QXD0115 - ESTRUTURA DE DADOS AVANÇADA - 01A - 2024.1

AC02 - envio

Fase de avaliação

Fase de configuração	Fase de envio	Fase de avaliação Fase atual	Fase de cálculo da nota da avaliação	Encerrado
	Envie seu trabalho Prazo limite dos envios: sexta, 13 set 2024, 13:00 (4 dias atrás)	Avaliar colegas total: 2 pendente: 2 Prazo limite da avaliação: terça, 17 set 2024, 23:59 (hoje)		

Seu envio

Instruções para avaliação

Soluções

- **Questão 1.** (Exercício 5.8 -- Livro do Jayme) Dê exemplo de uma família de árvores AVL cuja exclusão de nós implica a realização de $O(\lg n)$ operações de rotação para o rebalanceamento. Você deve fornecer um argumento/justificativa para a sua resposta.

  Solução: Uma família de árvores que é solução para essa questão é a família das árvores de Fibonacci T_h, que são as árvores AVL que possuem o menor número de nós para uma certa altura h fixa.

  Uma árvore de Fibonacci T_h é definida recursivamente da seguinte forma:

  - se $h = 0$, então $T_h = \emptyset$;
  - se $h = 1$, então T_h é uma folha;
  - se $h > 1$, então para a raiz r de T_h vale a seguinte regra: a subárvore esquerda de r é a árvore T_{h-2} e a subárvore direita de r é a árvore T_{h-1}.

  Pela definição recursiva dada acima, obtemos imediatamente que o número de nós de T_h é dado pela seguinte relação de recorrência:

$$N(T_h) = \begin{cases} 0 & \text{se } h = 0; \\ 1 & \text{se } h = 1; \\ 1 + N(T_{h-1}) + N(T_{h-2}) & \text{se } h > 1. \end{cases}$$

  O nó v que precisa ser removido da árvore de Fibonacci T_h a fim de desencadear o rebalanceamento de todos os seus ancestrais é o nó localizado mais à esquerda na árvore T_h. Quando h é ímpar, temos que v é uma folha; e quando h é par, temos que v não tem filho esquerdo mas tem uma folha como filho direito.

  Ao removermos v da árvore, automaticamente o pai de v fica desregulado e passa a ter balanço igual a $+2$, o que é regulado com uma rotação simples à esquerda. A partir daí, essa situação vai se repetindo para todos os nós ancestrais de v à medida que o algoritmo recursivo de inserção vai voltando das chamadas recursivas: todos os ancestrais de v passam a ficar desregulados com balanço igual a $+2$, situação esta que é solucionada com uma rotação simples à esquerda. Isso ocorre até chegarmos à raiz de T_h, e aí com uma última rotação simples à esquerda, regulamos toda a árvore T_h. Como $altura(T_h) = O(\lg N(T_h))$, então são realizadas no máximo $O(\lg N(T_h))$ rotações.

  É preciso ainda argumentar porquê os nós ancestrais vão ficando com balanço $+2$. Isso não é difícil de argumentar. Por definição de árvore de Fibonacci, todos os ancestrais p de v já tinham $balance(p) = +1$ antes da remoção. Após a remoção de v, o balanço do nó ancestral mais próximo de v salta de $+1$ para $+2$ e vai precisar ser regulado. Cada regulagem desregula o próximo ancestral mais próximo.

- **Questão 2.** (Exercício 5.9 -- Livro do Jayme) **Nesta questão, sua tarefa é detalhar o algoritmo de remoção de chaves em árvore AVL.** Essa questão é uma questão dissertativa. Imagine-se escrevendo uma seção de livro sobre inserção em árvore AVL, ou um artigo para colocar no seu blog da internet, no qual você se propõe a explicar o algoritmo de remoção de chaves em árvores AVL. É preciso explicar desde a etapa de busca pela chave a ser removida, até a etapa final de exclusão da chave, se ela existir. Junto com o texto, apresente os pseudocódigos das operações envolvidas. A etapa de regulagem dos nós ancestrais deve estar clara. Você também deve explicar os casos em que um nó deve ser regulado.

  Diretrizes para a correção dessa questão: A sua tarefa aqui era pôr em palavras tudo o que eu discuti em sala e que coloquei também nos slides.

  O que deve ser avaliado nessa questão:

  - Foi descrito como funciona os primeiros passos do algoritmo a fim de encontrar o nó a ser deletado?
  - Após encontrar o nó a ser deletado, foram discutidos os possíveis problemas que podem ocorrer ao se tentar deletar um nó que tem dois filhos, ou só um filho, ou nenhum filho? Em quais casos o aluno dividiu a análise dessa parte do algoritmo? Ele soube fazer isso?
  - Após ter decidido qual nó foi removido, foram descritos os possíveis casos de remoção que podem levar à desregulagem do nó p ancestral mais próximo? Foi descrito como resolver cada desregulagem? Qual rotação foi utilizada para resolver cada caso?

  Tudo isso acima precisa ter sido escrito para que a questão seja considerada completamente correta.

- **Questão 3.** 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: minFixUp(A,i)
  Entrada: vetor A e índice i do elemento a ser movido para cima
  Saída: heap de mínimo A

1.  pai = i/2
2.  if pai >= 1 then
3.    if A[i] < A[pai] then
4.      troca A[i] e A[pai]
5.      minFixUp(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
1.  if A[i] < newKey then
2.      error "wrong key"
3.  A[i] = newKey
4.  minFixUp(A,i)
```

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

1.  j = 2 * i
2.  if j <= A.heapSize then
3.    if j < A.heapSize then
4.      if A[j+1] < A[j] then
5.        j = j + 1
6.    if A[i] > A[j] then
7.      troca A[i]  e A[j]
8.      minFixDown(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
if A[i] > newKey
    error "wrong key"
A[i] = newKey
minFixDown(A,i)
```

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

if A.heapSize == A.length then
    error "heap overflow"
A.heapSize = A.heapSize + 1
A[A.heapSize] = newKey
minFixUp(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
minFixDown(A,1)
return dados
```

- **Questão 4.** 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 i < 1 or i > A.heapSize then
    error "invalid index"
if i == A.heapSize then
    A.heapSize = A.heapSize - 1
else if A[i] > A[A.heapSize] then
  A[i] = A[A.heap-size]
  A.heapSize = A.heapSize - 1
  maxFixDown(A,i)
else
  max-heap-increaseKey(A, i, A[A.heapSize])
  A.heapSize = A.heapSize - 1
```

- **Questão 5.** 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:

$$A\left[\frac{i-1}{d}\right] \geq A[i] \quad \text{para todo } 1 \leq i \leq n.$$

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 neste caso mais genérico 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 d^{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,
$n = \frac{d^h-1}{d-1} \Longrightarrow n(d-1) = d^h - 1 \Longrightarrow n(d-1)+1 = d^h \Longrightarrow \log_d (n(d-1)+1) = \log_d d^h \Longrightarrow h = \log_d (n(d -$
.
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 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 maxFixDown(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.

◄ Visualização de um heap máximo

Seguir para...

Slides: tabela hash ►

Cedro – Quixadá – Ceará CEP: 63902-580
Secretaria do Campus: (88) 3411-9422