

UFPI - CCN - DC  
Ciência da Computação  
Estrutura de Dados

## Tabela de Símbolos (Busca Binária)



Prof. Raimundo Moura  
[rsm@ufpi.edu.br](mailto:rsm@ufpi.edu.br)

95

### TS implementada com busca binária

- ▶ Supor que as chaves de uma TS são *comparáveis* (*menor, igual, maior*) e o tipo *Key* tem um método *compareTo()*.
  - ▶ Ex: Key igual a Integer, ou Double, ou String etc.
- ▶ Dizemos então que a TS é *ordenada (ordered)*.
- ▶ Nossas *convenções gerais para TSs* continuam valendo em TSs ordenadas.

96

## TS implementada com busca binária

- O *posto (rank)* de qualquer objeto  $k$  do tipo Key (que pode ou não estar na TS) é o número de chaves da TS que são estritamente menores que  $k$ .
- Convém acrescentar a operação `rank()` à API da classe `ST` no caso de TSs comparáveis:

<code>int rank(Key key)</code>	número de chaves menores que <code>key</code>
--------------------------------	---

97

## TS implementada com vetores ordenados

		keys[]										vals[]										
key	value	0	1	2	3	4	5	6	7	8	9	N	0	1	2	3	4	5	6	7	8	9
S	0	S										1	0									
E	1	E	S									2	1	0								
A	2	A	E	S								3	2	1	0							
R	3	A	E	R	S							4	2	1	3	0						
C	4	A	C	E	R	S						5	2	4	1	3	0					
H	5	A	C	E	H	R	S					6	2	4	1	5	3	0				
E	6	A	C	E	H	R	S					6	2	4	6	5	3	0				
X	7	A	C	E	H	R	S	X				7	2	4	6	5	3	0	7			
A	8	A	C	E	H	R	S	X				7	8	4	6	5	3	0	7			
M	9	A	C	E	H	M	R	S	X			8	8	4	6	5	9	3	0	7		
P	10	A	C	E	H	M	P	R	S	X		9	8	4	6	5	9	10	3	0	7	
L	11	A	C	E	H	L	M	P	R	S	X	10	8	4	6	5	11	9	10	3	0	7
E	12	A	C	E	H	L	M	P	R	S	X	10	8	4	12	5	11	9	10	3	0	7
		A	C	E	H	L	M	P	R	S	X		8	4	12	5	11	9	10	3	0	7

Trace of ordered-array ST implementation for standard indexing client

98

## Desempenho de BinarySearchST

- ▶ O método `put()` de `BinarySearchST` faz no máximo  $\lg N + 1$  comparações entre chaves, sendo  $N$  o número de chaves.
- ▶ Aplique `BinarySearchST` a uma lista com  $N$  chaves. Na execução de `get(k)`, uma chave da TS é tocada quando comparada com  $k$ . Na execução de `put(k,v)`, uma chave da TS pode ser tocada em duas ocasiões:
  - quando comparada com  $k$  durante a busca binária,
  - quando é deslocada para a direita para criar espaço para  $(k,v)$ .

99

## Desempenho de BinarySearchST

- ▶ Quantas das  $N$  chaves são tocadas durante um `put()`? quantas durante um `get()`?

	Nr. Máximo de Toques em Comparações	Nr. Máximo de Toques em Deslocamentos
<code>get()</code>	$\lg N$	-
<code>put()</code>	$\lg N$	$N$

- ▶ A operação de busca é, portanto, muito rápida. Mas a operação de inserção é muito lenta no pior caso.

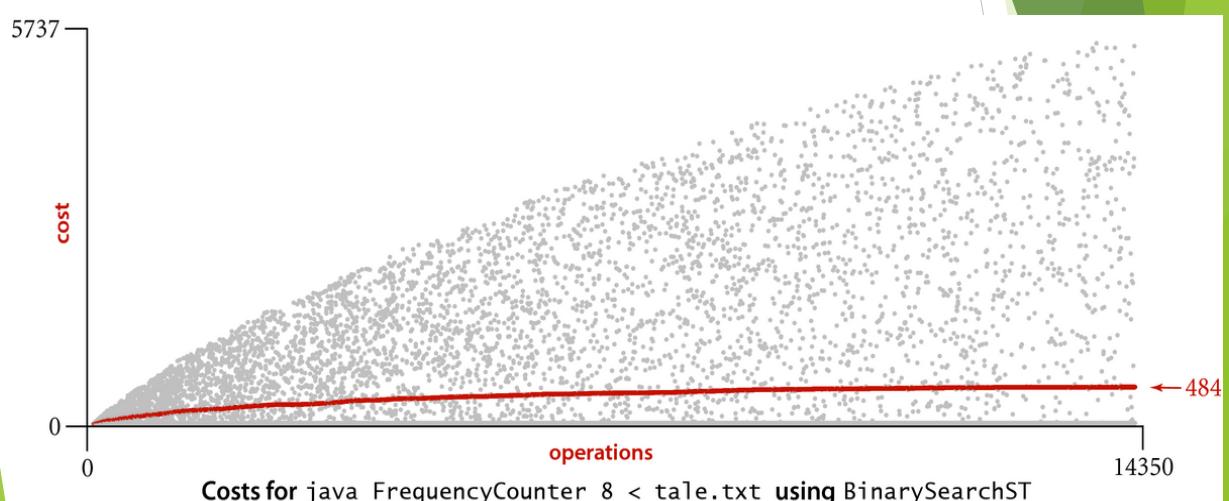
100

## Desempenho de BinarySearchST

- ▶ **Consequência:** Inserir  $N$  chaves distintas numa TS de vetor ordenado inicialmente vazio consome  $N^2$  unidades de tempo no pior caso.
- ▶ O número *médio* de chaves tocadas por `put()` é a metade: apenas  $N/2$ .

101

Nr. de chaves tocadas por cada chamada de `put()`.  
Os pontos vermelhos dão a média corrente.



102

## Custos das Implementações Básicas: Resumo

Algoritmo	PIOR CASO (tabela com N chaves)		CASO MÉDIO (tabela com N chaves aleatórias)		Operações ordenadas eficientes?
	busca	insere	busca (acerto)	insere	
Busca Sequencial (lista ligada)	N	N	N/2	N	não
Busca binária (vetor ordenado)	$\lg N$	N	$\lg N$	N/2	sim

103

## Operações adicionais em TS

### ► Operação de Remoção

<code>void delete(Key key)</code>	remove chave key e o correspondente valor
-----------------------------------	---

### ► Operações em TSs ordenadas

<code>Key min()</code>	a menor chave desta TS
<code>Key max()</code>	a maior chave desta TS
<code>Key floor(Key key)</code>	o piso de key, ou seja, a maior chave $\leq$ key
<code>Key ceiling(Key key)</code>	o teto de key, ou seja, a menor chave $\geq$ key
<code>Key select(int k)</code>	chave cujo <u>posto</u> é k
<code>void deleteMin()</code>	remove a menor chave (e o valor associado)
<code>void deleteMax()</code>	remove a maior chave (e o valor associado)

104

## Custos das operações em BinarySearchST para tabela com $N$ chaves:

MÉTODO	ordem de crescimento do tempo de execução
delete()	$N$
min()	1
max()	1
floor()	$\log N$
ceiling()	$\log N$
rank()	$\log N$
select()	1
deleteMin()	$N$
deleteMax()	1