Cabula AED
May The Force Be With You

Diogo Sousa, Helder Martins, Daniel Parreira

17 de Janeiro de 2010

1 *Disclaimer*

Se algo estiver errado avisem, mas se chumbarem no exame porque o que aqui está não está correcto, *i couldn't care less.*

2 Resolução de Recorrências

2.1 Recorrência I

Seja $a \geq 0$, $b \geq 1$ e $c \geq 1$ constantes.

$$T(n) = \begin{cases} a & n = 0 \\ b \cdot T(n-1) + c & n > 0 \end{cases}$$

$$= \begin{cases} O(n) & b = 1 \\ O(b^n) & b > 1 \end{cases}$$

2.2 Recorrência II (*Master Theorem*)

Seja $a \geq 0$, $b \geq 1$ e $c > 1$ constantes e $f(n) = O(n^k)$, para qualquer $k \geq 0$.

$$T(n) = \begin{cases} a & n = 0 \\ b \cdot T(\frac{n}{c}) + f(n) & n > 0 \end{cases}$$

$$= \begin{cases} O(n^k) & b < c^k \\ O(n^k \, log_c(n)) & b = c^k \\ O(n^{log_c(b)}) & b > c^k \end{cases}$$

3 Tipos Abstractos de Dados

3.1 Operações

- *Stack*
  - boolean isEmpty()
  - int size()
  - E top()
  - void push(E e)
  - E pop()
- *Queue*
  - boolean isEmpty()
  - int size()
  - void enqueue(E e)
  - E dequeue()
- *List*
  - boolean isEmpty()
  - int size()
  - Iterator<E> iterator()
  - E getFirst()
  - E getLast()
  - E get(int p)
  - int find(E e)
  - void add(int p, E e)
  - void addFirst(E e)
  - void addLast(E e)
  - E remove(int p)
  - E removeFirst()
  - E removeLast()
  - boolean remove(E e)
- *Dictionary*
  - boolean isEmpty()
  - int size()
  - V find(K k)
  - V insert(K k, V v)
  - V remove(K k)
  - Iterator<Entry<K,V>> iterator()
- *Ordered Dictionary*
  - Todos os métodos de *Dictionary*.
  - Entry<K,V> minEntry()
  - Entry<K,V> maxEntry()
- *Priority Queue*
  - boolean isEmpty()
  - int size()
  - Entry<K,V> minEntry()
  - void insert(K k, V v)
  - Entry<K,V> removeMin()
- *Iterator<E>*
  - boolean hasNext()
  - E next()
  - void rewind()
- *TwoWayIterator<E>*
  - Todos os métodos de *Iterator<E>*
  - boolean hasPrevious()
  - E previous()
  - void fullForward()
- *Entry<K,V>*
  - V getKey()
  - K getValue()

3.2 Implementações

- *Stack*
- *Queue*
- *List*
  - Lista Ligada
    - Simples
    - Dupla
    - Apenas com cabeça
    - Com cabeça e cauda
- *Dictionary*
  - *Hashtable*
    - *Separate Chaining* (Dispersão Aberta)
    - *Open Addressing* (Dispersão Fechada)
      - *Linear Probing* (Sondagem linear)
      - *Double Hashing* (Dispersão dupla)
- *Ordered Dictionary*
  - *Binary Search Tree*
  - *Red-Black Tree*
  - *AVL Tree*
- *Priority Queue*
  - *Heap*

3.3 Nós de EDs

- *BSTnode*
  - BSTNode(K key, V value)
  - BSTNode(K key, V value, BSTNode<K,V> left, BSTNode<K,V> right)
  - EntryClass<K,V> getEntry()
  - K getKey()
  - V getValue()
  - BSTNode<K,V> getLeft()
  - BSTNode<K,V> getRight()
  - void setEntry(EntryClass<K,V> newEntry)
  - void setEntry(K newKey, V newValue)
  - void setKey(K newKey)
  - void setValue(V newValue)
  - void setLeft(BSTNode<K,V> newLeft)
  - void setRight(BSTNode<K,V> newRight)
  - boolean isLeaf()
- *DListNode*
  - DListNode(E theElement)
  - DListNode(E theElement, DListNode<E> thePrevious, DListNode<E> theNext)
  - E getElement()
  - DListNode<E> getPrevious()
  - DListNode<E> getNext()
  - void setElement(E newElement)
  - void setPrevious(DListNode<E> newPrevious)
  - void setNext(DListNode<E> newNext)
- *AVLNode*
  - AVLNode(K key, V value)
  - AVLNode(K key, V value, char balance, AVLNode<K,V> left, AVLNode<K,V> right)
  - char getBalance()
  - void setBalance(char newBalance)
- *RBNode*
  - RBNode(K key, V value)
  - RBNode(K key, V value, boolean colour, RBNode<K,V> left, RBNode<K,V> right)
  - boolean getColour()
  - boolean isRed()
  - boolean isBlack()
  - void setColour(boolean newColour)
  - void makeRed()
  - void makeBlack()

4 Complexidades Temporais das Estruturas de Dados

São omitidas as seguintes estruturas de dados por terem complexidade constante em todas as operações, em todos os casos: *Stack com lista ligada*, *Queue com lista ligada*. A *lista ligada* é omitida só porque sim!

4.1 *HashTable (Separate Chaining)*

Assume-se que a resolução de colisões é feita com a *OrderedDLL*.

	Melhor caso	Pior Caso	Caso Esperado
Pesquisa	$O(1)$	$O(n)$	$O(1+\lambda)$
Insersão	$O(1)$	$O(n)$	$O(1+\lambda)$
Remoção	$O(1)$	$O(n)$	$O(1+\lambda)$

4.2 *Binary Search Tree*

	Melhor caso	Pior Caso	Caso Esperado
Pesquisa	$O(1)$	$O(n)$	$O(lg(n))$
Insersão	$O(1)$	$O(n)$	$O(lg(n))$
Remoção	$O(1)$	$O(n)$	$O(lg(n))$
Min/Max	$O(1)$	$O(n)$	$O(lg(n))$
Iterar †	$O(n)$	$O(n)$	$O(n)$

† Isto é o custo de iterar, não de obter o iterador!

4.3 *AVL/Red-Black Tree*

	Melhor caso	Pior Caso	Caso Esperado
Pesquisa	$O(1)$	$O(lg(n))$	$O(lg(n))$
Insersão	$O(lg(n))$	$O(lg(n))$	$O(lg(n))$
Remoção	$O(lg(n))$	$O(lg(n))$	$O(lg(n))$
Min/Max	$O(lg(n))$	$O(lg(n))$	$O(lg(n))$
Iterar †	$O(n)$	$O(n)$	$O(n)$

† Isto é o custo de iterar, não de obter o iterador!

4.4 *Binary Heap*

	Melhor caso	Pior Caso	Caso Esperado
Criar c/ n elementos	$O(n)$	$O(n)$	$O(n)$
Inserir	$O(1)$	$O(lg(n))$	$O(lg(n))$
Mínimo	$O(1)$	$O(1)$	$O(1)$
Remover Mínimo	$O(1)$	$O(lg(n))$	$O(lg(n))$

4.5 Ordenação

	Melhor caso	Pior Caso	Caso Esperado	Estavel
Insertion	$O(n)$	$O(n^2)$	$O(n^2)$	Sim
Bubble	$O(n^2)$	$O(n^2)$	$O(n^2)$	Sim
Selection	$O(n^2)$	$O(n^2)$	$O(n^2)$	Não
Heap	$O(n)$	$O(n\,lg(n))$	$O(n\,lg(n))$	Não
Merge	$O(n\,lg(n))$	$O(n\,lg(n))$	$O(n\,lg(n))$	Sim
Quick	$O(n\,lg(n))$	$O(n^2)$	$O(n\,lg(n))$	Não
Jogo	$O(\infty)$	$O(\infty)$	$O(\infty)$	?

5 Generalidades

5.1 Árvores Binárias

- Uma árvore com n nós tem $n+1$ sub-árvores vazias.
- Uma árvore com n nós tem altura entre $\lceil lg(n+1) \rceil$ e n.
- Uma árvore diz-se perfeitamente equilibrada se para todo o nó o número de nós no seu ramo esquerdo varia no máximo em uma unidade absoluta do número de nós no seu ramo direito.
- Uma árvore diz-se equilibrada se para todo o nó a altura do seu ramo esquerdo varia no máximo em uma unidade absoluta da altura do seu ramo direito.
- Percursos
  - Prefixo — raiz, sub-árvore esquerda, sub-árvore direita.
  - Infixo — sub-árvore esquerda, raiz, sub-árvore direita.
  - Sufixo — sub-árvore esquerda, sub-árvore direita, raiz.

5.2 *Red-Black Tree*

Mantém as seguintes propriedades:

1. Cada nó é preto ou vermelho.
2. A raiz é preta.
3. Filhos de nós vermelhos são pretos.
4. Todos os caminhos da raiz a qualquer árvore vazia têm o mesmo número de nós pretos.