CONCURSO ITA 2025

EDITAL: 03/ITA/2025

CARGO: TECNOLOGISTA

PERFIL: TL-14

CADERNO DE QUESTÕES

1. Esta prova tem duração de **4 (quatro) horas**.
2. Você poderá usar **apenas** caneta esferográfica de corpo transparente com tinta preta, lápis ou lapiseira, borracha, régua transparente simples e compasso. **É proibido portar qualquer outro material escolar ou equipamento eletrônico.**
3. Esta prova é composta de **25 questões de múltipla escolha** (numeradas de 01 a 25) e de **3 questões dissertativas**.
4. Você recebeu este **caderno de questões, uma folha de leitura óptica e um caderno de respostas** que deverão ser devolvidos ao final do exame.
5. As questões de **múltipla escolha devem ser respondidas na folha de leitura óptica**. Assinale a opção correspondente à resposta de cada uma das questões, de **01 a 25**. Cada questão de múltipla escolha admite uma única resposta.
6. A folha de leitura óptica, deve ser preenchida usando caneta preta. Você deve preencher todo o campo disponível para a resposta, sem extrapolar os limites, conforme instruções na folha de leitura óptica.
7. Cuidado para não errar no preenchimento da folha de leitura óptica. Ela não será substituída.
8. Não haverá tempo suplementar para o preenchimento da folha de leitura óptica.
9. As **questões dissertativas devem ser respondidas no caderno de respostas. Responda usando caneta preta, no campo destinado a cada questão**.
10. **É obrigatória a devolução do caderno de questões, do caderno de respostas e da folha de leitura óptica**, sob pena de desclassificação do candidato.
11. **Aguarde o aviso para iniciar a prova. Ao terminá-la, avise o fiscal e aguarde-o no seu lugar**.

Questão 1. Sobre o algoritmo de Dijkstra para encontrar caminhos mínimos sobre um grafo G=(V,E) não-dirigido e com pesos nas arestas. Uma variação comum do algoritmo armazena em cada nó um ponteiro u.p para o seu antecessor no caminho mais curto da raiz até ele. Chame essa variação de DijkstraP.

Defina: o subgrafo S(G) = (V,E') como um subgrafo de G, cujos vértices são os mesmos de G, mas E' é um subconjunto de E. Uma aresta (u,v) de E pertence a E' apenas se, depois de executado DijsktraP, v.p = u;

Marque a alternativa FALSA:

A ( ) O custo computacional em tempo de DijkstraP está na mesma classe de complexidade do Dijkstra original.

B ( ) Em termos de espaço, DijkstraP requer $\Theta(n)$ espaço adicional em relação ao Dijkstra original.

C ( ) Se G é conexo, S(G) é uma árvore.

D ( ) Defina o grafo I = (G, mas uma constante inteira C é adicionada ao peso de cada aresta de G). Pode-se provar que S(G) = S(I) exceto pelos pesos.

E ( ) S(G) pode ser uma árvore binária.

Questão 2. Suponha um grafo onde os vértices são cidades, as arestas estradas entre cidades, e o peso das arestas corresponde a distância entre pares de cidades.

É possível, portanto, se for fornecido um valor da velocidade do veículo, usar Dijkstra para encontrar o caminho mais rápido entre 2 cidades, e obter o tempo gasto nesse caminho mínimo. Para isto, basta modificar o custo das arestas para indicar o tempo = distância / velocidade e não a distância.

Qual das seguintes variações do problema exige, para ser resolvida, modificar o algoritmo Dijkstra original, não sendo possível utilizar o Dijstra inalterado, como caixa preta, e apenas modificar o grafo de entrada, inclusive a escolha dos vértices inicial e final?

Marque a alternativa CORRETA.

A ( ) Considerar que o usuário gasta um tempo extra na cidade onde começa a viagem e na cidade onde termina (sair/chegar de/em casa, des/arrumar a bagagem). É fornecida uma tabela destes tempos para cada cidade.

B ( ) Considerar que cada estrada tem uma velocidade máxima definida e fornecida, possivelmente menor que a velocidade nominal do veículo, e que o motorista nunca ultrapassa a velocidade permitida.

C ( ) Considerar que ao passar por uma cidade, o usuário gasta um tempo fixo e igual para todas as cidades (dado) para se locomover dentro da cidade.

D ( ) Considerar que ao passar por uma cidade, o usuário gasta um tempo predeterminado para se locomover dentro da cidade que varia de acordo com as estradas de chegada e de partida da cidade. Por exemplo, tabulamos um tempo gasto por quem chega em São José dos Campos pela Rodovia Dutra e parte pela Rodovia Tamoios, e outro tempo para quem chega pela Tamoios e parte pela Rodovia Monteiro Lobato. Para cada cidade, é dada uma matriz onde linhas e colunas são todas as estradas que incidem sobre a cidade, e os elementos são o tempo gasto para se deslocar dentro da cidade entre uma estrada e outra.

E ( ) Todas as alternativas anteriores podem ser resolvidas com o algoritmo original, se for permitido modificar a entrada, em tempo polinomial mesmo considerando a modificação da entrada.

Para as questões 3 e 4, considere as seguintes funções de hash em pseudo-código para gerar códigos de hash para strings.

```
int P( char *s, int tablesize)  {
     return s[0] % tablesize;
}

int S( char *s, int tablesize)  {
     int sum = 0;
     for (char *p = s; *p; p++)
          sum = (sum + *p) % tablesize;
     return sum;
}
```

Questão 3. As alternativas comparam as duas funções em relação à conveniência de serem usadas como função de hash, considerando o tempo de busca médio.

Marque a alternativa FALSA.

A ( ) S será provavelmente melhor que P, pois em muitos casos comuns, há muito mais strings que iniciam com alguns caracteres do que com outros. Um exemplo válido são palavras em português: há muitas palavras iniciando com ‘a’.

B ( ) S permite aumentar o tamanho da tabela de hash arbitrariamente, enquanto para P, se o tamanho da tabela de hash for maior que o número de caracteres do alfabeto, algumas posições (buckets) serão pouco ocupadas.

C ( ) Suponha que todas as strings contém apenas caracteres maiúsculos. Se aumentarmos o tamanho da tabela até milhares de posições, a ocupação da tabela de hash será pouco uniforme. Porque as somas dos caracteres estarão concentradas em torno dos múltiplos do valor médio dos caracteres.

D ( ) O problema descrito em C) pode ser resolvido, fazendo com que a ocupação da tabela de hash seja uniforme, bastando aumentar o tamanho da tabela de hash, definindo-o como o próximo número primo. E.g., se o tamanho era 1000, o mudamos para 1009.

E ( ) Se as strings armazenadas são sempre aleatórias, por exemplo, são sempre senhas geradas aleatoriamente, P e S podem ter, na média, desempenho equivalente em termos de colisões. Mas nesse caso, P é O(1) enquanto S é O(n).

Questão 4. Esta questão se refere a serviços P ou S que permite que usuários enviem strings que são armazenadas em uma tabela hash utilizando, respectivamente, as funções de hash de mesmo nome. Um atacante deseja rapidamente forçar que o tempo de busca de tais strings se torne Θ (n) ao invés de O(1), através do envio ao servidor de um conjunto apropriado de strings.

Marque a opção FALSA.

A ( ) O atacante escolhe entradas que iniciam pelo mesmo caractere. P é afetado enquanto S provavelmente não.

B ( ) O atacante escolhe um conjunto de entradas cuja soma de caracteres seja a mesma. S é afetado enquanto P provavelmente não.

C ( ) Uma forma de frustrar ataques como o descrito acima contra a função S, é escolher um tamanho primo para a tabela de hash.

D ( ) Em um sistema de código aberto onde o código da função de hash seja público, um atacante pode sempre escolher um conjunto de entrada específico para atacar qualquer função de hash completamente determinística.

E ( ) Uma defesa contra-ataques como o descrito acima é o Hashing Universal: ao iniciar a execução do serviço, uma função de hash aleatória é escolhida aleatoriamente.

Para as questões 5, 6, 7 e 8, defina G_{nxn} como um grafo não dirigido 2D grid com 4-vizinhança da seguinte forma:

- cada vértice é identificado com uma coordenada inteira 2D, de (1,1) até (n,n);
- há n x n vértices, correspondendo a todos os valores de (1,1) até (n,n);
- em cada vértice incidem até 4 arestas, a saber, para um vértice (i,j)

  $(i,j) \leftrightarrow (i-1,j)$
  $(i,j) \leftrightarrow (i+1,j)$
  $(i,j) \leftrightarrow (i\ ,j-1)$
  $(i,j) \leftrightarrow (i\ ,j+1)$

onde as arestas não existem se alguma coordenada é < 1 ou > n. A Figura 1 (a seguir) mostra um exemplo para n = 5

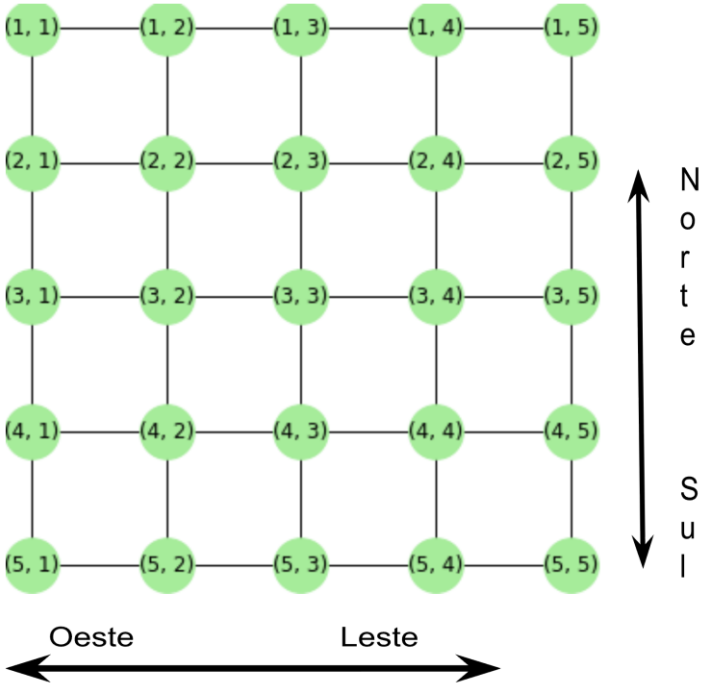

Figura 1

Questão 5. Marque a resposta mais exata e precisa. O número de arestas em G_{nxn} é:

A ( ) O(log n)

B ( ) O(n · log n)

C ( ) O(n)

D ( ) O(n^2)

E ( ) O(n^2 · log n)

Questão 6. Defina uma operação de 'contração' da seguinte forma:

```
para cada vértice u de grau 2
        se u foi deletado, ignore e continue o 'para'.
        v1 e v2 são os 2 vizinhos de u
        se não existe aresta (v1,v2)
                adicione ao grafo uma aresta (v1,v2)
        delete u, (u,v1), (u,v2)
```

Se aplicarmos a operação de contração a G_{nxn}, gerando um grafo G'_{nxn}, quantos vértices e arestas a menos terá G'_{nxn} em relação ao grafo original, supondo n maior que 10:

A ( ) Zero, o número de vértices e arestas continuará igual.

B ( ) 4 vértices, 4 arestas.

C ( ) 4 vértices, 8 arestas.

D ( ) O(n) vértices, O(n) arestas.

E ( ) O(n^2).

Questão 7. Ao aplicar o algoritmo DFS a G_{nxn} para obter uma árvore de busca, contendo apenas as arestas entre sucessor e antecessor na busca DFS, podemos afirmar sobre esta árvore:

A ( ) A profundidade será n², sempre.

B ( ) a profundidade será $O(n^2)$, sempre, mas alcançará exatamente o limite superior ou não dependendo da ordem em que os vizinhos de cada vértice são listados.

C ( ) A profundidade será O(log n), sempre.

D ( ) O número de arestas da árvore pode ser diferente de n^2-1, dependendo de como os vizinhos são ordenados.

E ( ) O número de folhas da árvore é ω(log n), sempre.

Questão 8. Suponha que os vizinhos de um vértice, durante a execução do DFS, são sempre listados na ordem Norte, Leste, Sul, Oeste, onde os pontos cardeais correspondem a uma realização do grafo sobre um sistema de coordenadas da forma trivial: o vértice (i,j) será colocado na coordenada (i,j), a primeira coordenada corresponde ao eixo Oeste (-) Leste (+), e a segunda ao eixo Sul (-) Norte (+), ainda correspondendo à figura anterior.

Suponha que o DFS inicia no vértice (1,1), e aplicamos a operação de contração sobre a árvore de busca DFS. Podemos afirmar:

A ( ) A árvore de busca DFS contraída tem profundidade O(1) e número de vértices O(1).

B ( ) A árvore de busca DFS contraída tem profundidade Ω(log n) e número de vértices Ω(1).

C ( ) A árvore de busca DFS contraída tem profundidade Ω(log n) e número de vértices Ω(log n).

D ( ) A árvore de busca DFS contraída tem profundidade Ω(log n) e número de vértices Ω(n).

E ( ) A árvore de busca DFS contraída tem profundidade Ω(n) e número de vértices Ω(n).

Questão 9. Sobre o padrão de projeto Iterator, onde, ao invés de percorrer uma estrutura com índices e ponteiros, há um objeto Iterator, com operações similares a begin, end, next. Suponha que várias estruturas de dados em uma biblioteca implementem Iterator.

Uma sintaxe comum para percorrer todos os itens de uma estrutura de dados E é:

```
tipo::iterator it;
for (it = E.begin(); it != E.end(); ++it) {
     // algum código onde o valor de cada item é acessado com *it
}
```

onde 'tipo' é um tipo apropriado de estrutura de dados, por exemplo, std::vector<string>.

Marque a opção FALSA.

A ( ) Uma vantagem é possibilitar substituir a estrutura de dados apenas trocando o tipo do objeto, mantendo inalterado o código que percorre todos os elementos.

B ( ) É possível implementar Iterators com a mesma interface com árvores balanceadas, por exemplo, uma árvore Rubro-Negra (*Red-Black Tree* em inglês)

C ( ) Uma vantagem é diminuir a possibilidade de erros devido a condições ou inicialização incorreta de índices, por exemplo, em for(int indice = 0; indice < n-1; indice++) é preciso acertar os valores 0 e n-1.

D ( ) Uma vantagem é diminuir significativamente o tempo de execução em relação à indexação com valores explícitos, pois as operações das classes Iterator são normalmente bem otimizadas.

E ( ) Para percorrer a estrutura de dados em outra ordem, pode ser preciso modificar ou substituir tanto as funções begin e end quanto a operação de incremento do Iterator.

Questão 10. Considere as definições:

```
public class SuperClass {
      public int op(int a, int b) { return a - b; }
}
public class SubClass extends SuperClass {
      public int op(int a, int b) { return a + b; }
}
// código Cliente de SuperClass e SubClass
SuperClass supC = new SuperClass( );  // linha 01
SubClass subC = new SubClass( );      // linha 02
 x1 = supC.op(6,4);                   // linha 03
supC = subC;                          // linha 04
x2 = supC.op(6,4);                    // linha 05
```

Assinale a alternativa mais correta sobre os valores de x1 e x2 depois da execução do pseudo-código acima.

A ( ) O valor de x1 será 2 e x2 será 10.

B ( ) Os valores de x1 e x2 serão 2.

C ( ) Haverá erro na linha 4, portando a linha 5 não executará e o valor de x2 não será definido.

D ( ) Haverá erro na linha 5, portando a linha 5 não executará e o valor de x2 não será definido.

E ( ) Nenhuma das anteriores.

Questão 11. Definição da operação de Partição:

- Entrada: uma lista ordenada de números, e um valor pivô.
- Saída: a mesma lista reordenada. Tal que sigam a seguinte ordenação: valores menores que pivô, seguidos de valores iguais a pivô, seguidos de valores maiores que pivô.

Por exemplo:

Entrada: {3, 4, 2, 9, 4, 5, 2, 8}, pivo 5

Saída: {3, 4, 2, 4, 2, 5, 9, 8}

Partição é usada para implementar a função Quicksort.

Se alguma opção de a) a d) for falsa, marque-a. Caso contrário, marque e).

A ( ) A operação Partição pode ser implementada com entrada do tipo vetor de inteiros, em O(n) tempo e O(1) espaço adicional.

B ( ) A operação Partição pode ser implementada com entrada do tipo lista simplesmente ligada, em O(n) tempo e O(1) espaço adicional, apenas manipulando-se ponteiros, sem alocar ou desalocar nenhuma outra lista ou nó.

C ( ) A função Quicksort pode ser implementado com entrada do tipo lista simplesmente ligada, em $O(n \cdot \log n)$ tempo e $O(\log n)$ espaço adicional.

D ( ) A operação Partição pode ser implementada com entrada do tipo lista simplesmente ligada, em O(n) tempo e O(1) espaço adicional, se alocarmos 3 listas auxiliares para os valores menores, iguais, e maiores que o pivô, e depois concatenarmos estas 3 listas.

E ( ) Todas as anteriores são verdadeiras.

Questão 12. Sobre as estruturas listadas abaixo (árvores balanceadas, listas), quais destas estruturas de dados NÃO podem ser concatenadas em O(1), mantendo as propriedades indicadas em cada alternativa?

A ( ) 2 listas ligadas ordenadas, com ponteiros para início e fim, e com $\max(1ª) \leq \min(2ª)$.

B ( ) 2 árvores de busca onde o maior elemento da primeira é menor que o menor elemento da segunda.

C ( ) 2 árvores de busca balanceadas onde o maior elemento da primeira é menor que o menor elemento da segunda.

D ( ) 2 listas duplamente ligadas não ordenadas.

E ( ) Todas podem ser concatenadas em O(1).

Para as questões 13, 14, 15 e 16, entre as seguintes estruturas de dados:

1. Árvore de busca balanceada
2. Árvore de busca não balanceada
3. Vetor de elementos ordenados
4. Lista ligada de elementos ordenados
5. Lista duplamente ligada de elementos ordenados

Questão 13. Quantas permitem percorrer os elementos tanto em ordem crescente quanto decrescente em O(n) tempo e O(log n) espaço adicional, no pior caso?

A ( ) 1 **B** ( ) 2 **C** ( ) 3 **D** ( ) 4 **E** ( ) 5

Questão 14. Quantas permitem percorrer os elementos tanto em ordem crescente quanto decrescente em O(n) tempo e O(n) espaço adicional, no pior caso?

A ( ) 1 **B** ( ) 2 **C** ( ) 3 **D** ( ) 4 **E** ( ) 5

Questão 15. Quantas permitem buscar elementos em O(log n) tempo e O(1) espaço adicional, no pior caso?

A ( ) 1 **B** ( ) 2 **C** ( ) 3 **D** ( ) 4 **E** ( ) 5

Questão 16. Quantas permitem buscar elementos em O(log n) tempo e O(log n) espaço adicional, no pior caso?

A ( ) 1 **B** ( ) 2 **C** ( ) 3 **D** ( ) 4 **E** ( ) 5

Questão 17. De acordo com a notação Big-O, marque a alternativa incorreta de a) a d), ou se todas estão corretas, marque e).

A ( ) $O(n) = O(n^2 \log n)$

B ( ) $1 + O(1/n) = O(1)$

C ( ) $O(1) = 1 + O(1/n)$

D ( ) $3 \cdot n + O(\log_{10} n) = O(n)$

E ( ) Todas as anteriores são corretas.

Questão 18. Ainda sobre a notação Big-O, marque a alternativa incorreta de a) a d), ou se todas estão corretas, marque e).

A ( ) se f = O(g) e g=O(f), então f = $\boldsymbol{\Theta}$(g) e g = $\boldsymbol{\Omega}$(f).

B ( ) se f = O(g) e g=O(f), então f = $\boldsymbol{\Theta}$(g) e g = $\boldsymbol{\Theta}$(f).

C ( ) se f = $\boldsymbol{\omega}$(g), então f não pertence a $\boldsymbol{\Theta}$(g).

D ( ) se f = $\boldsymbol{\Omega}$(g), então f não pertence a $\boldsymbol{\Theta}$(g).

E ( ) Todas as anteriores são corretas.

Questão 19. Marque a alternativa mais correta.

A ( ) O algoritmo de Dijkstra é guloso pois a cada momento trata apenas da fronteira entre os vértices com distância já definida e os ainda não visitados, e escolhemos o vértice de menor distância nessa fronteira.

B ( ) O algoritmo de Dijkstra é um algoritmo de programação dinâmica, pois se o grafo é representado como uma matriz de adjacências, Dijkstra pode ser implementado por um preenchimento organizado de uma matriz de distâncias.

C ( ) O algoritmo de Dijkstra é um algoritmo de divisão e conquista, pois o grafo é particionado em vértices com distância já definida; vértices ainda não visitados; e vértices na fronteira. Portanto, o problema é resolvido em partes, que depois compõem a versão final.

D ( ) Dijkstra não pode ser classificado em nenhum dos 3 paradigmas: Guloso; Programação Dinâmica; Divisão e Conquista.

E ( ) Todas as anteriores são verdadeiras.

Questão 20. Marque a alternativa mais correta.

A ( ) Todo algoritmo de divisão e conquista é necessariamente implementado com funções recursivas.

B ( ) Um algoritmo guloso não pode depender de preencher uma tabela, nesse caso seria considerado um algoritmo de Programação Dinâmica.

C ( ) Um algoritmo de Programação Dinâmica só faz sentido para problemas intratáveis, pressupondo-se que soluções eficientes são impossíveis, pois preencher uma tabela é necessariamente ineficiente.

D ( ) Um algoritmo guloso é necessariamente sub-ótimo.

E ( ) Todas as anteriores são falsas.

As questões 21, 22, e 23 se referem ao problema definido abaixo:

Definição do problema: para uma árvore binária balanceada (não necessariamente uma árvore de busca; cada nó contém apenas chave e dois ponteiros para os filhos esquerdo e direito), calcular o custo do caminho mais caro entre a raiz e uma folha, e imprimir o custo total e a lista de chaves dos nós deste caminho.

Notas:

- O custo de um caminho em uma árvore é a soma das chaves dos nós deste caminho.
- Considere que a altura da árvore é denotada por h, e que o tipo dos dados é inteiro.

Questão 21. Marque a alternativa FALSA.

A ( ) É possível estender o algoritmo de percurso pré-ordem recursivo para resolver este problema. Uma das soluções envolve passar uma pilha de inteiros como um parâmetro adicional na chamada recursiva do percurso pré-ordem.

B ( ) É possível estender o algoritmo de percurso pré-ordem iterativo para resolver este problema. Uma das soluções envolve manipular uma pilha de inteiros adicional.

C ( ) Pode ser resolvido em O(n · h) tempo e O(h) espaço adicional, sem modificar a estrutura de dados, se for permitido passar uma pilha como parâmetro por referência, de forma que não seja necessário copiar todo o conteúdo da pilha quando esta for passada como parâmetro.

D ( ) Pode ser resolvido em O(n · h) tempo e O(h) espaço adicional, sem modificar a estrutura de dados, mesmo se for permitido passar uma pilha como parâmetro apenas por valor de forma que seja necessário copiar todo o conteúdo da pilha passada como parâmetro a cada chamada de função.

E ( ) Se a árvore for balanceada, h = O(log n) e, portanto, as classes de complexidade O(n · h) e O(h) poderiam ser escritas como O(n · log n) e O(log n), respectivamente.

Questão 22 Marque a alternativa FALSA.

A ( ) É preciso percorrer toda a árvore para encontrar o caminho máximo entre a raiz e uma folha, portanto, um limite inferior para qualquer algoritmo é $\boldsymbol{\Omega}$(n) em tempo de execução.

B ( ) Como se requer a impressão dos valores ao longo do caminho máximo da raiz a uma folha, o tamanho da resposta é $\boldsymbol{\Theta}$(h). Portanto, um limite inferior para qualquer algoritmo é $\boldsymbol{\Omega}$(h) em tempo de execução.

C ( ) Considerando os itens a) e b) acima, e que é preciso encontrar o caminho antes de imprimi-lo, um limite inferior para qualquer algoritmo é $\boldsymbol{\Omega}$(n + h) em tempo de execução.

D ( ) A existência de um algoritmo $\boldsymbol{\Theta}$(n) em tempo de execução é compatível com a existência de um limite inferior para qualquer algoritmo $\boldsymbol{\Omega}$(n + h) em tempo de execução.

E ( ) O raciocínio expresso nas alternativas a), b), c) e d) não é válido para árvores balanceadas, é válido apenas se h = $\boldsymbol{\omega}$(log n).

Questão 23. Marque a opção mais correta e precisa. Se uma árvore tem n nós, o número de arestas é:

A ( ) n **B** ( ) n - 1 **C** ( ) O(n) **D** ( ) n/2 + O(log n) **E** ( ) O(log n)

Questão 24. Se há alguma opção falsa entre a) e d), marque-a; senão, marque e). Se uma árvore tem n > 3 nós:

A ( ) A altura da árvore é O(n).

B ( ) A altura da árvore é O(log n).

C ( ) A altura da árvore é o máximo entre as alturas dos filhos esquerdos e direito da raiz, acrescido de mais uma unidade.

D ( ) Se a árvore for completa, a altura é O(log n).

E ( ) Todas as anteriores são verdadeiras.

Questão 25. Considere o seguinte algoritmo para salvar e recuperar árvores de busca (BST) balanceadas em/de arquivos. No exemplo, consideramos uma árvore Rubro-Negra.

Salvar arvore em arquivo

1. Percorra a árvore em-ordem, e salve os elementos em um arquivo na ordem em que foram percorridos.

Recuperar uma arvore a partir de um arquivo:

2. Leia o arquivo, e coloque os elementos em um vetor. Estes elementos estarão ordenados.
3. Insira os elementos, na ordem em que foram lidos, em uma árvore Rubro-Negra
4. Temos uma BST com os elementos originais.

Se existe uma alternativa falsa entre a) e d), marque-a; senão, marque e).

A ( ) O custo de salvar a árvore é O(n).

B ( ) O custo de recuperar a árvore a partir do arquivo é O(n log n).

C ( ) A árvore recuperada é exatamente igual à original, em toda a sua estrutura e topologia.

D ( ) A árvore recuperada tem os mesmos elementos da árvore original.

E ( ) Todas são verdadeiras.

QUESTÕES DISCURSIVAS

Questão discursiva 1. Suponha uma nova operação Tree::FindKth(i), que retorna o elemento da árvore binária de busca T com a i-ésima menor chave. Assuma que todos os elementos possuem chaves distintas. Explique como modificar a BST para suportar essa operação em O(log n) no caso médio, sem sacrificar o tempo de execução de nenhuma outra operação.

Atenção: é necessário informação adicional nos nós. Para calcular e armazenar esta informação em cada nó, deve-se realizar uma varredura prévia na árvore, em tempo O(n). A operação FindKth é O(log n) apenas se supormos que a varredura prévia já foi feita e já existe a informação adicional.

Explique como implementar a operação nas condições indicadas, em alto-nível, com pseudo-código, desenhos, ou diagramas que exemplifiquem a execução.

Questão discursiva 2. Considere o seguinte código de um destrutor de uma árvore binária, onde TreeNodeObject é a classe que representa um nó de árvore e left_child e right_child são atributos da mesma classe, ponteiros para os filhos esquerdo e direito do nó de tipo apropriado. Considere que estes dois atributos são os únicos atributos de tipo ponteiro da classe TreeNodeObject.

Versão 1

```
TreeNodeObject::~TreeNodeObject // destrutor da classe
{
     if (left_child) delete left_child;
     if (right_child) delete right_child;
}
```

Um aluno que não sabe C++, adaptou o código para uma função em C:

Versão 2

```
void deleteTree(TreeNodeStruct *node)
{
     if (left_child) free(node->left_child);
     if (right_child) free(node->right_child);
}
```

E no fim do `main.cpp`, o aluno deleta a árvore com:

```
deleteTree(rootNode); // rootNode é um ponteiro para a raiz da árvore
```

Sendo avisado que o código acima está errado, substituiu-o por uma versão com recursão explícita:

Versão 3

```
void deleteTree(TreeNodeStruct *node)
{
      if(!node) return;
      deleteTree(node->left_child);
      deleteTree(node->right_child);
}
```

e ainda deleta a árvore da mesma forma.

```
deleteTree(rootNode); // rootNode é um ponteiro para a raiz da árvore
```

Em sua resposta, aborde necessariamente os seguintes itens:

a) O código Versão 1 está correto? Se não, explique porque e corrija-o.
b) Explique porque o código Versão 2 está errado e que tipo de erro ele causa. (note que não é preciso corrigi-lo)
c) O código Versão 3 está correto? Se não, explique porque e corrija-o.

Questão discursiva 3. Se a classe Elipse for pai da classe Circle (faz sentido, porque círculos são elipses), a classe Circle pode reusar todo o conteúdo da classe Elipse, basta sobrescrever métodos para garantir que os eixos menor e maior permaneçam iguais. No entanto, o método:

```
// “estica” a elipse na direcao do eixo maior
void Ellipse.stretchMaior()
```

não funciona com círculo, pois o resultado deixa de ser um círculo.

Se fizermos círculo pai de Elipse, seria conceitualmente errado, já que nem toda elipse é círculo.

Além disso, o seguinte método não faz sentido com uma Elipse.

```
// retorna o raio de um círculo
double Circle.getRadius()
```

a) Explique este dilema conceitualmente usando o vocabulário de POO, especialmente os conceitos de responsabilidade e herança.
b) Forneça uma solução que ainda promova o reuso de código. A sua solução pode ter uma desvantagem, no ponto de vista do programador que usa as suas classes. Explique conceitualmente a solução e a desvantagem, usando o vocabulário de POO, do ponto de vista do programador que usa as suas classes. A solução pode ser específica para uma linguagem.

Não é necessário implementações completas, mas use trechos de pseudo-código na resposta quando relevante.