Prova III - Teoria

Entrega 20 jun em 10:30 **Pontos** 10 **Perguntas** 32
Disponível 20 jun em 8:30 - 20 jun em 10:30 2 horas
Limite de tempo 120 Minutos

Instruções

Nossa 3a prova de AEDs II tem duas partes: teórica e prática. Cada uma vale 10 pontos. A prova teórica será realizada no Canvas e a prática, no Verde.

A prova teórica será realizada no dia 20 de junho, no horário da aula teórica. Essa prova tem 30 questões de verdadeiro ou falso e duas questões abertas de código. As questões de verdadeiro ou falso valem 0,1 e as abertas, 3.5. Abaixo, seguem as regras para a prova.
1) O código de acesso à prova será fornecido pelo professor no início da prova.
2) Após o envio de uma questão não é permitido que o aluno volte na mesma.
3) A prova é individual e é permitida a consulta à cola que contém o nome do aluno.
4) A interpretação faz parte da prova.
5) Se existir algum erro, após a divulgação do gabarito, peça a anulação da questão.
6) Os alunos da manhã 1/manhã farão a prova nos lab 9.
7) Os alunos da turma 2/manhã farão a prova nos lab 2.
8) Os alunos da tarde farão a prova no lab 10.

Desejamos uma excelente prova para todos.

Este teste foi travado 20 jun em 10:30.

Histórico de tentativas

	Tentativa	Tempo	Pontuação
MAIS RECENTE	Tentativa 1	83 minutos	4,7 de 10

Pontuação deste teste: **4,7** de 10
Enviado 20 jun em 10:17
Esta tentativa levou 83 minutos.

Pergunta 1 — 0,1 / 0,1 pts

Os métodos *hashing* envolvem um processo de transformação de uma chave em um endereço. Sobre esses métodos, podemos afirmar que

quando duas ou mais chaves possuem o mesmo endereço primário ocorre uma colisão. Mesmo que se obtenha uma função *hash* que distribua as chaves de forma uniforme, existe grande chance de haver colisões. Além disso, deve haver uma forma de tratar as colisões. Uma das formas de se resolver as colisões é construindo uma lista encadeada para cada endereço da tabela. Assim, todas as chaves com mesmo endereço são encadeadas (TCE-RS'14, adaptada).

Correto!

- (●) Verdadeiro
- ( ) Falso

A afirmação é verdadeira. Ela aborda as colisões são o maior desafio no projeto das tabelas *hash*. A resolução adequada das colisões implica em reduzir o custo de pesquisa.

Pergunta 2

0,1 / 0,1 pts

Os métodos *hashing* envolvem um processo de transformação de uma chave em um endereço. Sobre esses métodos, podemos afirmar que o tempo gasto com pesquisas depende do tamanho da tabela e da quantidade de elementos inseridos. Além disso, a função *hash* de transformação deve envolver uma operação simples sobre a chave (TCE-RS'14, adaptada).

Correto!

- (●) Verdadeiro
- ( ) Falso

A afirmação é verdadeira. Primeiro, a dependência do tamanho da tabela e da quantidade de elementos acontece porque tais fatores influenciam nas colisões e o quão maior o número de colisões, maior o custo para encontrar elementos. Segundo, é importante que a operação de transformação seja simples, evitando o *overhead* de processamento.

Pergunta 3

0,1 / 0,1 pts

No método de transformação (*hashing*), os registros armazenados em uma tabela são diretamente endereçados a partir de uma transformação aritmética sobre a chave de pesquisa. Com relação às funções de transformação e colisões, podemos afirmar que a técnica de endereçamento aberto (*hash* direto) resolve colisões usando uma matriz esparsa (TRE-PI'16, adaptado).

Correto!

- (●) Falso
- ( ) Verdadeiro

A afirmação é falsa. O *hash* direto consiste em inserir os elementos que sofreram colisões dentro da própria tabela. Por exemplo, podemos utilizar área de reserva ou *overflow* e funções de *rehash*.

Pergunta 4

0,1 / 0,1 pts

Considere um arquivo sequencial, com 10.000 registros, cujas chaves identificadoras são números inteiros de até 8 dígitos. Para criar um índice tipo *hashing* para esse arquivo, contendo endereços de 0 até 11.999, uma definição adequada para uma função de *hashing* f(x),

onde x é uma chave e (a mod b) é o resto da divisão de a por b, seria f = x mod 12000 (DPE-RJ'14, adaptado).

- Falso
- Verdadeiro

Correto!

A afirmação é verdadeira. Uma função de transformação tradicional consiste no resto da divisão inteira do elemento pelo tamanho da tabela.

Pergunta 5

0,1 / 0,1 pts

Uma estrutura de dados eficiente é a tabela *hash*. Um desafio no projeto dessa estrutura é seu dimensionamento. Em uma tabela adequadamente dimensionada, com *n* chaves, o número médio de acessos para localização de uma chave é $\Theta(1)$ comparações (TJ-PI'15, adaptada).

- Falso
- Verdadeiro

Correto!

Uma tabela *hash* é bem dimensionada quando seu custo é constante, $\Theta(1)$ comparações.

Pergunta 6

0 / 0,1 pts

Uma vantagem da fragmentação na descida é que, na ascensão é preciso uma pilha para restaurar o equilíbrio da árvore repassando o caminho inverso de pesquisa no caso de fragmentações.

ocê respondeu

- (●) Falso

esposta correta

- ( ) Verdadeiro

A afirmação é verdadeira. A fragmentação na descida faz apenas atualizações locais envolvendo um nó e seu pai. Por outro lado, a técnica por ascensão faz com que exista um efeito cascata de fragmentações.

Pergunta 7

0,1 / 0,1 pts

Nas árvores 2.3.4, uma das vantagens da técnica de inserção com fragmentação na descida sobre a na ascensão é que a primeira normalmente consome menos espaço de memória.

Correto!

- (●) Falso
- ( ) Verdadeiro

A afirmação é falsa, pois a fragmentação na descida adianta algumas fragmentações, criando mais nós e, assim, consumindo mais memória.

Pergunta 8

0,1 / 0,1 pts

Na árvore 2.3.4, após uma inserção utilizando fragmentação na descida, podemos garantir a inexistência de nós do tipo 4 no caminho entre a raiz e a folha em que aconteceu a inserção.

Correto!

- (●) Falso

Verdadeiro

A afirmação é falsa. Na inserção com fragmentação na descida, quando chegamos em um nó do tipo 4, esse é fragmentado. Nessa fragmentação, se o pai era do tipo 3, ele ficará sendo do tipo 4. Da mesma forma, se inserirmos em uma folha do tipo 3, essa ficará sendo do tipo 4.

Pergunta 9

0,1 / 0,1 pts

A inserção dos números 4, 24, 10, 16, 2, 22, 18, 14, 8, 26, 20, 12 e 6 em uma 2.3.4 usando as técnicas de fragmentação por ascenção e na descida gera a mesma árvore resultante.

Correto!

- (•) Verdadeiro
- ( ) Falso

A afirmação é verdadeira. A inserção usando as duas técnicas é igual até a inserção do 20. Elas ficam diferentes após a inserção do 12 e voltam a ficar iguais com a inserção do 6.

Pergunta 10

0,1 / 0,1 pts

A árvore 2.3.4 é uma estrutura de pesquisa cujos nós são de três tipos (2-nó, 3-nó ou 4-nó) e as folhas estão situadas no mesmo nível.

- ( ) Falso

Correto!

- (•) Verdadeiro

A afirmação é verdadeira dado que os três tipos de nós foram citados e nesta árvore sempre inserimos novos valores nas folhas.

Pergunta 11

0,1 / 0,1 pts

Uma das estruturas de dados utilizadas na modelagem de sistemas de software denomina-se árvore rubro-negra. Nesse caso, um nó será vermelho quando, na árvore 2-3-4 correspondente, o valor desse nó for gêmeo do nó pai. Caso contrário, o nó é preto. Em uma árvore rubro-negra temos que a quantidade de nós vermelhos é sempre igual à de pretos.

- ○ Verdadeiro
- ◉ Falso

Correto!

A afirmação é falsa conforme os conceitos de árvores alvinegras apresentados na sala de aula.

Pergunta 12

0,1 / 0,1 pts

Uma das estruturas de dados utilizadas na modelagem de sistemas de software denomina-se árvore rubro-negra. Nesse caso, um nó será vermelho quando, na árvore 2-3-4 correspondente, o valor desse nó for gêmeo do nó pai. Caso contrário, o nó é preto. Em uma árvore rubro-negra temos que a quantidade de nós vermelhos é sempre par.

- ○ Verdadeiro
- ◉ Falso

Correto!

A afirmação é falsa conforme os conceitos de árvores alvinegras apresentados na sala de aula.

Pergunta 13

0,1 / 0,1 pts

Em uma árvore alvinegra, se tivermos dois nós pretos seguidos, temos que fazer uma rotação com o avô.

- Falso
- Verdadeiro

Correto!

A afirmação é verdadeira conforme os conceitos de árvores alvinegras apresentados na sala de aula.

Pergunta 14

0 / 0,1 pts

Uma árvore rubro-negra possui 18 valores inteiros distintos armazenados em seus 18 nós. A função recursiva boolean busca (int val) visita os nós desse tipo de árvore à procura de um determinado valor (val). O algoritmo utilizado tira partido das características de uma árvore rubro-negra, com o objetivo de ser o mais eficiente possível. Podemos afirmar que o número máximo de chamadas da função que será necessário para informar se um determinado valor está, ou não, armazenado na árvore é igual a seis (BNDES'13, adaptada).

esposta correta

- Verdadeiro

ocê respondeu

- Falso

A afirmação é verdadeira e pode ser confirmada executando o código disponibilizado pelo professor.

Pergunta 15

0,1 / 0,1 pts

Uma das estruturas de dados utilizadas na modelagem de sistemas de software denomina-se árvore rubro-negra. Nesse caso, um nó será vermelho quando, na árvore 2-3-4 correspondente, o valor desse nó for gêmeo do nó pai. Caso contrário, o nó é preto. Em uma árvore rubro-negra temos que se um nó é preto, seus filhos são pretos.

- Verdadeiro
- Falso

Correto!

A afirmação é falsa conforme os conceitos de árvores alvinegras apresentados na sala de aula.

Pergunta 16

0,1 / 0,1 pts

A AVL é uma árvore de busca autobalanceada. Isso significa que pode possuir até duas raízes (SEFAZ-PI'15, adaptado).

- Verdadeiro
- Falso

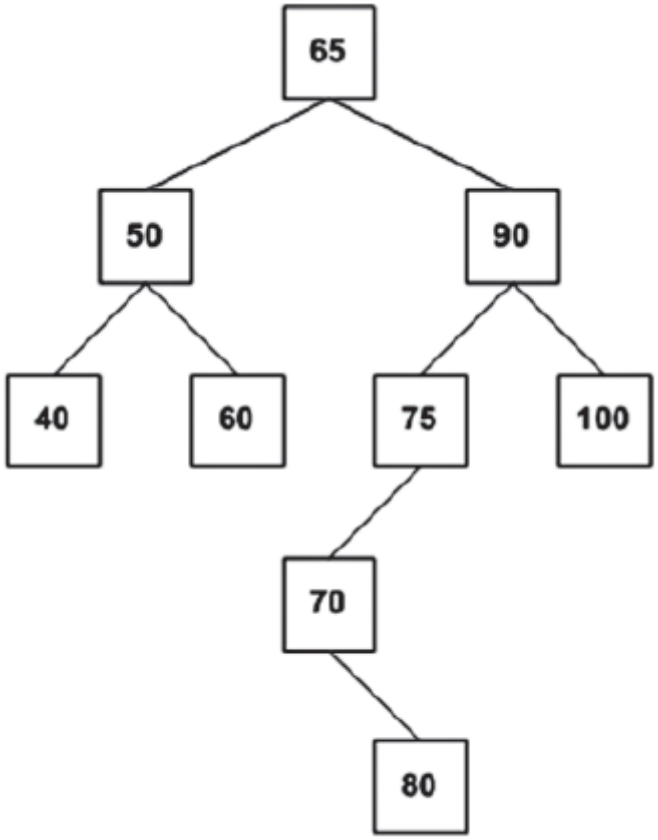
Correto!

A afirmação é falsa conforme o conceito de árvore AVL.

Pergunta 17

0,1 / 0,1 pts

A estrutura de dados AVL é uma árvore binária de busca balanceada criada pelos soviéticos Adelson, Velsky e Landis em 1962. Podemos afirmar que a figura abaixo representa uma árvore AVL (PETROBRAS'12, adaptada).



- Verdadeiro
- Falso

Correto!

A afirmação é falsa conforme o conceito de árvore AVL.

Pergunta 18

0,1 / 0,1 pts

A AVL é uma árvore de busca autobalanceada. Isso significa que as alturas das duas sub-árvores a partir de cada nó diferem no máximo em duas unidades (SEFAZ-PI'15, adaptado).

Correto!

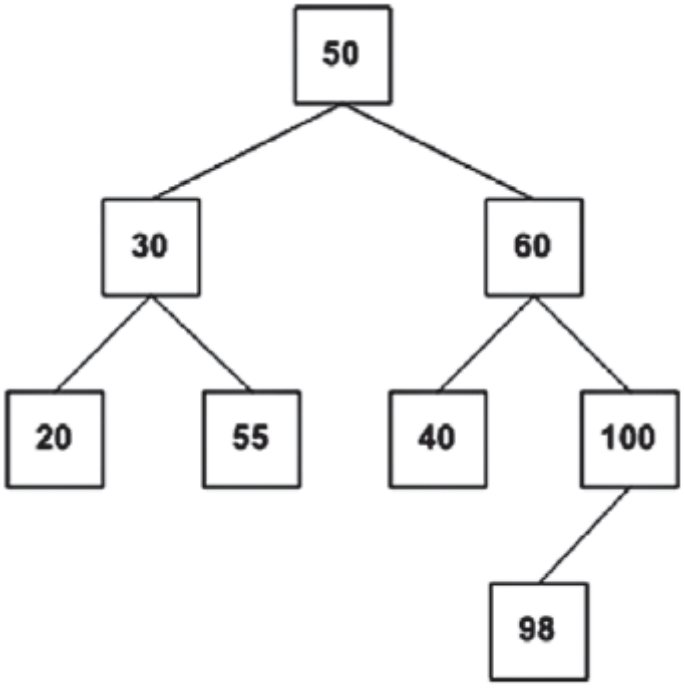
- (•) Falso
- ( ) Verdadeiro

A afirmação é falsa conforme o conceito de árvore AVL.

Pergunta 19

0 / 0,1 pts

A estrutura de dados AVL é uma árvore binária de busca balanceada criada pelos soviéticos Adelson, Velsky e Landis em 1962. Podemos afirmar que a figura abaixo representa uma árvore AVL (PETROBRAS'12, adaptada).



ocê respondeu

- (●) Verdadeiro

esposta correta

- ( ) Falso

A afirmação é falsa conforme o conceito de árvore AVL.

Pergunta 20

0 / 0,1 pts

Considere a assinatura das duas funções abaixo na linguagem C++:

void troca(double &x, double &y)
void troca(double* x, double* y)

Podemos afirmar que a primeira função utiliza passagem de parâmetros por referência e a segunda, por valor.

ocê respondeu

- (●) Falso

esposta correta

- ( ) Verdadeiro

A afirmação é verdadeira considerando os conceitos sobre passagem de parâmetros na linguagem C++ apresentados na sala de aula.

Pergunta 21

0,1 / 0,1 pts

Podemos afirmar que o código abaixo em C++ imprime "1 1" como saída

```
void f (int val, int& ref) {
   val ++; ref++;
```

```
}

void main () {
    int i = 1, j = 1;
    f(i, j);
   printf("%i -- %i", i, j);
}
```

Correto!

- (●) Falso
- ( ) Verdadeiro

A afirmação é falta. A variável ref foi passada por referência, assim, seu valor final é dois.

Pergunta 22

0,1 / 0,1 pts

Avaliando os códigos abaixo nas linguagens C e C++, é correto afirmar que os dois imprimem os mesmos valores para as variáveis "x" e "y" no final de suas execuções. Isso acontece mesmo com a passagem de parâmetros do primeiro código sendo por valor e a do segundo, sendo por referência.

Código em C:

```
void troca(double *x, double *y){
        double aux = *x;
        *x = *y;
        *y = aux;
}

int main(){
        double x = 7.0;
        double y = 5.0;
        troca(&x, &y);
        printf("X: %lf Y: %lf", x, y);
}
```

Código em C++:

```
void troca(double& x, double& y){
        double aux = x;
        x = y;
        y = x;
}

int main(){
        double x = 7.0;
        double y = 5.0;
        troca(x, y);
        cout << "X: " << x;
        cout << "Y: " << y;
}
```

- Falso
- Verdadeiro

Correto!

A afirmação é verdadeira conforme discutido em sala de aula.

Pergunta 23

0 / 0,1 pts

Considere que a Manausprev armazena os nomes dos beneficiários de aposentadorias em uma Árvore Binária de Busca (ABB). Ao se armazenar, nesta ordem, os nomes Marcos, José, Carolina, Paula, Rui, Pedro e Maria, a ABB resultante tem altura 3, que corresponde à altura mínima para armazenar os 7 nomes (MANAUSPREV'15, adaptada).

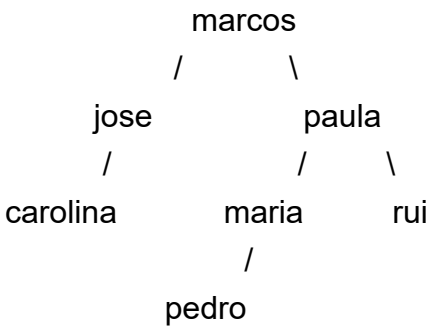
ocê respondeu

- Verdadeiro

esposta correta

- Falso

A afirmação é falsa e a árvore resultante é mostrada abaixo.



Pergunta 24

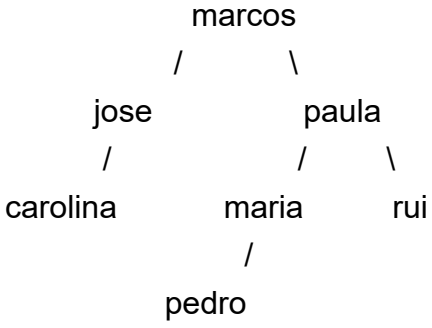
0,1 / 0,1 pts

Considere que a Manausprev armazena os nomes dos beneficiários de aposentadorias em uma Árvore Binária de Busca (ABB). Ao se armazenar, nesta ordem, os nomes Marcos, José, Carolina, Paula, Rui, Pedro e Maria, a ABB resultante requer no máximo 4 comparações para localizar qualquer um dos 7 nomes (MANAUSPREV'15, adaptada).

- Falso
- Verdadeiro

Correto!

A afirmação é verdadeira e a árvore resultante é mostrada abaixo.



Pergunta 25

0,1 / 0,1 pts

A altura mínima de uma árvore binária com *n* nós é lg(n).

- Verdadeiro
- Falso

Correto!

A afirmação é falsa a altura mínima de uma árvore binária com n nós é $\lfloor \lg(n) \rfloor + 1$.

Pergunta 26

0 / 0,1 pts

Considere que a Manausprev armazena os nomes dos beneficiários de aposentadorias em uma Árvore Binária de Busca (ABB). Ao se armazenar, nesta ordem, os nomes Marcos, José, Carolina, Paula, Rui, Pedro e Maria, a ABB resultante tem o nó contendo José possui dois filhos (MANAUSPREV'15, adaptada).

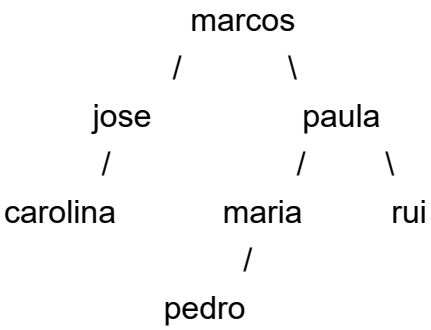
- Verdadeiro

ocê respondeu

- Falso

esposta correta

A afirmação é falsa e a árvore resultante é mostrada abaixo.



Pergunta 27

0,1 / 0,1 pts

Nas linguagens Java e C++, os atributos de uma classe podem ter as visibilidades public, private ou protected. Um atributo público é acessível dentro e fora da classe. Um privado, somente dentro da classe. Um protegido, somente dentro da classe ou de classes filhas.

Correto!

- (●) Verdadeiro
- ( ) Falso

As linguagens Java e C++ são orientadas por objetos e possuem os três tipos de visibilidade apresentados com suas respectivas definições.

Pergunta 28

0,1 / 0,1 pts

Nas linguagens C/C++, acessamos um atributo de um registro/objeto apontado por um ponteiro fazendo ponteiro->atributo.

Correto!

- (●) Verdadeiro
- ( ) Falso

O símbolo "->" permite acessarmos os atributos/funções de um registro/objeto apontados por um ponteiro.

Pergunta 29

0,1 / 0,1 pts

O código C++ abaixo tem um objeto chamado "quadrado_teste" do tipo Quadrado que pode ser usado globalmente.

```
class Quadrado {
   private:
      int lado;
   public:
      void set_values (int);

   //...
} ;

int main() {
   Quadrado quadrado_teste;

   //...
   return 0;
}
```

Correto!

- (•) Falso
- ( ) Verdadeiro

A declaração da classe e a criação do objeto foram feitos de maneira adequada. Entretanto, como o objeto foi declarado dentro da função main, o escopo de vida do mesmo é aquela função a partir do seu ponto de declaração.

Pergunta 30

0,1 / 0,1 pts

A linguagem C++ permite criar funções cuja implementação do código acontece “fora” da classe. Nesse caso, “dentro” da mesma temos a assinatura da função e,”fora”, implementamos a função usando o nome da classe e o operador “::”. O exemplo abaixo mostra uma

classe Cubo com sua função getVolume implementada de forma externa.

```
class Cubo {
   public:
      int lado;
      int getVolume();
}

int Cubo :: getVolume() {
   return lado*lado*lado;
}
```

- Falso
- Verdadeiro

Correto!

A afirmação é verdadeira e a implementação externa acontece conforme o exemplo apresentado.

Pergunta 31

2 / 3,5 pts

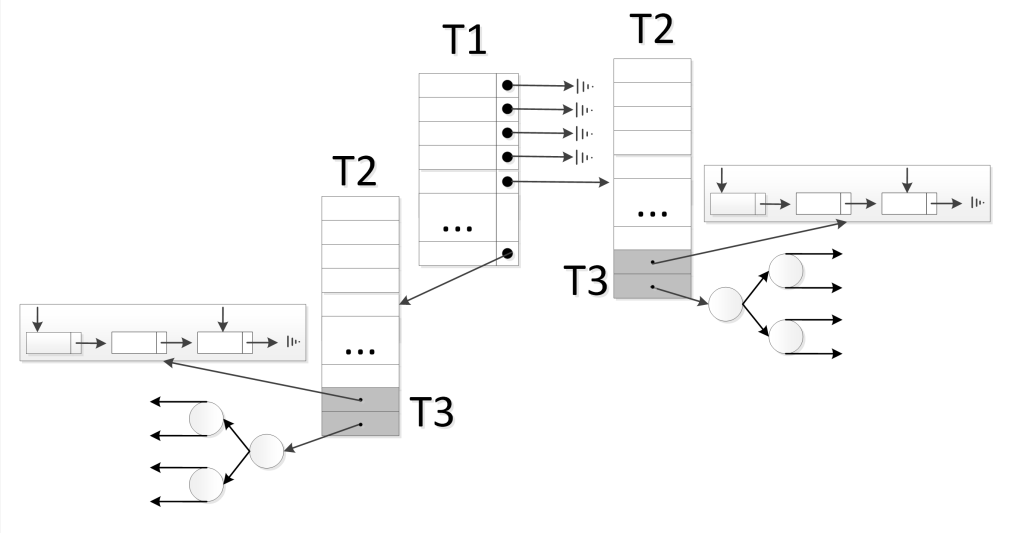
Seja a árvore Trie apresentada na sala de aula implemente um método para MOSTRAR todas as palavras contidas na árvore. Considere que a classe Nó da árvore Trie contém uma LISTA FLEXÍVEL para armazenar o endereço dos nós filhos. Apresente a ordem de complexidade do seu método. Sua resposta deve conter somente o método solicitado e sua complexidade.

Trie.java (https://pucminas.instructure.com/files/6327188/download)

Pergunta 32

0,3 / 3,5 pts

Considere a estrutura de dados Doidona ilustrada abaixo. O primeiro nível dessa estrutura contém uma tabela hash T1 sendo que cada célula dessa tabela possui um número inteiro elemento e um ponteiro T2 para outra tabela hash, T2. Cada célula de T1 aponta para uma T2. O tamanho de T1 é tam1 e de cada T2, tam2. Distribuímos os elemento em T1 com uma função int hashT1(int x) que retorna (x % tamT1). A T2 de cada célula distribui seus elementos com uma função int hashT2(int x) que retorna (x % tamT2) e trata as colisões com uma função int rehashT2(int x) que retorna (++x % tam2). Quando a função rehash} não consegue tratar as colisões, a T2 de cada nó trata essas colisões com a tabela hash} T3 (logicamente implementada). A função int hashT3(int x) retorna (x % 2). Quando esse retorno é zero, inserimos o valor em uma lista listaT3. Quando um, em uma árvore binária arvT3. Implemente o algoritmo para mostrar os elementos cadastrados nessa estrutura. Você N.Ã.O deve usar as estruturas de Árvore Binária nem de Lista, use apenas o Nó da Árvore e a Célula da Lista. A classe Doidona tem o atributo CelulaT1 t1[]. A classe CelulaT1 tem os atributos int elemento e HashT2 t2. A classe HashT2 tem os atributos int[] tab; Celula primeiroT3, ultimoT3; e No raizT3. As classes Celula e nó são iguais as vistas na sala de aula. Faça a análise de complexidade do seu método.



Doidona.java (https://pucminas.instructure.com/files/6327385/download)

Pontuação do teste: **4,7** de 10