# ESTRUTURA DE DADOS AVANÇADA - 01A - 2023.1



## Avaliação 07: Fila de prioridade usando heaps

### Fase de avaliação

Instruções para avaliação

# Respostas

Corrija com seriedade as questões do colega. Forneça comentários e explique a ele o que ele errou.

**Questão 1:** Escreva pseudocódigos para os procedimentos **decreaseKey(A,i,newKey), minMoveDown(A, i), minMoveUp(A,i), increaseKey(A,i,newKey), minHeapInsert(A,key) e extractMinimum(A)**, que implementem uma fila de prioridade mínima com um heap de mínimo. **Obs.:** Note que todos esses procedimentos são versões similares às vistas em sala, só que aqui estou pedindo para uma fila de prioridades mínima.

**Solução:**

```
Algoritmo: moveUp(A,i)
  Entrada: vetor A e índice i do elemento a ser movido para cima
  Saída: heap de mínimo A

pai = i/2
if pai >= 1 then
  if A[i] < A[pai] then
    troca A[i] e A[pai]
    moveUp(A, pai)
```

```
Algoritmo: min-heap-decreaseKey(A,i,newKey)
  Entrada: vetor A, índice i e nova chave do elemento A[i]
  Saída: heap de mínimo A

A[i] = newKey
moveUp(A,i)
```

```
Algoritmo: moveDown(A,i)
  Entrada: vetor A e índice i do elemento a ser movido para baixo
  Saída: heap de mínimo A

j = 2 * i
if j <= A.heapSize then
  if j < A.heapSize then
    if A[j+1] < A[j] then
      j = j + 1
  if A[i] > A[j] then
    troca A[i]  e A[j]
    moveDown(A,j)
```

```
Algoritmo: min-heap-increaseKey(A,i,newKey)
  Entrada: vetor A, índice i e nova chave do elemento A[i]
  Saída: heap de mínimo A

A[i] = newKey
moveDown(A,i)
```

```
Algoritmo: minHeapInsert(A,newKey)
  Entrada: vetor A e nova valor a ser inserido
  Saída: heap de mínimo A

A.heapSize = A.heapSize + 1
A[A.heapSize] = newKey
moveUp(A, A.heapSize)
```

```
Algoritmo: extractMinimum(A)
  Entrada: vetor A
  Saída: valor do elemento mínimo

if A.heapSize < 1 then
  error "heap underflow"
dados = A[1]
A[1] = A[A.heapSize]
A.heapSize = A.heapSize - 1
moveDown(A,1)
return dados
```

**Questão 2:** Seja $A$ um heap de máximo. A operação **HEAP-DELETE(A, i)** elimina o item no nó $i$ do heap $A$. Dê uma implementação correta de **HEAP-DELETE(A,i)** que seja executada no tempo $O(\lg n)$ para um heap de máximo de $n$ elementos.

**Solução:**

```
Algoritmo: HEAP-DELETE(A, i)
  Entrada: heap A e índice i do elemento a ser removido
  Saída: heap de máximo A

if A[i] > A[A.heap-size] then
  A[i] = A[A.heap-size]
  max-heap-moveDown(A,i)
else
  max-heap-increaseKey(A, i, A[A.heap-size])
A.heapSize = A.heapSize - 1
```

**Questão 3:** Um heap $d$-ário é semelhante a um heap binário, mas (com uma única exceção possível) nós que não são folhas têm $d$ filhos em vez de dois filhos.

a. Como você representaria um heap $d$-ário de máximo em um vetor? Justifique.
   **Solução:**

   Um heap d-ário de máximo é um vetor $A[0..n-1]$ com $n$ elementos, que respeita a seguinte propriedade:

$$A\left[\frac{i}{d}\right] \geq A[i] \quad \text{para todo } d \leq i \leq n.$$

   Vou considerar que o número no slot A[i] é a própria prioridade. Também estou considerando que o vetor começa no índice 0, diferente do que fiz no heap binário da aula, pois assim os cálculos dos índices dos filhos e do pai fica mais fácil, mas você poderia ter considerado um heap d-ário começando do índice 1 também (não teria problema).
   Assim, o pai de um elemento com índice $i > 0$ é o elemento com índice $\left\lfloor \frac{(i-1)}{d} \right\rfloor$.
   Os filhos de um elemento com índice $i > 0$ são os elementos com índices no intervalo de $d \cdot i + 1$ até $d \cdot i + d$.
   **Justificativa:** Essa representação de heap nos dá uma árvore d-ária completa na qual todos os níveis dessa árvore tem todos os nós possíveis, com exceção do último nível, que pode ter menos do que $2^{h-1}$ nós. Ademais, os nós nessa árvore são preenchidos da esquerda para a direita, a partir da raiz, como se estivéssemos seguindo um percurso em largura da esquerda para a direita. Assim, os nós do último nível estão todos o mais a esquerda possível na árvore.

b. Qual é a altura de um heap $d$-ário de $n$ elementos em termos de $n$ e $d$? Justifique.
**Solução:**

Seja $n \geq 1$ o número total de elementos em um heap d-ário $A$ com $d \geq 2$. Então, a altura $h$ desse heap d-ário $A$ é

$$h \leq \lceil \log_d (n(d-1)+1) \rceil.$$

**Prova:** Estou considerando que toda folha numa árvore tem altura igual a 1.
Seja então $A$ um heap d-ário com $n \geq 1$ elementos, como descrito no enunciado.
Já sabemos, que esse heap pode ser visto como uma árvore d-ária completa $T$ de altura $h$.
Para facilitar os cálculos também vou supor que $T$ além de completa é também cheia, ou seja, todos os $h$ níveis da árvore $T$ possuem o máximo número de nós possíveis.
Veja que, com essa suposição, a altura $h$ que eu encontrar para a árvore d-ária cheia $T$ será um limitante superior para todas as árvores d-árias completas de altura $h$.
Logo, no nível 1 da árvore $T$ temos $d^0$ nós. No nível 2 temos $d^1$ nós. No nível 3 temos $d^2$ nós. No nível 4 temos $d^3$ nós e, assim por diante, até que, por fim, no nível último nível $h$ temos $d^{h-1}$ nós.
Assim, o número de nós $n$ da árvore d-ária cheia $T$ é dado por:

$$n = d^0 + d^1 + d^2 + \cdots + d^{h-1} = \sum_{i=0}^{h-1} d^i = \frac{d^h - 1}{d-1}.$$

Ou seja,
$n = \frac{d^h-1}{d-1} \implies n(d-1) = d^h - 1 \implies n(d-1)+1 = d^h \implies \log_d (n(d-1)+1) = \log_d d^h \implies h = \log_d (n(d-1)$ ·
.
Logo, considerando todas as árvores d-árias completas $T'$ de altura $h$, temos que sua altura é limitada pela altura da árvore d-ária cheia de mesma altura. Assim, temos que $h \leq \lceil \log_d (n(d-1)+1) \rceil$. ■

c. Dê uma implementação eficiente de EXTRACT-MAXIMUM() em um heap de máximo $d$-ário. Analise seu tempo de execução em termos de $d$ e $n$.
**Comentário:** Não vou resolver essa questão. Acredito que de tudo o que já foi exposto em sala, nos slides, no livro e do que foi feito nessa atividade, se você chegou até aqui, acredito que você consiga ler a resposta do colega e avaliar se ele fez ou não essa questão correta.
A dificuldade nessa resolução reside na construção da função moveDown(A,i) para heaps d-ários. Pois, agora, como um nó pode ter $d$ filhos, escolher para onde descer no heap envolve fazer da ordem de $d$ comparações para saber por qual ramo descer na árvore.

Envios atribuídos para avaliação ▾

◄ heap de mínimo online

Seguir para...

modelo MathJax usado para escrever a atividade ►