrar ou não. Outra função necessária é listar todos os funcionários que podem acessar a empresa, para isso, a função recebe o dia do mês como entrada, e realiza a impressão **em ordem** na AVL.

O código fornecido já tem o menu de entrada e o método de impressão. **Não** altere essas funções. O caminho mais fácil para a solução do exercício é não alterar a assinaturas das funções, dessa forma, não resultará em conflito com o código fornecido.

O dado a ser armazenado na árvore é composto por uma **chave** do tipo *inteiro*, um **nome** do tipo *string*, e um **CPF** do tipo *unsigned long long int*.

Entradas:

1. i - para **inserir** elemento : deve ler chave [inteiro] e valor [string]
2. r - para **remover** elemento : deve ler chave [inteiro]
3. b - para **buscar** elemento : deve ler chave [inteiro]
4. v - para **verificar** funcionario : deve ler chave [inteiro] e dia do mês [inteiro]
5. l - para **listar** funcionários : deve ler dia do mês [inteiro]
6. e - para **imprimir** árvore
7. f - finalizar

Saídas:

1. i - inserir elemento : A operação de inserção **não** produz saída

2. r - remover elemento : A operação de remoção **não** produz saída
3. b - buscar elemento : A operação de busca escreve a frase: "Nome: [nome] | CPF: [cpf] | Cod: [chave] ", em que [nome], [cpf] e [chave] são as informações armazenadas do funcionário.". Caso o elemento não seja encontrado, imprime a mensagem: "Erro na busca: elemento não encontrado!".
4. v - verificar funcionario : imprime a frase: "Autorizado.", caso a paridade seja a mesma entre o CPF e o dia do mês, caso contrário imprime "Não autorizado.", ou ainda, imprime "Erro na verificacao: chave inexistente!", caso a chave informada não esteja cadastrada.
5. l - listar funcionários : deve listar os funcionários permitidos a trabalhar no dia informado, um por linha, segundo a formatação "Nome: [nome] | CPF: [cpf] | Cod: [chave] ", em que [nome], [cpf] e [chave] são as informações armazenadas do funcionário.
6. e - imprimir árvore : imprime a árvore seguindo o padrão de formatação tree
7. f - finalizar : finaliza o programa

Exemplo de Entrada 1:

```
i 1 Gelo_Tancredo 11111111111
i 2 Sauna_Jose 22222222222
i 3 Gray_Fernando 33333333333
i 4 Liar_Itamar 44444444444
i 5 Rico_Fernando 55555555555
i 6 Hino_Luiz 66666666666
i 7 Huck_Dilma 77777777777
i 8 Coragem_Michel 88888888888
i 9 Naro_Jair 99999999999
e
f
```

Exemplo de Saída 1:

```
(4,Liar_Itamar)
 ├e─(2,Sauna_Jose)
 │    ├e─(1,Gelo_Tancredo)
 │    └d─(3,Gray_Fernando)
 └d─(6,Hino_Luiz)
      ├e─(5,Rico_Fernando)
      └d─(8,Coragem_Michel)
           ├e─(7,Huck_Dilma)
           └d─(9,Naro_Jair)
```

Exemplo de Entrada 2:

```
i 1 Gelo_Tancredo 11111111111
i 2 Sauna_Jose 22222222222
i 3 Gray_Fernando 33333333333
i 4 Liar_Itamar 44444444444
i 5 Rico_Fernando 55555555555
i 6 Hino_Luiz 66666666666
i 7 Huck_Dilma 77777777777
i 8 Coragem_Michel 88888888888
i 9 Naro_Jair 99999999999
v 3 21
```

```
v 3 12
v 10 12
l 11
b 5
r 5
b 5
l 22
v 5 12
r 9
f
```

Exemplo de Saída 2:

```
Autorizado.
Não autorizado.
Erro na verificacao: chave inexistente!
Nome: Gelo_Tancredo | CPF: 11111111111 | Cod: 1
Nome: Gray_Fernando | CPF: 33333333333 | Cod: 3
Nome: Rico_Fernando | CPF: 55555555555 | Cod: 5
Nome: Huck_Dilma | CPF: 77777777777 | Cod: 7
Nome: Naro_Jair | CPF: 99999999999 | Cod: 9
Nome: Rico_Fernando | CPF: 55555555555 | Cod: 5
Erro na busca: elemento não encontrado!
Nome: Sauna_Jose | CPF: 22222222222 | Cod: 2
Nome: Liar_Itamar | CPF: 44444444444 | Cod: 4
Nome: Hino_Luiz | CPF: 66666666666 | Cod: 6
Nome: Coragem_Michel | CPF: 88888888888 | Cod: 8
Erro na verificacao: chave inexistente!
```

Observação: para baixar o arquivo, clique com o botão direito do mouse, depois clique em "salvar link como" e escolha uma pasta do seu computador. Não tente abrir o código no navegar e copiar o texto. Isso resultará em mudança na codificação do texto de UTF-8 para ISO, deformando a saída formatada da árvore.

Peso: 1

Nova Resposta:

Selecione o arquivo com o código fonte do programa que resolve o problema para enviá-lo.

Escolher arquivo | Nenhum arquivo selecionado | Enviar Resposta

Questão 12: Estrutura de dados AVL recursiva com números - Pokemon

Está chegando o próximo mega-híper-super campeonato de Pokemon Come e você resolve colocar à prova os seus fabulosos conhecimentos. Para isso, você resolveu fazer um programinha para cadastrar os seus pokemons e, para maior eficiência, você optou pelo uso de uma árvore binária de busca balanceada - AVL. Utilizando o código fornecido, implemente um árvore AVL com as funções de inserir e remover elementos. A ideia principal do problema é que

implementação da remoção seja utilizando recursão, não sendo necessário o uso da função transplanta. A estratégia de remoção de elementos que não estejam em nós folhas é a substituição pelo elemento posterior (sucessor).

O código fornecido já tem o menu de entrada e o método de impressão. **Não** altere essas funções. O caminho mais fácil para a solução do exercício é não alterar a assinaturas das funções, dessa forma, não resultará em conflito com o código fornecido. O dado a ser armazenado na árvore é composto por um **id** do tipo *inteiro*, o **nome do pokemon** do tipo *string*, o **tipo principal do pokemon** do tipo **string** e o **nível alcançado** do tipo **inteiro**.

Entradas:

1. i - para **inserir** elemento : deve ler id [inteiro] , nome do pokemon [string], tipo [string] e nível [inteiro]
2. r - para **remover** elemento : deve ler id [inteiro]
3. b - para **buscar** elemento : deve ler id [inteiro]
4. l - para fazer o **levantamento** de pokemons de um determinado tipo e nível: deve ler uma string e um inteiro
5. e - para **imprimir** árvore
6. f - finalizar

Saídas:

1. i - inserir elemento : A operação de inserção **não** produz saída
2. r - remover elemento : A operação de remoção **não** produz saída
3. b - buscar elemento : A operação de busca escreve a frase: "Elemento buscado: ([id],[nome],[tipo],[nível])", em que [id], [nome], [tipo] e [nível] devem ser substituídos pelo respectivo id, nome do pokemon, tipo principal e nível alcançado."
4. l - levantamento por marca: Imprime: "Levantamento de pokemons tipo [tipo] e nível [nível]: [quantidade]", em que [tipo] é o tipo de pokemon, [nível] o nível alcançado e [quantidade] é a quantidade de pokemons atendendo esses dois requisitos.
5. e - imprimir árvore : imprime a árvore seguindo o padrão de formatação tree
6. f - finalizar : finaliza o programa

Exemplo de Entrada 1:

```
i 12 pikachu eletrico 25
i 51 charizard fogo 30
i 302 bulbassauro agua 40
i 14 eevee normal 45
i 96 magmar fogo 25
i 301 flareon fogo 30
i 13 jolteon eletrico 25
i 55 raichu eletrico 30
l fogo 30
l eletrico 25
l fogo 25
e
f
```

Exemplo de Saída1 :

```
Levantamento de pokemons tipo fogo e nível 30: 2
Levantamento de pokemons tipo eletrico e nível 25: 2
Levantamento de pokemons tipo fogo e nível 25: 1
(51,charizard)
 ├e─(13,jolteon)
 │   ├e─(12,pikachu)
 │   └d─(14,eevee)
 └d─(301,flareon)
     ├e─(96,magmar)
     │   └e─(55,raichu)
     └d─(302,bulbassauro)
```

Exemplo de Entrada 2:

```
i 12 pikachu eletrico 25
i 51 charizard fogo 30
i 302 bulbassaur agua 40
i 14 eevee normal 45
i 96 magmar fogo 25
i 102 cleffairy normal 45
i 402 snorlax normal 45
i 9 rattata normal 45
i 77 ditto normal 42
i 66 meowth normal 23
i 88 ponyta fogo 30
i 2 charmander fogo 30
i 301 flareon fogo 30
i 13 jolteon eletrico 25
i 55 raichu eletrico 30
e
l fogo 30
r 9
r 96
r 402
b 14
b 9
l normal 45
l fogo 25
e
f
```

Exemplo de Saída 2:

```
(77,ditto)
 ├e─(12,pikachu)
 │   ├e─(9,rattata)
 │   │   └e─(2,charmander)
 │   └d─(51,charizard)
 │       ├e─(14,eevee)
 │       │   └e─(13,jolteon)
 │       └d─(66,meowth)
 │           └e─(55,raichu)
 └d─(102,cleffairy)
     ├e─(96,magmar)
     │   └e─(88,ponyta)
     └d─(302,bulbassaur)
         ├e─(301,flareon)
         └d─(402,snorlax)
Levantamento de pokemons tipo fogo e nível 30: 4
```

```
Elemento buscado: (14,eevee,normal,45)
Erro na busca: elemento não encontrado!
Levantamento de pokemons tipo normal e nível 45: 2
Levantamento de pokemons tipo fogo e nível 25: 0
(77,ditto)
  ├e─(51,charizard)
  │    ├e─(12,pikachu)
  │    │    ├e─(2,charmander)
  │    │    └d─(14,eevee)
  │    │         └e─(13,jolteon)
  │    └d─(66,meowth)
  │         └e─(55,raichu)
  └d─(102,cleffairy)
       ├e─(88,ponyta)
       └d─(302,bulbassaur)
            └e─(301,flareon)
```

Observação: para baixar o arquivo, clique com o botão direito do mouse, depois clique em "salvar link como" e escolha uma pasta do seu computador. Não tente abrir o código no navegar e copiar o texto. Isso resultará em mudança na codificação do texto de UTF-8 para ISO, deformando a saída formatada da árvore.

Peso: 1

Nova Resposta:

Selecione o arquivo com o código fonte do programa que resolve o problema para enviá-lo.

Escolher arquivo | Nenhum arquivo selecionado | Enviar Resposta

Minutos Restantes: 306

Usuário: Gabriel Nathan Almeida Silva

Notas:
Q1: ?
Q2: ?
Q3: ?
Q4: ?
Q5: 80
Q6: ?
Q7: ?
Q8: ?
Q9: ?
Q10: 36
Q11: ?
Q12: ?
Total: 10