1ª Prova de EDA-II - Curso: Engenharia de Software

Data: 29 de Abril de 2025 **Horário limite:** 15:50

Perguntas comuns e suas respostas:

- P: Tenho uma dúvida na questão tal.
  R: A compreensão do enunciado faz parte da prova.
- P: O que será corrigido?
  R: A lógica, a criatividade, a sintaxe, o uso correto dos comandos e dos tipos, os nomes das variáveis, a indentação, uso equilibrado de comentários no código e, evidentemente, a clareza. Nesta prova, você deverá sobretudo escrever códigos modulares, usando corretamente funções e/ou procedimentos, conforme o caso, além de uso correto de variáveis locais ou globais e a passagem de parâmetros por referência ou por valor.
- P: Posso fazer a lápis?
  R: Não. A prova deverá ser feita a caneta.
- P: Posso responder na folha de questões?
  R: Não. A prova deverá ser respondida na folha de respostas.
- P: Quanto vale esta prova?
  R: Esta prova vale **30** pontos. Há mais pontos distribuídos na prova, no entanto, você poderá somar no máximo **30** pontos.

(5 pontos) 1) Considere a implementação de uma tabela hash utilizando Endereçamento Aberto (Sondagem Linear), para tratamento de colisões tendo 10 como o limite de colisões aceitável. Responda as perguntas abaixo:

a) **(3 pontos)** O `grau` de colisão é a quantidade de colisões que um hash específico possui. Implemente uma função `grau`, em C, que recebe a tabela hash, e uma chave, a função deve devolver a quantidade de colisões para esta hash.

- É possível responder o `grau` de uma hash qualquer em tempo constante, i.e $\mathcal{O}(1)$?

b) **(2 pontos)** Qual o impacto nas operações(inserção,remoção,busca) quando se aumenta o limite de 10 colisões para $M/2$, sendo M a quantidade de Itens que podem ser armazenados na tabela hash?

(8 pontos) 2) Uma **árvore Red-Black Descendente Esquerdista** é uma árvore binária de busca balanceada, na qual cada nó pode ser colorido de vermelho ou preto, obedecendo a certas propriedades que garantem altura limitada e desempenho eficiente nas operações.

1. **(3 pontos)** Liste e explique as propriedades que uma árvore Red-Black Descendente Esquerdista deve obedecer.
2. **(3 pontos)** Considere a seguinte sequência de inserções em uma árvore Red-Black Descendente Esquerdista vazia: **30, 15, 60, 7, 22, 17**. Desenhe a árvore resultante após cada inserção, mostrando as cores dos nós.
3. **(2 pontos)** Explique por que a operação de inserção em uma árvore Red-Black Descendente Esquerdista pode exigir rotações e recolorações, mas a operação de busca não

(5 pontos) 3) *"Small HashTables are Slower than Arrays"* , esta afirmação, que pode ser traduzida para "Tabelas Hash pequenas são mais lentas que vetores", é uma afirmação muito forte e também esquisita. Há alguma verdade nesta afirmação? É possível estimar quão pequena uma tabela hash precisa ser para ser mais lenta que vetores? Estime os tamanhos comparando os diferentes tipos de métodos de colisão de chaves

em tabelas hash; vetor ordenado, e; vetor não ordenado. E considere também as situações: Insere todos os elementos de uma vez e depois faz muitas buscas, e; Inserções e buscas são intercaladas.

(5 pontos) 4) Considere o cenário de implementação de um compilador, onde é necessário gerenciar uma tabela de símbolos, responsável pelo armazenamento e recuperação rápida de identificadores (nomes de variáveis, funções, etc.) durante a análise do código fonte. Nesta tabela de símbolos, as seguintes operações são necessárias com frequência:

- Inserir novos identificadores à medida que são encontrados.
- Buscar rapidamente por um identificador para verificar se já foi declarado.
- Listar os identificadores em ordem alfabética quando solicitado pelo usuário para fins de depuração.

Dentre as seguintes estruturas de dados: **árvore red-black, tabela hash, vetor ordenado, lista encadeada**, escolha qual seria a mais adequada para implementar a tabela de símbolos neste cenário. Justifique sua resposta levando em conta as operações mencionadas e as características de cada estrutura.

(8 pontos) 5) A **skip-list** é uma estrutura de dados probabilística usada para manter um conjunto ordenado de elementos, permitindo buscas, inserções e remoções eficientes, de forma semelhante a árvores balanceadas, porém utilizando múltiplos "níveis" de listas ligadas.

1. **(2 pontos)** Explique como a estrutura de uma skip-list permite que operações de busca sejam realizadas, comparando o desempenho esperado dessas operações com árvores balanceadas como a árvore red-black.
2. **(3 pontos)** Suponha que você deseja inserir a chave 42 em uma skip-list já existente. Descreva, passo a passo, como ocorre essa inserção, incluindo o processo probabilístico que determina o nível da nova chave.
3. **(3 pontos)** Considere que você está utilizando uma skip-list para armazenar um conjunto dinâmico de números inteiros. Apresente e comente um trecho de código em C que realiza a operação de busca por um elemento nesta estrutura.

(10 pontos) 6) Você está implementando uma tabela hash, em C, para armazenar elementos inteiros, utilizando **endereçamento aberto** (sondagem linear) como estratégia de tratamento de colisões, e não impôs nenhum limite de colisões. No entanto, ao implementar as operações de inserção, busca e remoção, percebeu que a abordagem tradicional para busca e remoção pode apresentar problemas caso apenas o valor *NULLitem* (um valor especial representando posição vazia) seja usado para sinalizar que o slot está livre.

Problemas identificados:

- Em uma inserção ou busca, a exploração de slots deve parar ao encontrar um slot vazio (*NULLitem*).
- Contudo, ao remover um elemento, ao simplesmente definir o slot como *NULLitem*, isso pode causar problemas em buscas futuras, que podem terminar prematuramente e erroneamente sinalizar que um item não existe quando, na verdade, ele pode estar em outro slot mais à frente devido a colisões anteriores.

Nestes termos, responda:

1. Implemente funções C para **inserção**, **busca** e **remoção** que consideram o *NULLitem* para gerenciar corretamente a tabela hash sem limite arbitrário para colisões.

Assuma um vetor global `ht[]` de tamanho M, e os seguintes protótipos:

```
1 void HTinsert(int key);
2 Item HTsearch(int key);
3 void HTremove(int key);
```

Mostre e explique cada função.