UNIVERSIDADE FEDERAL DO CEARÁ
Campus de Quixadá
Prof. Arthur Araruna
QXD0115- Estrutura de Dados Avançada

LE6
2025.1

Nome: _____ Matrícula: _____

1 Exercícios de Fixação

1. Faça o que se pede:
   (a) Desenhe a visualização de árvore para um *heap-min* que resulte da inserção das seguintes chaves: $\langle 77, 22, 9, 68, 16, 34, 13, 8\rangle$.
   (b) Desenhe como esse heap ficará após duas remoções.

2. Considere o vetor $A = [29, 18, 10, 15, 20, 9, 5, 13, 2, 4, 15]$.
   (a) Esse vetor satisfaz a Propriedade Heap-max? Se não, conserte-o movendo o menor número de elementos.
   (b) Usando esse vetor (talvez corrigido), ilustre a execução de uma remoção. Para cada alteração no vetor, reescreva-o.

3. Uma fila de prioridade de duas pontas (*double-ended priority queue*) é uma fila de prioridade que permite remover tanto o elemento de maior prioridade quanto o de menor prioridade.

   Enquanto você e um amigo estavam escrevendo um programa, perceberam que seria necessária uma fila de duas pontas. Seu amigo sugere uma forma de implementar essa fila: Manter um *heap-max* e um *heap-min*, inserindo os novos elementos em ambos sempre que for necessária uma inserção. Para remover o de menor prioridade, removemos do *heap-min* e para remover o de maior prioridade, removemos do *heap-max*.

   Infelizmente, essa ideia não funciona. Escreva um exemplo onde essa forma de implementação fornece uma resposta diferente da esperada.

4. **ENADE 2011.**
   As filas de prioridades (*heaps*) são estruturas de dados importantes no projeto de algoritmos. Em especial, *heaps* podem ser utilizados na recuperação de informação em grandes bases de dados constituídos por textos. Basicamente, para se exibir o resultado de uma consulta, os documentos recuperados são ordenados de acordo com a relevância presumida para o usuário. Uma consulta pode recuperar milhões de documentos que certamente não serão todos examinados. Na verdade, o usuário examina os primeiros m documentos dos n recuperados, em que m é da ordem de algumas dezenas.

   Considerando as características dos *heaps* e sua aplicação no problema descrito acima, avalie as seguintes afirmações.

   I. Uma vez que o *heap* é implementado como uma árvore binária de pesquisa essencialmente completa, o custo computacional para sua construção é $O(n \log n)$.
   II. A implementação de *heaps* utilizando-se vetores é eficiente em tempo de execução e em espaço de armazenamento, pois o pai de um elemento armazenado na posição i se encontra armazenado na posição $2i + 1$.
   III. O custo computacional para se recuperar de forma ordenada os m documentos mais relevantes armazenados em um *heap* de tamanho n é $O(m \log n)$.
   IV. Determinar o documento com maior valor de relevância armazenado em um *heap* tem custo computacional $O(1)$.

   Está correto apenas o que se afirma em:

A. I e II.
B. II e III.
C. III e IV.
D. I, II e IV.
E. I, III e IV.

5. O tipo de *heaps* que vimos em sala é chamado de *heaps binários*, pois a ordem parcial induzida pela propriedade relaciona um elemento com outros dois, ou melhor dizendo, a visualização da ordem parcial, é dada por uma árvore binária. Quando a relação é dada de um para m elementos, denominamos a estrutura de *m-heap*.

   Descreva a propriedade que definiria um 3-*heap* (ou *heap ternário*) em um vetor, e escreva os respectivos algoritmos Subir e Descer desse tipo de heap.

   OBS: Lembre-se que a visualização da ordem parcial deve dar-se por uma árvore ternária, o que significa que não podem haver elementos com mais de um pai e muito menos elementos sem pai (fora a raiz). Considere que começamos a indexação do vetor pelo índice 1.

6. Você deseja armazenar n elementos e decidiu que irá usar um *heap* ou um vetor ordenado. Considere as operações a seguir e determine se, quando cada operação é a mais frequentemente necessária em uma aplicação, a melhor escolha de estrutura será o *heap*, o vetor ou se não fará diferença.
   (a) Encontrar o maior elemento.
   (b) Remover um elemento.
   (c) Encontrar o maior elemento.
   (d) Inserir um novo elemento.
   (e) Encontrar o maior e o menor elemento simultaneamente.
   (f) Determinar os k menores elementos.

2 Exercícios de Aplicação

7. Escreva o algoritmo HeapSort(V, n), que recebe um vetor V de n números inteiros e devolve esse vetor ordenado de maneira *não-decrescente*. Escreva todos os algoritmos necessários para essa tarefa.

8. Escreva um algoritmo que, dado um vetor V com n elementos não ordenados e um valor k com $1 \leqslant k \leqslant n$, é capaz de determinar o k-ésimo menor dos elementos de V com uma complexidade $O(n + k \lg n)$.

3 Desafios

9. Um elemento bastante procurado em coleções de elementos em certas aplicações é a *mediana*. A mediana de um conjunto de n valores é aquele elemento que "divide ao meio" o conjunto, ou seja, é maior que $\lfloor \frac{n}{2} \rfloor$ e menor que $\lfloor \frac{n}{2} \rfloor - 1$ elementos. Para essas aplicações, seria interessante saber qual é esse tal elemento eficientemente, e mais ainda, poder inserir elementos em uma estrutura de forma a manter a eficiência da busca pela mediana.

   Explique como, através do uso de exatamente um *heap-max* e um *heap-min*, podemos construir uma estrutura composta de coleção de números que sempre saiba em $O(1)$ qual é a mediana dos números colecionados. Escreva os algoritmos de inserção de novos elementos e de remoção da mediana para essa estrutura.

   OBS: A *mediana* de um conjunto de n elementos é um elemento que, se ordenássemos todos, residiria aproximadamente na metade do resultado. Dito de outra forma, existem aproximadamente $\frac{n}{2}$ elementos menores e aproximadamente $\frac{n}{2}$ elementos maiores que a mediana.