

Size Balanced Tree

中山纪念中学

高二 陈启峰

344368722@QQ.com

总揽全文

1. **BST & Rotations** : 预备知识
2. **Size Balanced Tree** : 定义 & 功能介绍
3. **Maintain** : 核心操作
4. **Analysis** : 时间复杂度分析
5. **Advantage** : 七大优点
6. **探索历程** : 感性与理性中螺旋前进

Binary Search Tree

- Binary Search Tree (abbr. BST) 是一棵具有以下性质的二叉树：
对于 BST 中任意一个结点，
 - (1) 左子树中的关键字不大于它的关键字；
 - (2) 右子树中的关键字不小于它的关键字。

Binary Search Tree

■ 为了方便讨论我们定义：

1. **Left [T]** : 结点 T 的左儿子

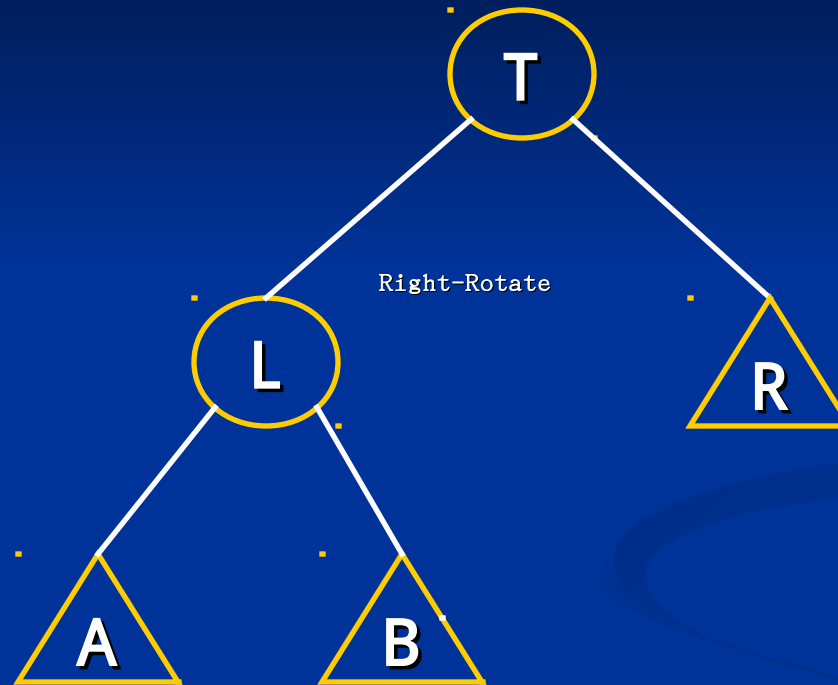
2. **Right [T]** : 结点 T 的右儿子

3. **S [T]** : 以 T 为根的子树的结点个数 (大小)

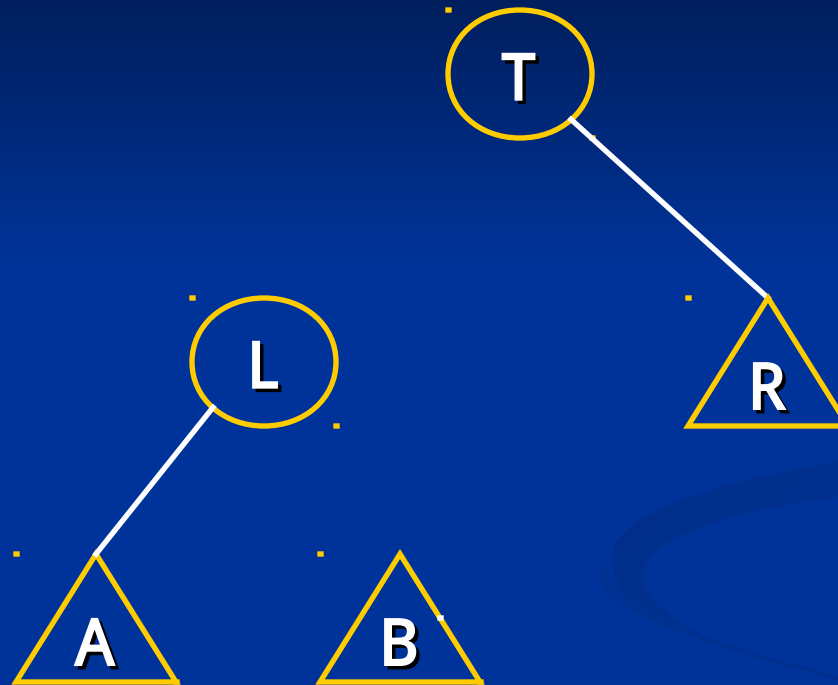
Rotations

1. 为了保持 BST 平衡，我们通常使用 Rotations 来改变树的形态。
2. Rotations 分为相对的两类型：
Right-Rotate
Left-Rotate

Rotations

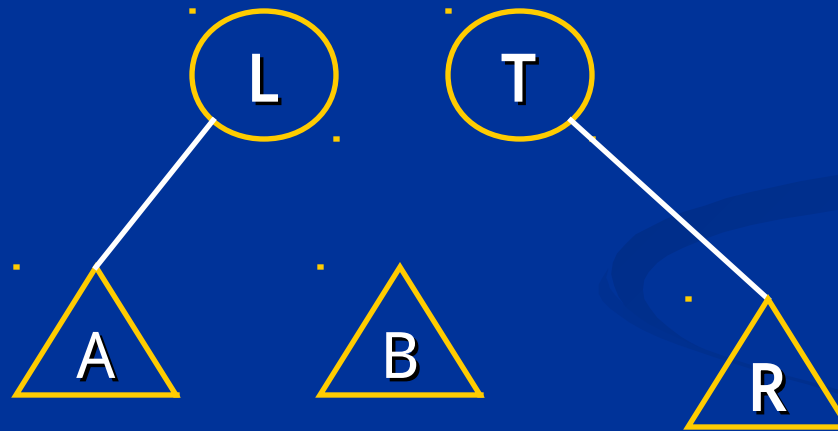


Rotations



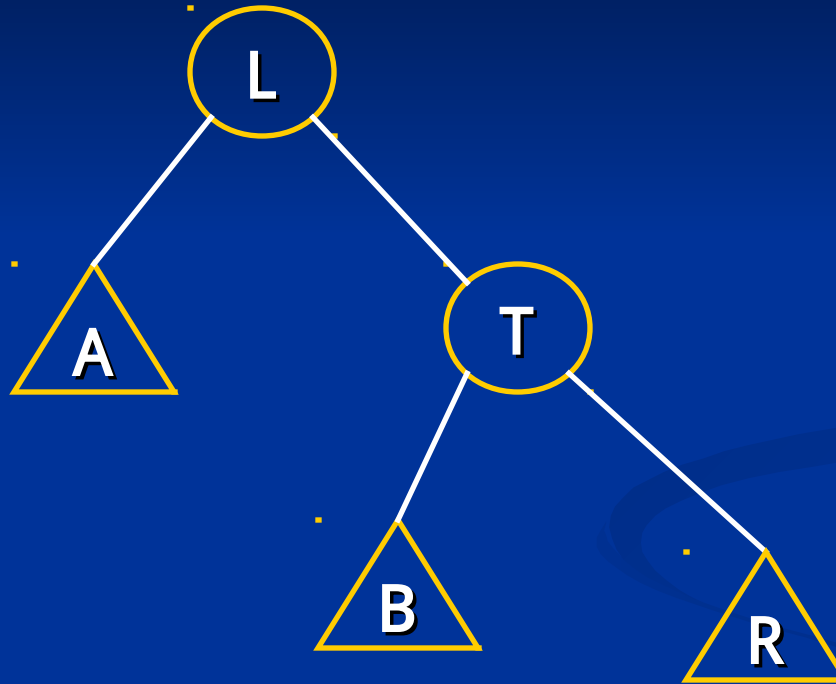
Right-Rotate

Rotations



Right-Rotate

Rotations



Right-Rotate

Size Balanced Tree

- Size Balanced Tree (abbr. SBT) 是一种通过大小来保持平衡的 BST。它总是满足：

对于 SBT 的每一个结点 t ,

性质 (a)

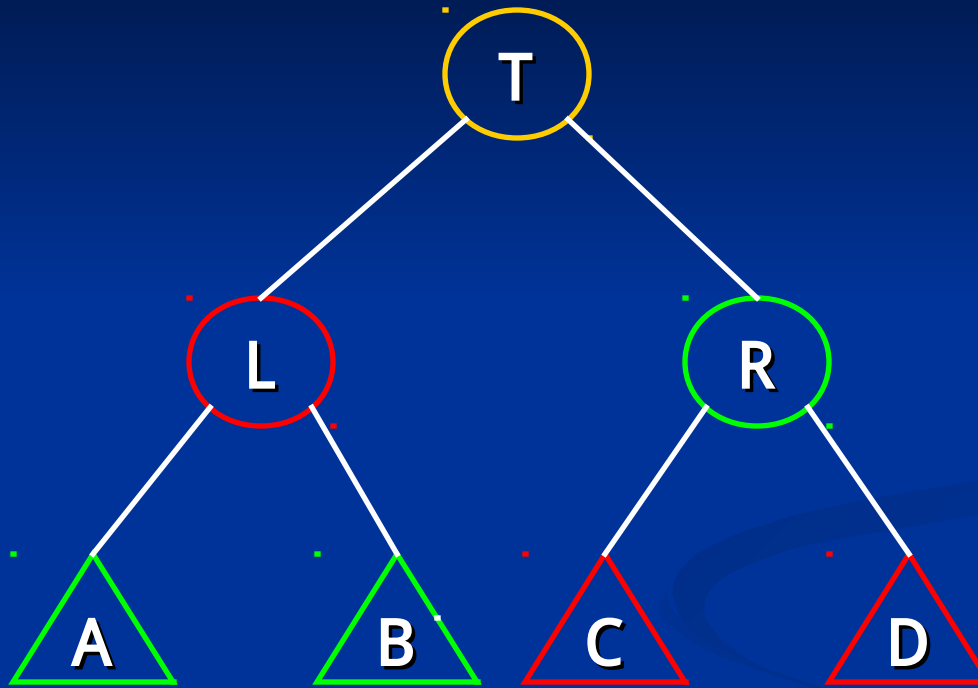
$$s[\text{right}[t]] \geq s[\text{left}[\text{left}[t]]], \\ s[\text{right}[\text{left}[t]]]$$

性质 (b)

$$s[\text{left}[t]] \geq s[\text{right}[\text{right}[t]]], \\ s[\text{left}[\text{right}[t]]]$$

i.e. 每棵子树的大小不小于其兄弟的子树大小

Size Balanced Tree



$$s[R] \geq s[A] , s[B]$$

$$s[L] \geq s[C] , s[D]$$

Size Balanced Tree

Insert	插入
Delete	删除
Find	查找
Rank	排名
Select	找第 K 小
Pred	前趋
Succ	后继

- 当我们插入或删除一个结点后，S B T的大小就发生了改变。

Maintain

- 这种改变有可能导致性质 (a) 或 (b) 被破坏。
- 这时，我们需要修复这棵树。

Maintain

- `Maintain(T)` 用于修复以 `T` 为根的 SBT。
- 调用 `Maintain(T)` 的前提条件是 `T` 的子树都已经是 SBT 了

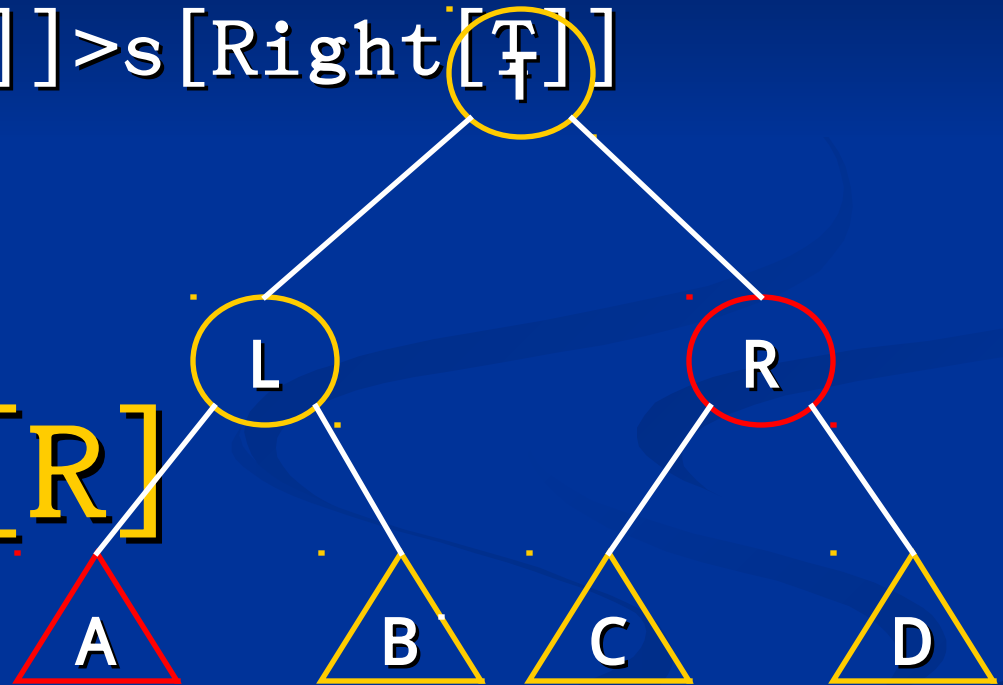
Maintain

- 由于性质 (a) 和 (b) 是对称的，下面仅对性质 (a) 被破坏的情况进行分析：

Maintain

- Case 1:
 $s[\text{Left}[\text{Left}[T]]] > s[\text{Right}[T]]$

$$S[A] > S[R]$$

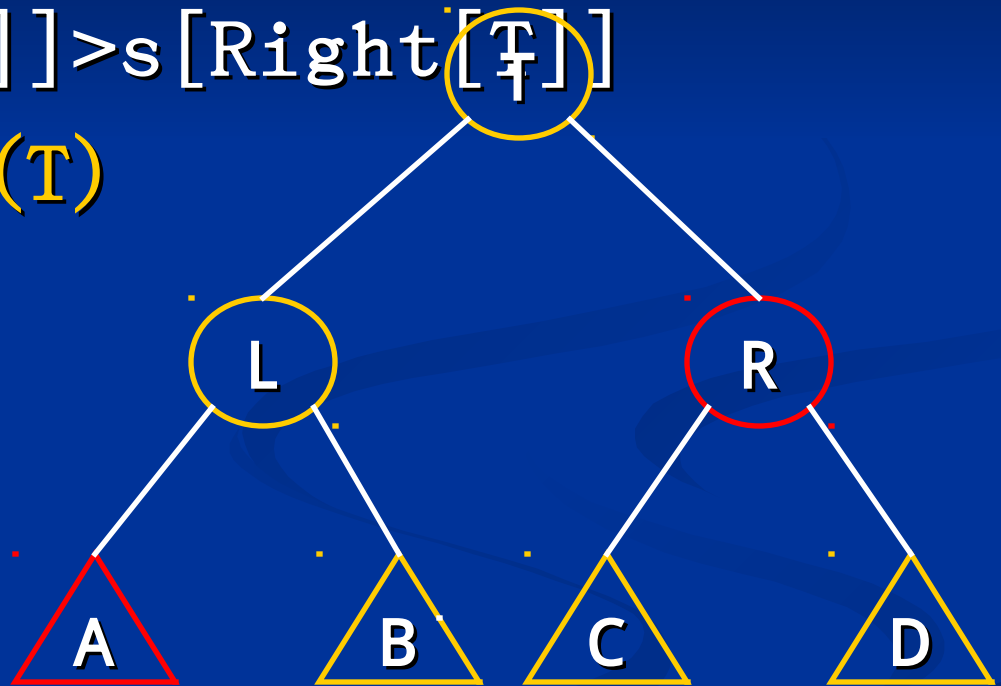


Maintain

- Case 1:

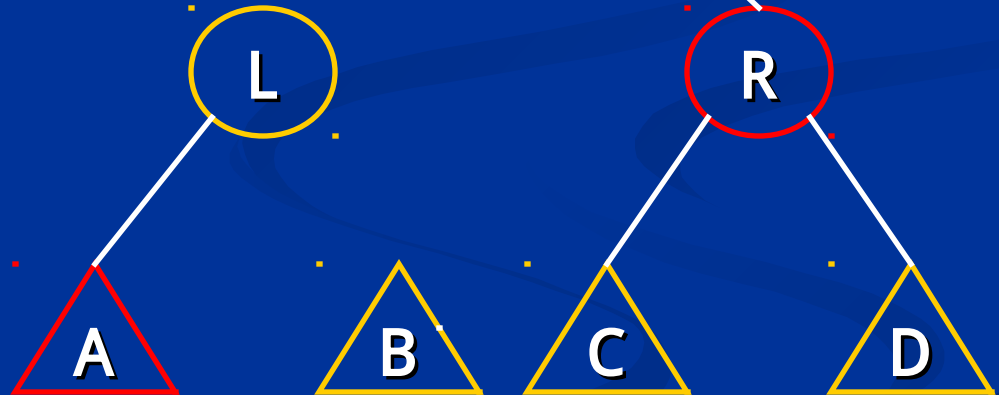
$s[\text{Left}[\text{Left}[T]]] > s[\text{Right}[T]]$

1. **Right-Rotate** (T)



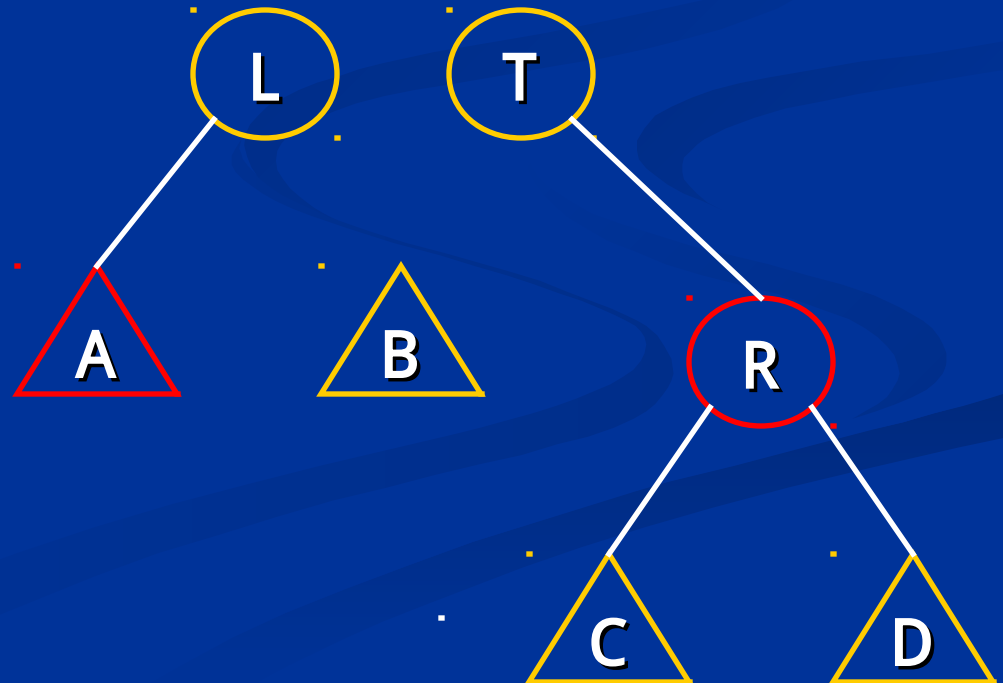
Maintain

- Case 1:
 $s[\text{Left}[\text{Left}[T]]] > s[\text{Right}[T]]$
 1. **Right-Rotate** (T)



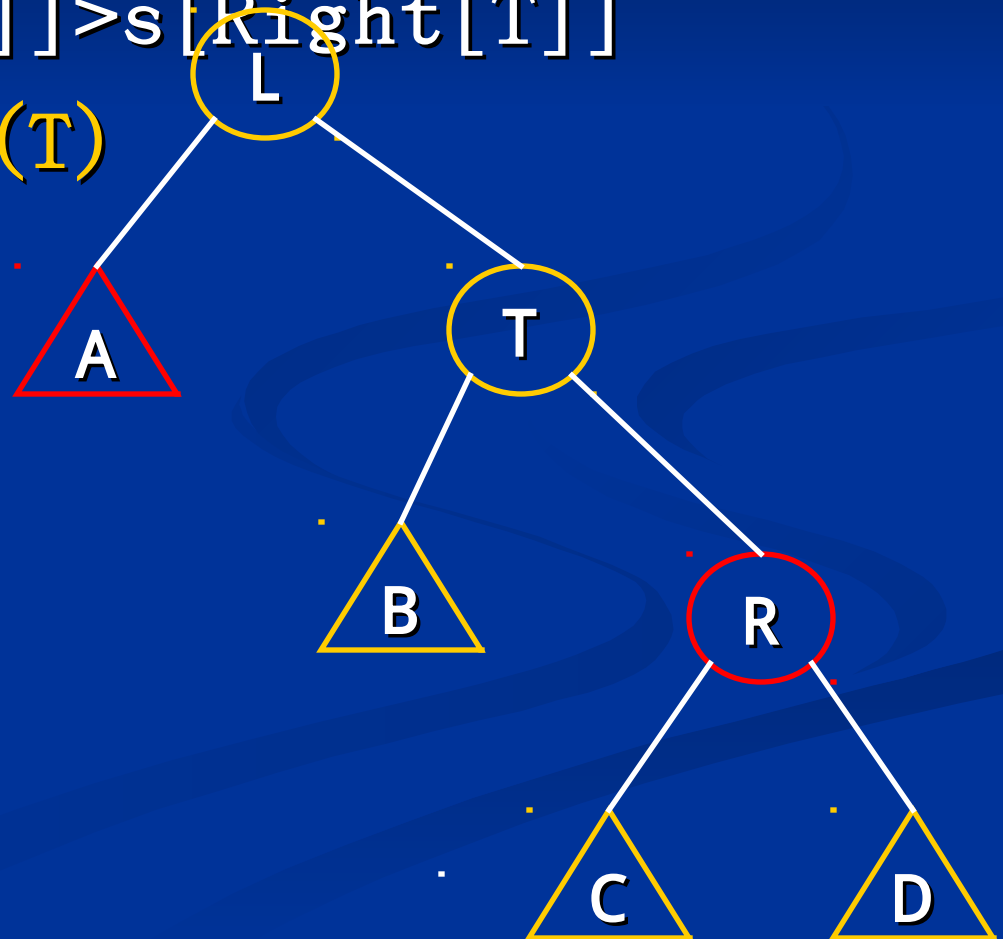
Maintain

- Case 1:
 $s[\text{Left}[\text{Left}[T]] > s[\text{Right}[T]]$
 1. **Right-Rotate** (T)



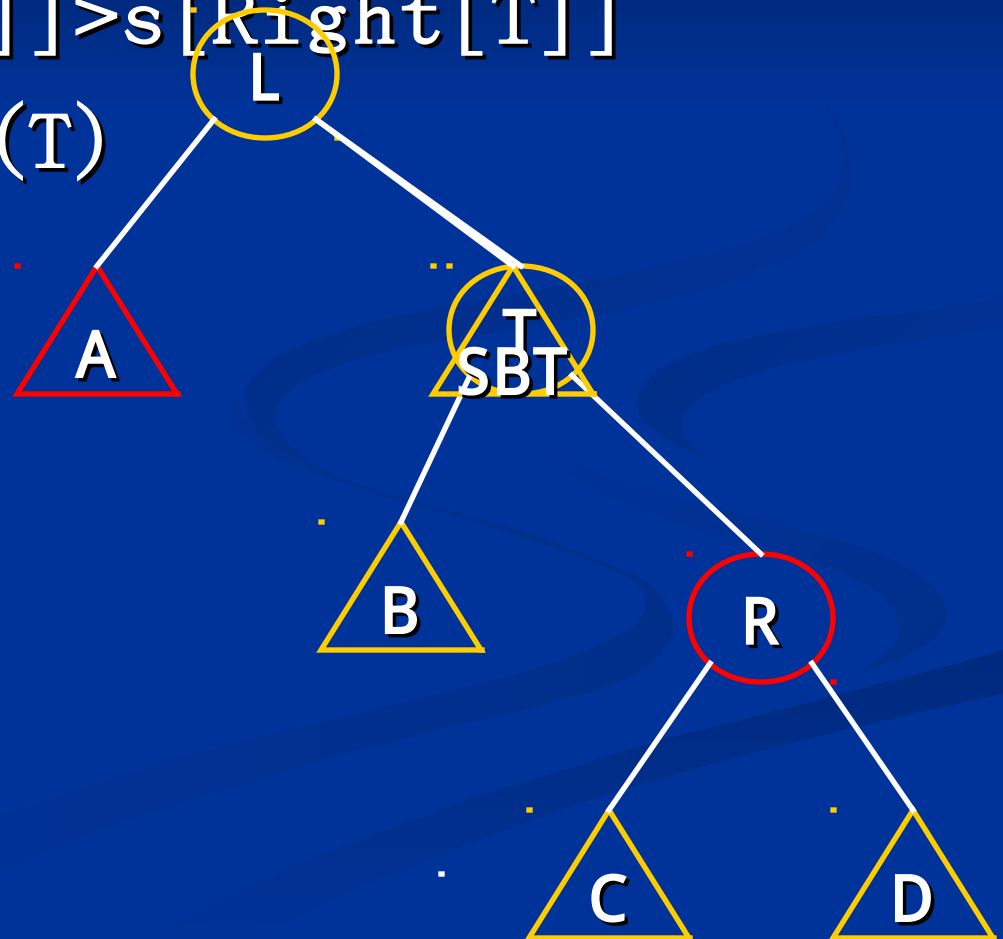
Maintain

- Case 1:
 $s[\text{Left}[\text{Left}[T]]] > s[\text{Right}[T]]$
 1. **Right-Rotate** (T)



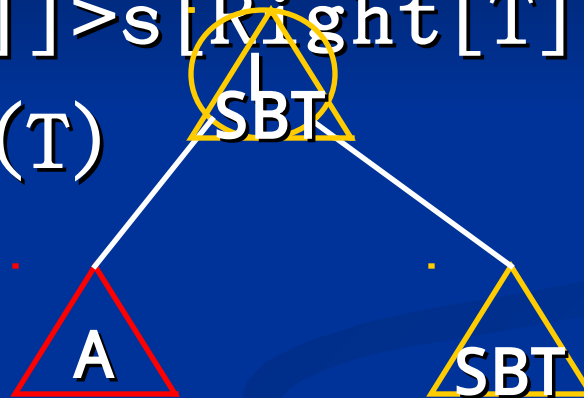
Maintain

- Case 1:
 $s[\text{Left}[\text{Left}[T]]] > s[\text{Right}[T]]$
 1. Right-Rotate (T)
 2. **Maintain** (T)



Maintain

- Case 1:
 $s[\text{Left}[\text{Left}[T]]] > s[\text{Right}[T]]$
- 1. Right-Rotate (T)
- 2. Maintain (T)
- 3. **Maintain (L)**

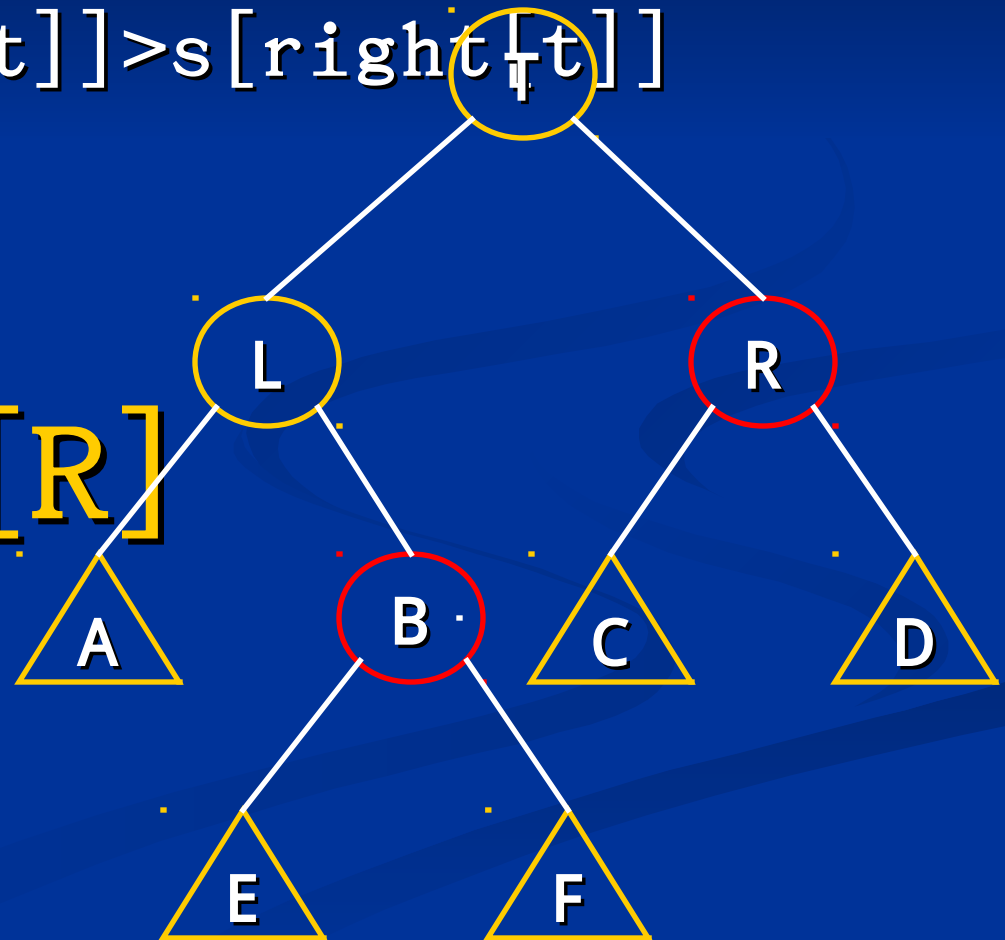


Maintain

- Case 2:

$$s[\text{right}[\text{left}[t]]] > s[\text{right}[t]]$$

$$s[B] > s[R]$$

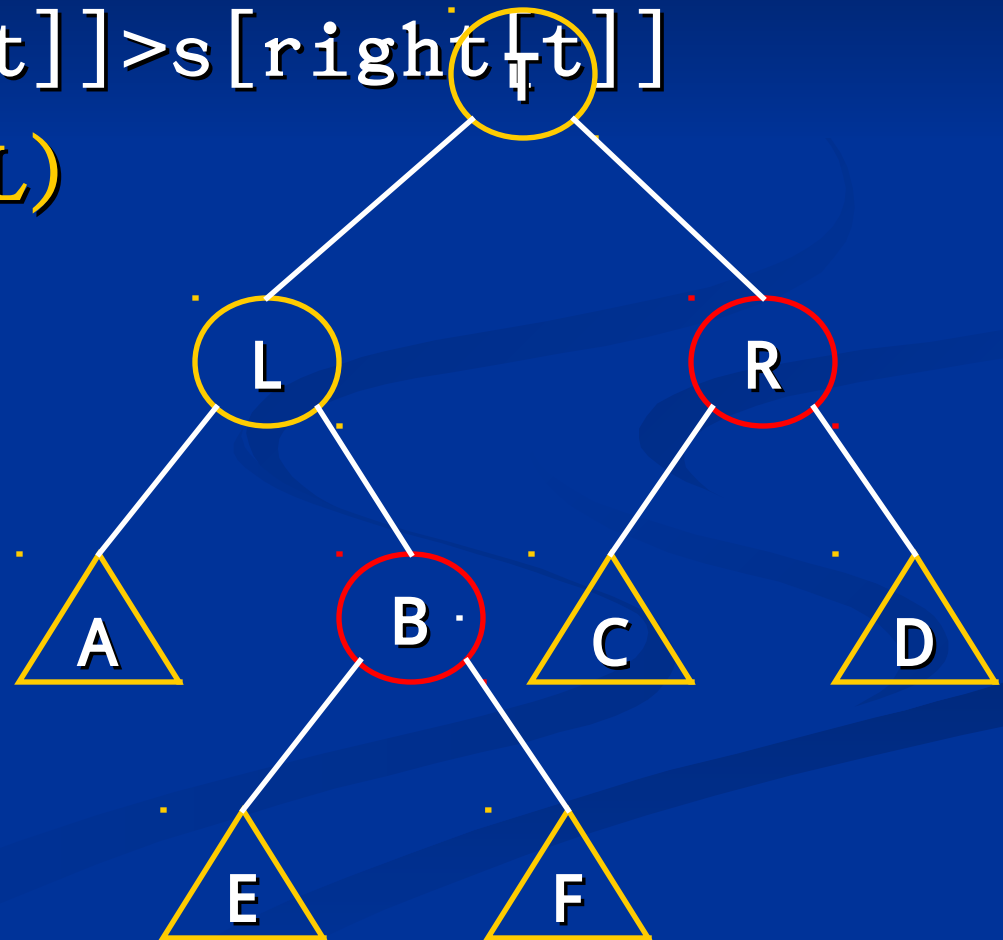


Maintain

- Case 2:

$s[\text{right}[\text{left}[t]]] > s[\text{right}[t]]$

1. **Left-Rotate (L)**

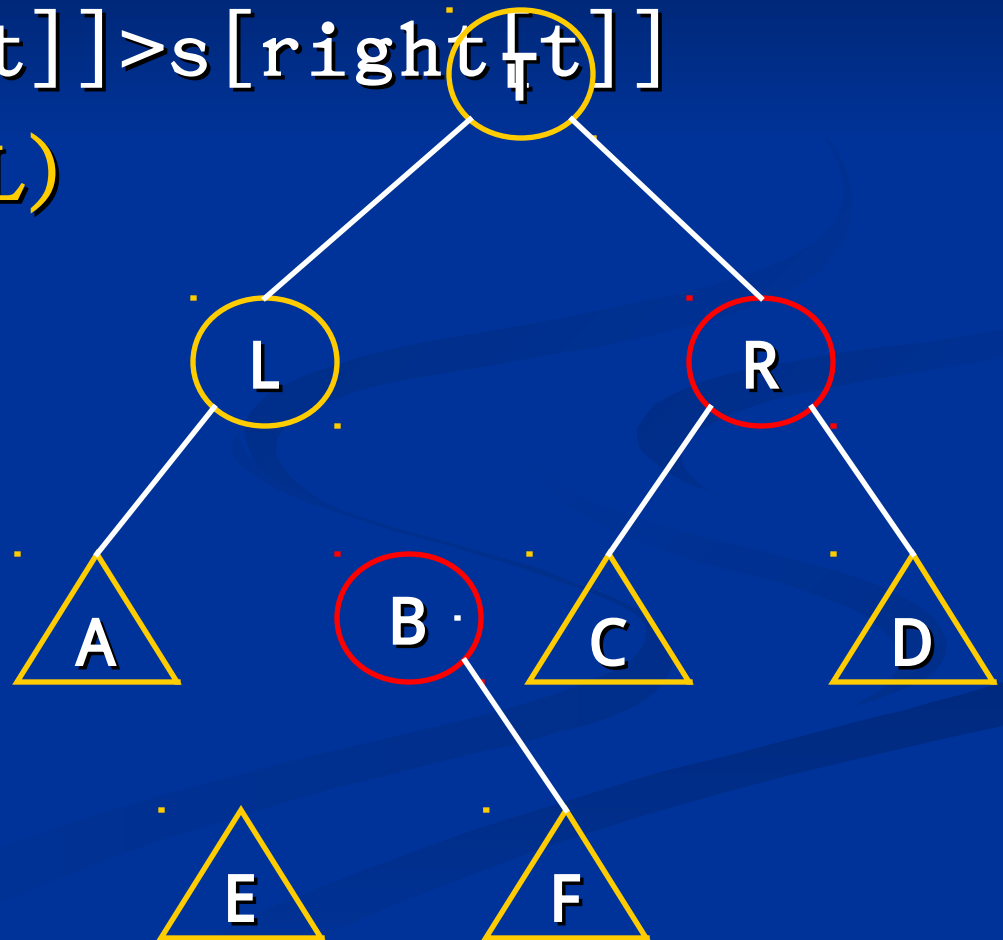


Maintain

- Case 2:

$$s[\text{right}[\text{left}[t]]] > s[\text{right}[t]]$$

1. **Left-Rotate (L)**

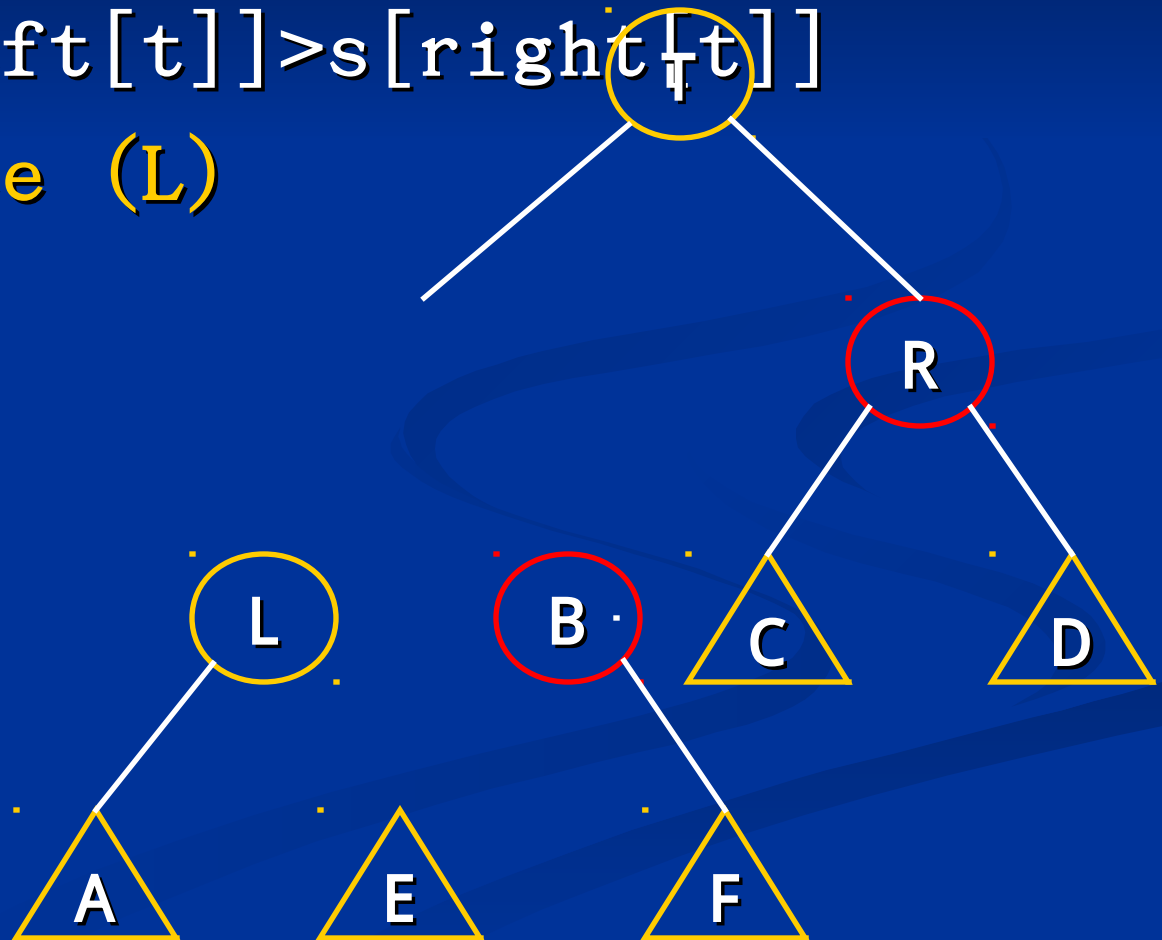


Maintain

- Case 2:

$s[\text{right}[\text{left}[t]]] > s[\text{right}[t]]$

1. Left-Rotate (L)

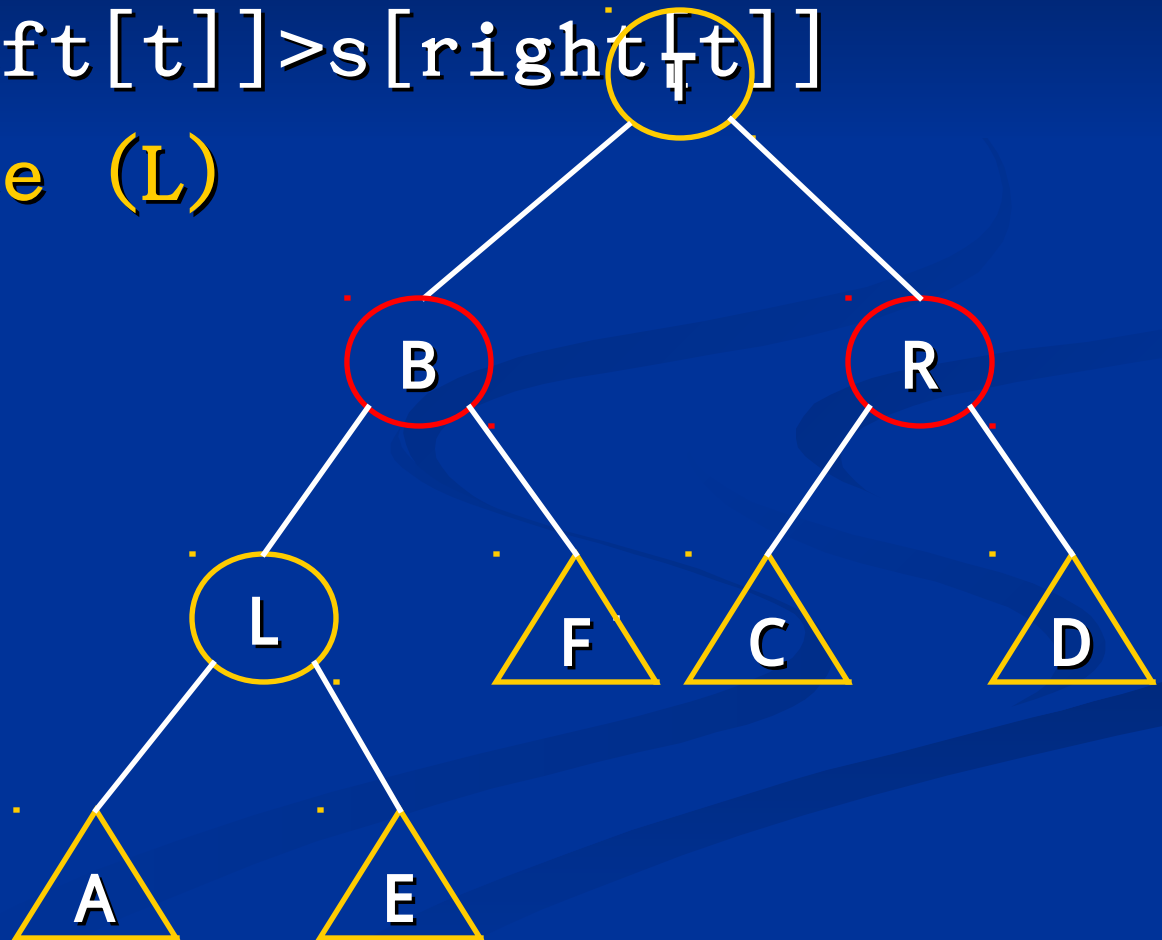


Maintain

- Case 2:

$s[\text{right}[\text{left}[t]]] > s[\text{right}[t]]$

1. **Left-Rotate (L)**

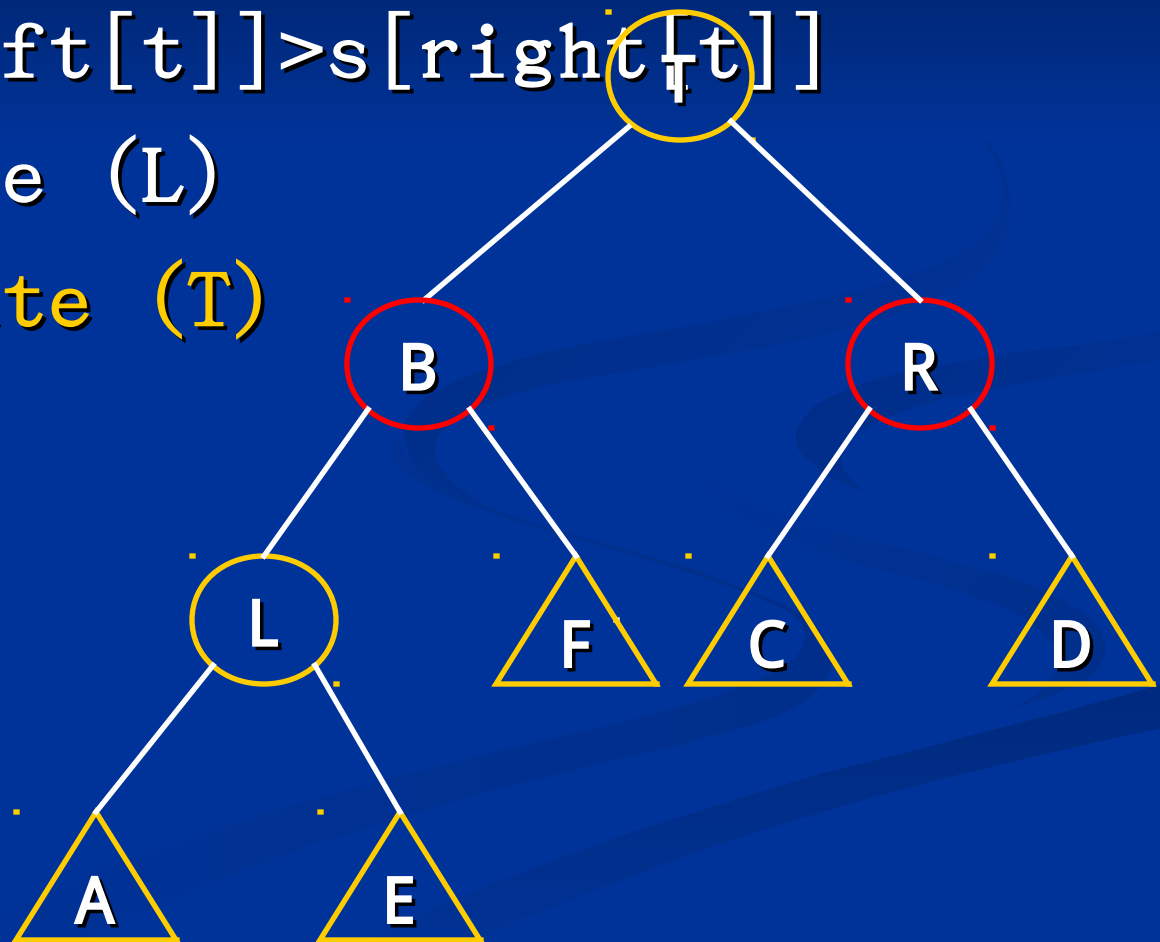


Maintain

■ Case 2:

$s[\text{right}[\text{left}[t]]] > s[\text{right}[t]]$

1. Left-Rotate (L)
2. Right-Rotate (T)

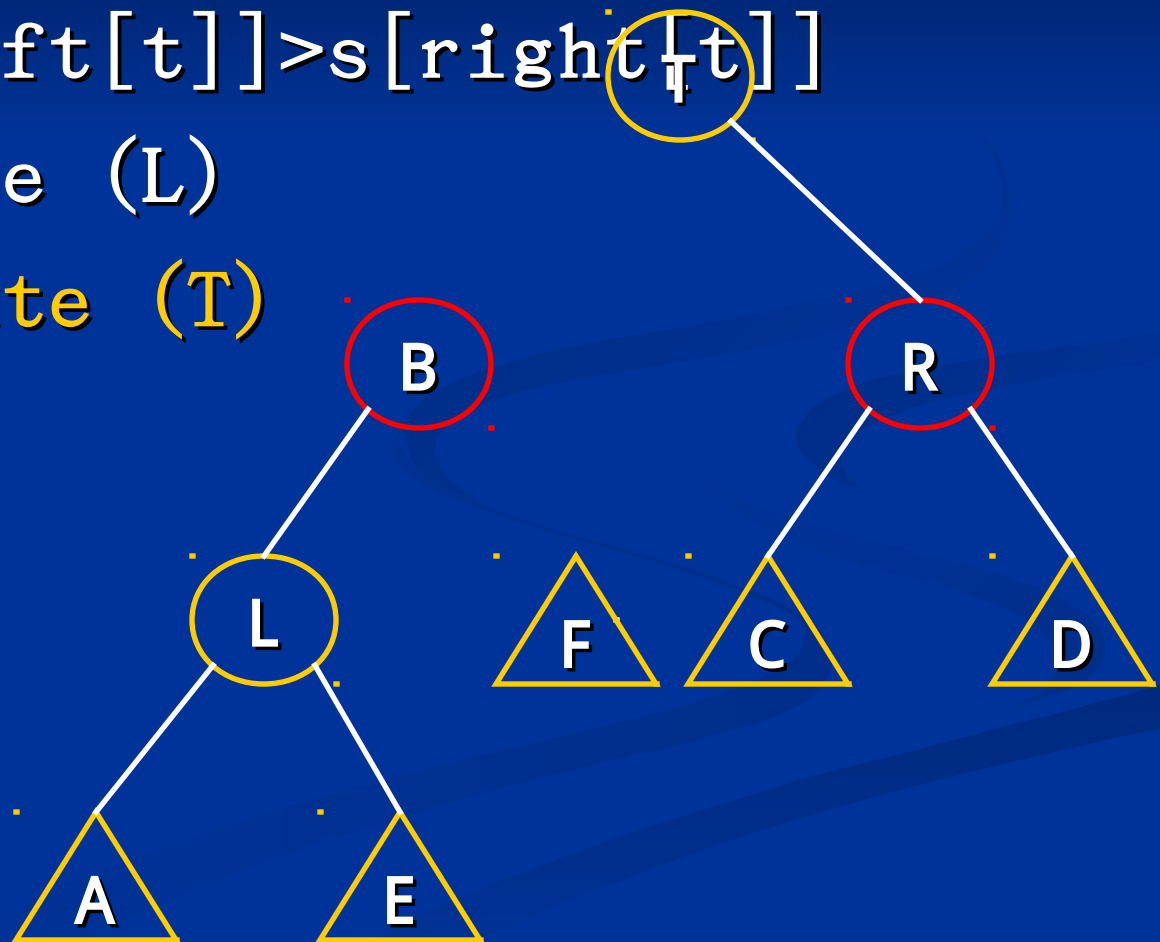


Maintain

■ Case 2:

$$s[\text{right}[\text{left}[t]]] > s[\text{right}[t]]$$

1. Left-Rotate (L)
2. Right-Rotate (T)

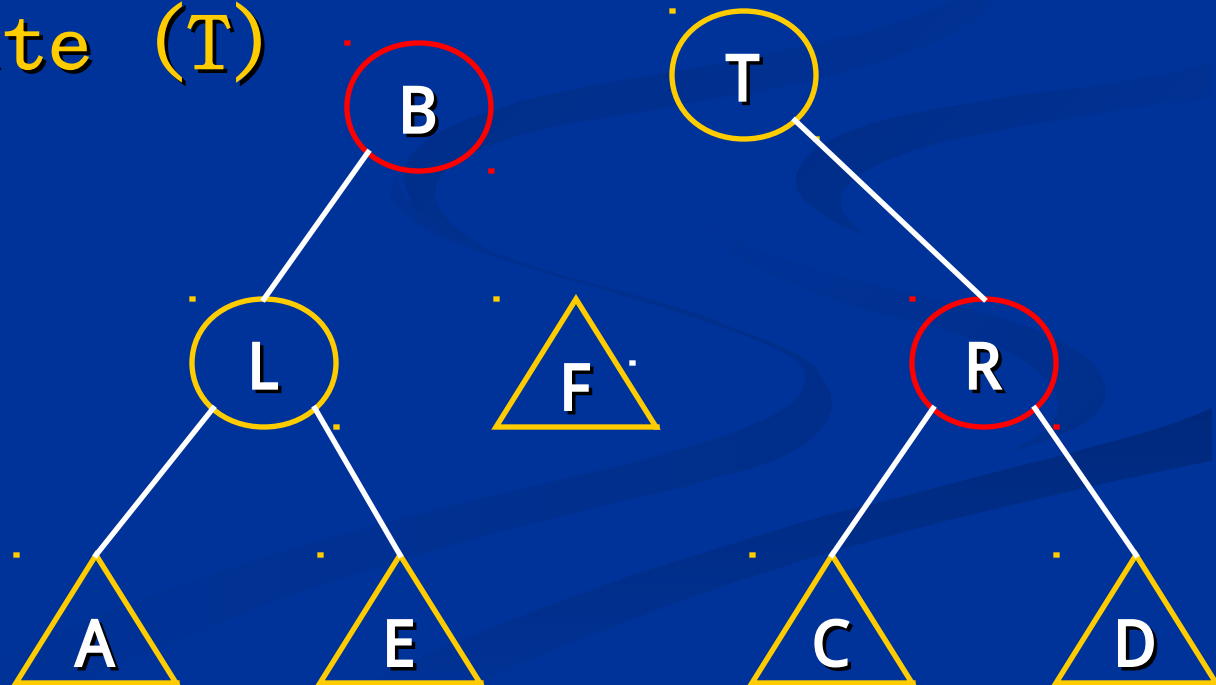


Maintain

- Case 2:

$s[\text{right}[\text{left}[t]]] > s[\text{right}[t]]$

1. Left-Rotate (L)
2. Right-Rotate (T)



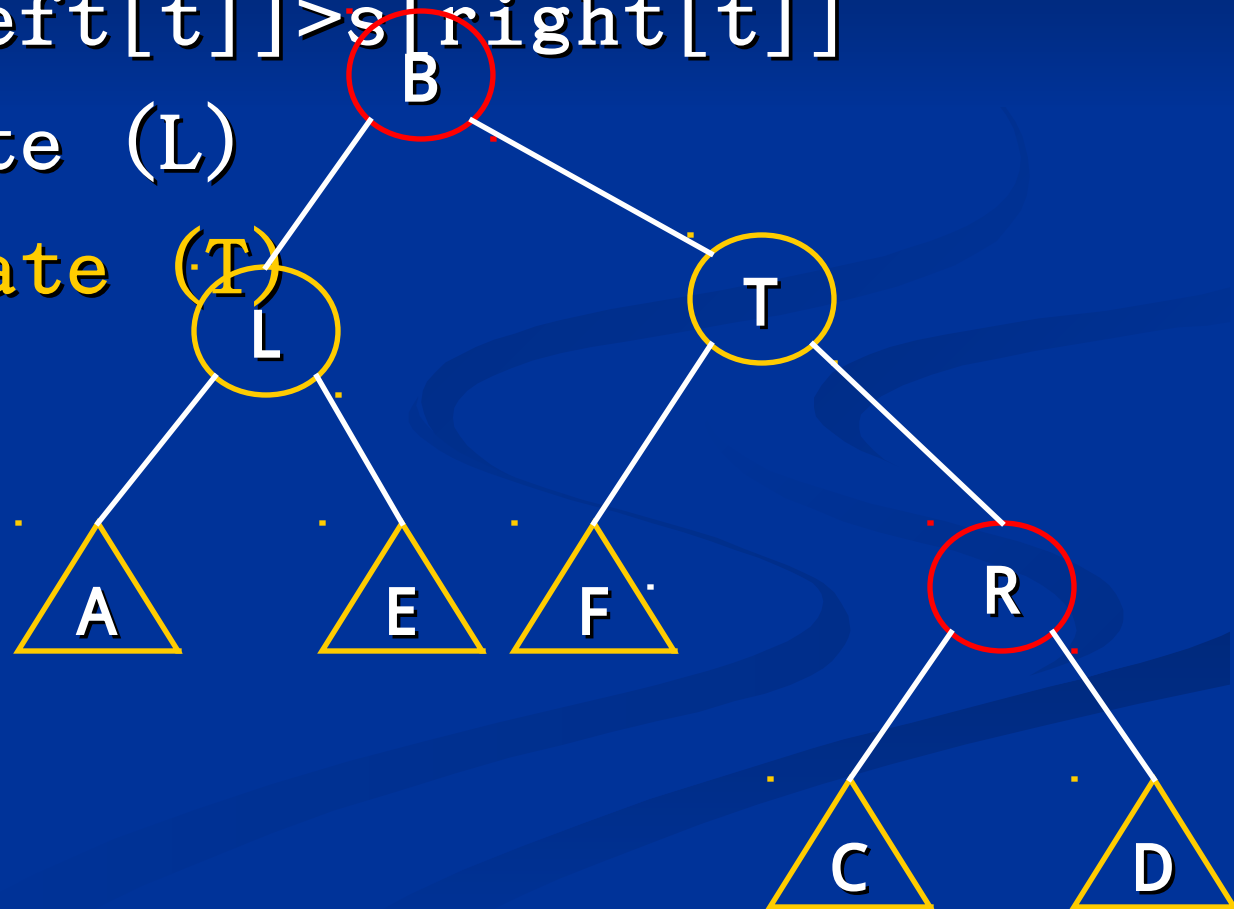
Maintain

■ Case 2:

$s[\text{right}[\text{left}[t]]] > s[\text{right}[t]]$

1. Left-Rotate (L)

2. Right-Rotate (R)



Maintain

■ Case 2:

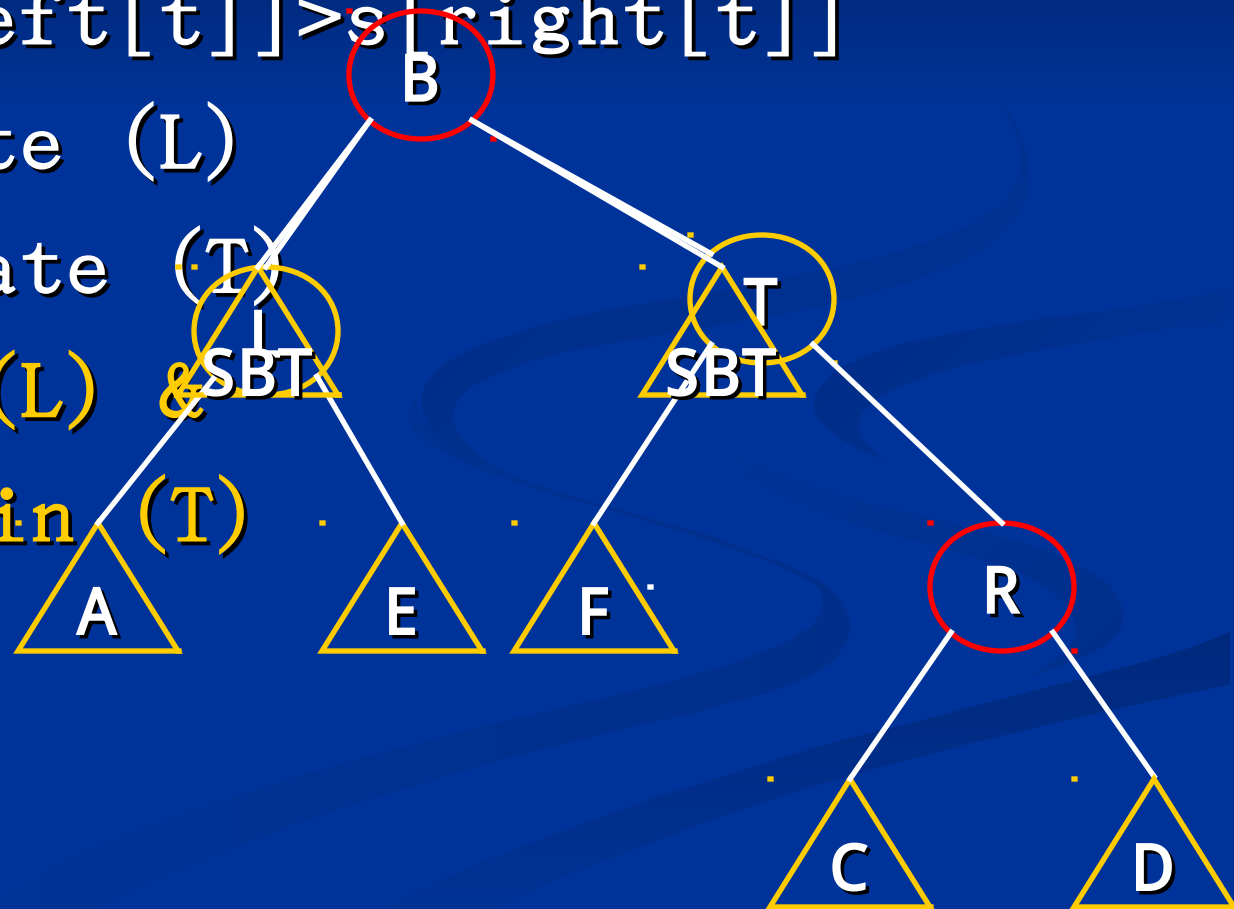
$s[\text{right}[\text{left}[t]]] > s[\text{right}[t]]$

1. Left-Rotate (L)

2. Right-Rotate (R)

3. Maintain (L) & SBT

Maintain (R)



Maintain

■ Case 2:

$s[\text{right}[\text{left}[t]]] > s[\text{right}[t]]$

1. Left-Rotate (L)

2. Right-Rotate (R)

3. Maintain (L) &

Maintain (R)

4. Maintain (B)



Maintain

- 通常我们可以确保性质 (a) 或 (b) 已经被满足了。
- 为了提高效率我们可以增加一个布尔型参数 **flag** 来去除无意义的检查。

Maintain

```
1.  If flag=false then
2.      If s[left[left[t]]>s[right[t]] then           //case 1
3.          Right-Rotate(t)                           //case 1
4.      Else if s[right[left[t]]>s[right[t]] then //case 2
5.          Left-Rotate(left[t])                       //case 2
6.          Right-Rotate(t)                           //case 2
7.      Else exit
      //needn' t repair
8.  Else if s[right[right[t]]>s[left[t]] then //case 1'
9.      Left-Rotate(t)                                //case
1'
10. Else if s[left[right[t]]>s[left[t]] then //case 2'
11.     Right-Rotate(right[t])                       //case 2'
12.     Left-Rotate(t)                               //case
2'
13. Else exit
    //needn' t repair
```

Maintain

- 14. `Maintain(left[t],false)` `//repair the`
 left subtree
- 15. `Maintain(right[t],true)` `//repair the`
 right subtree
- 16. `Maintain(t,false)` `//repair the`
 whole tree
- 17. `Maintain(t,true)` `//repair the`
 whole tree

Maintain(t,flag)

Analysis

- **Analysis Of Height**
- **F[H]**——高度为 **H** 的 **SBT** 最少的结点数。

$$\mathbf{F[H] = Fibonacci[H+2] - 1}$$

H	13	17	21	25	29
F[H]	986	6764	46367	317810	2178308

Analysis

- **Analysis Of Height**

- 定理:

N 个结点的 SBT 的最坏深度为最大的 H 且满足 $F[H] \leq N$ 。

因此

$$\text{Max_Height}[N] \leq 1.441 \log_2(N+1.5) - 1.33$$

$$\text{Height} = O(\log n)$$

Analysis

- **Analysis Of Maintain**
- 设 **SD** 为所有结点的深度之和

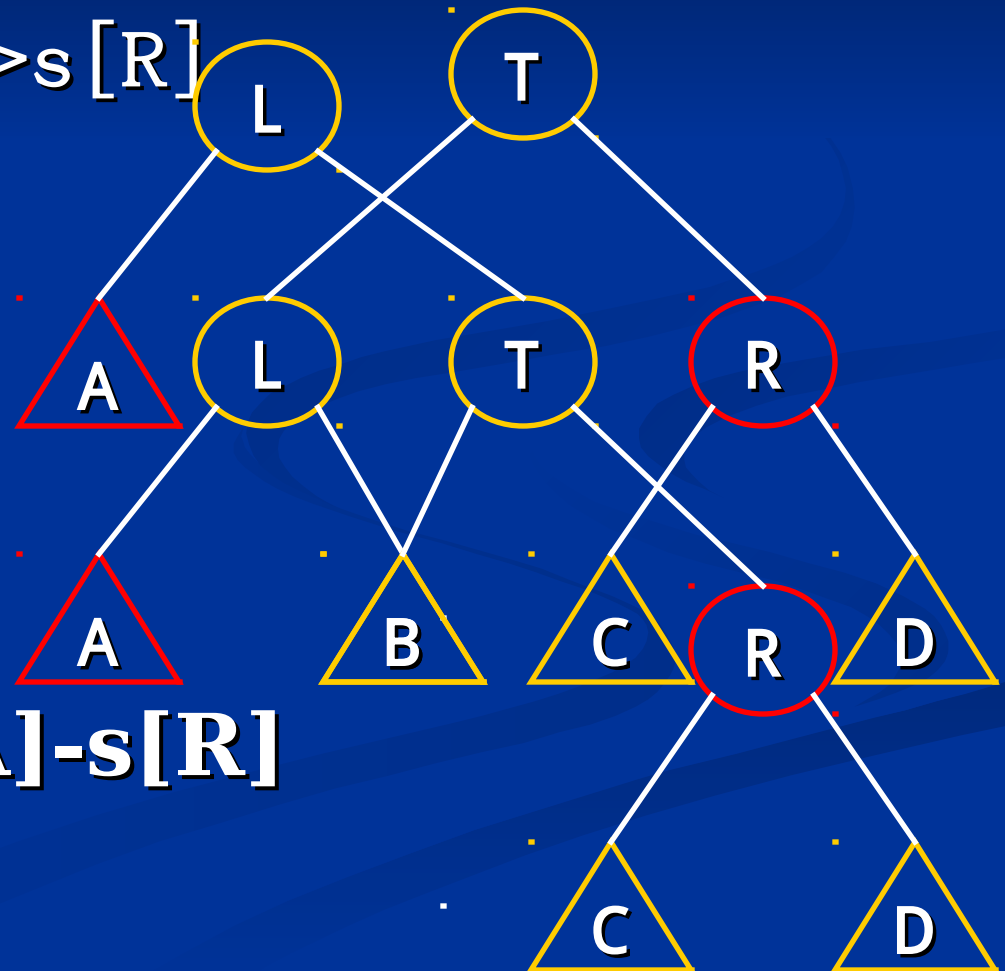
结论 (1)

在 **Maintain** 的旋转后 **SD** 总是递减

Analysis

- **Analysis Of Maintain**

- Case 1 : $s[A] > s[R]$

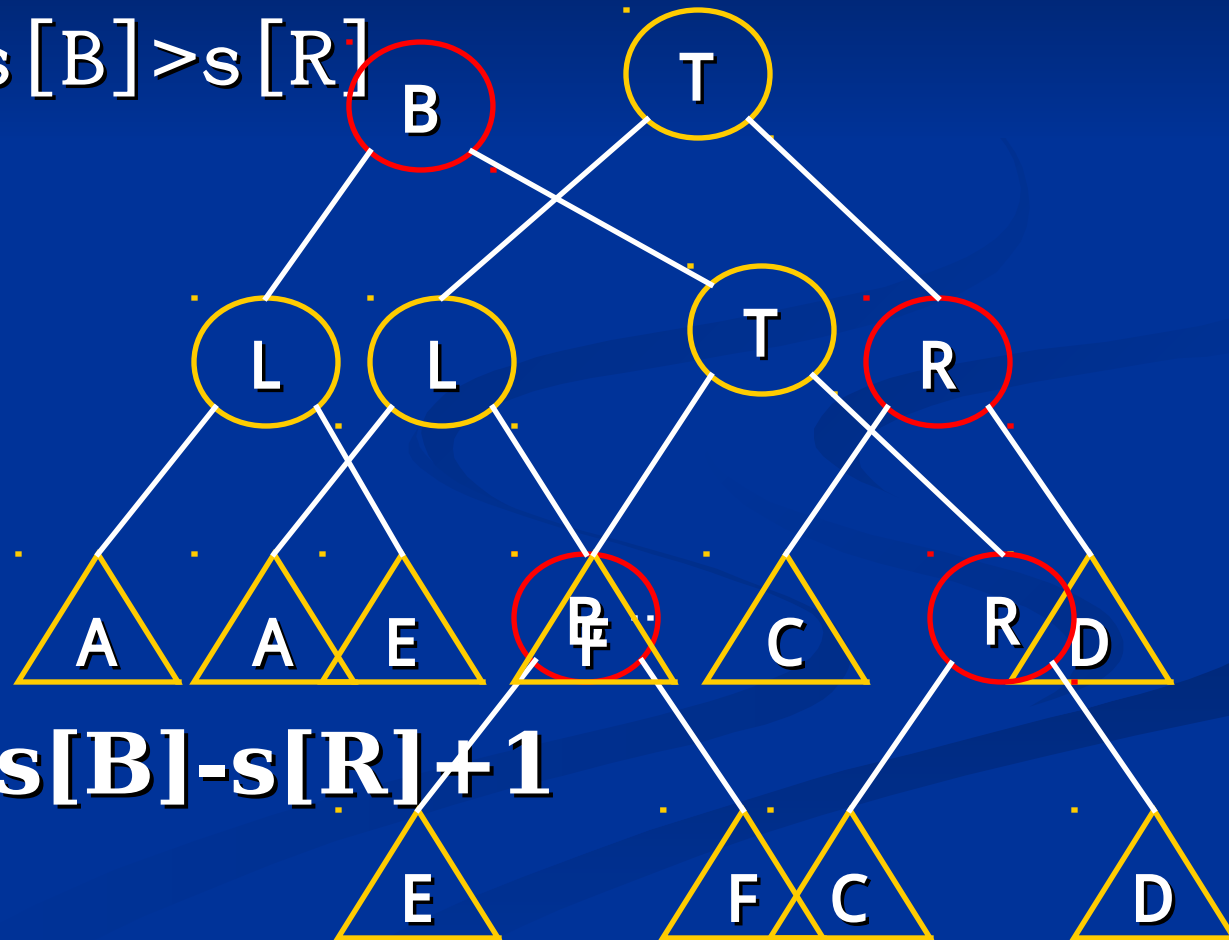


- **SD 減少了 $s[A]-s[R]$**
(正整数)

Analysis

- **Analysis Of Maintain**

- Case 2 : $s[B] > s[R]$



- **SD 减少了 $s[B] - s[R] + 1$**
(正整数)

Analysis

■ Analysis Of Maintain

结论 (2)

SD 总保持在 $O(n \log n)$

■ 综合结论 (1) (2) 我们得到

Maintain 的摊平时间复杂度 = $O(1)$

Analysis

■ Analysis Of Operations

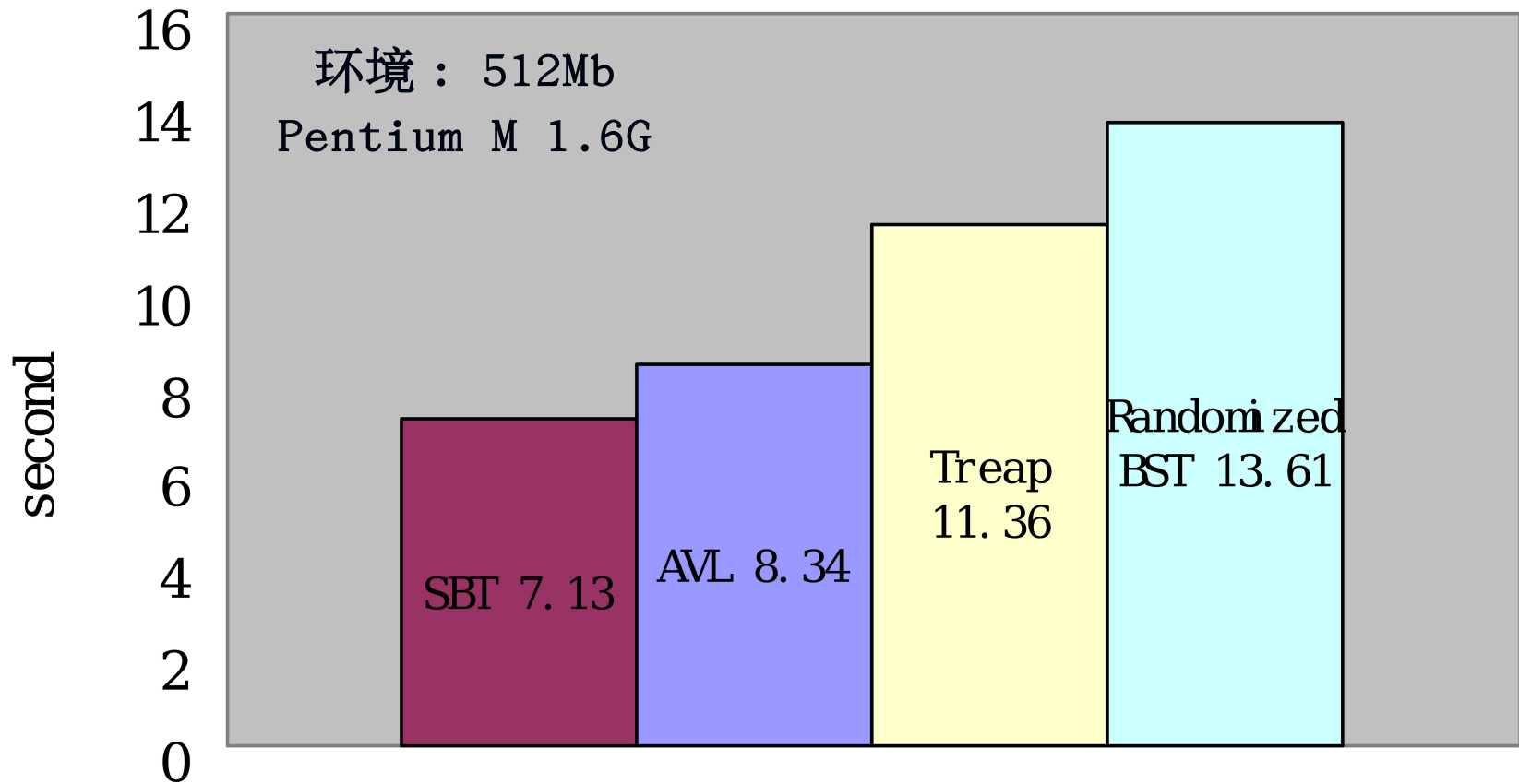
Insert	$O(\log n)$
Delete	$O(\log n)$
Find	$O(\log n)$
Rank	$O(\log n)$
Select	$O(\log n)$
Pred	$O(\log n)$
Succ	$O(\log n)$

Advantage

- 七大优点
- 1. 速度快

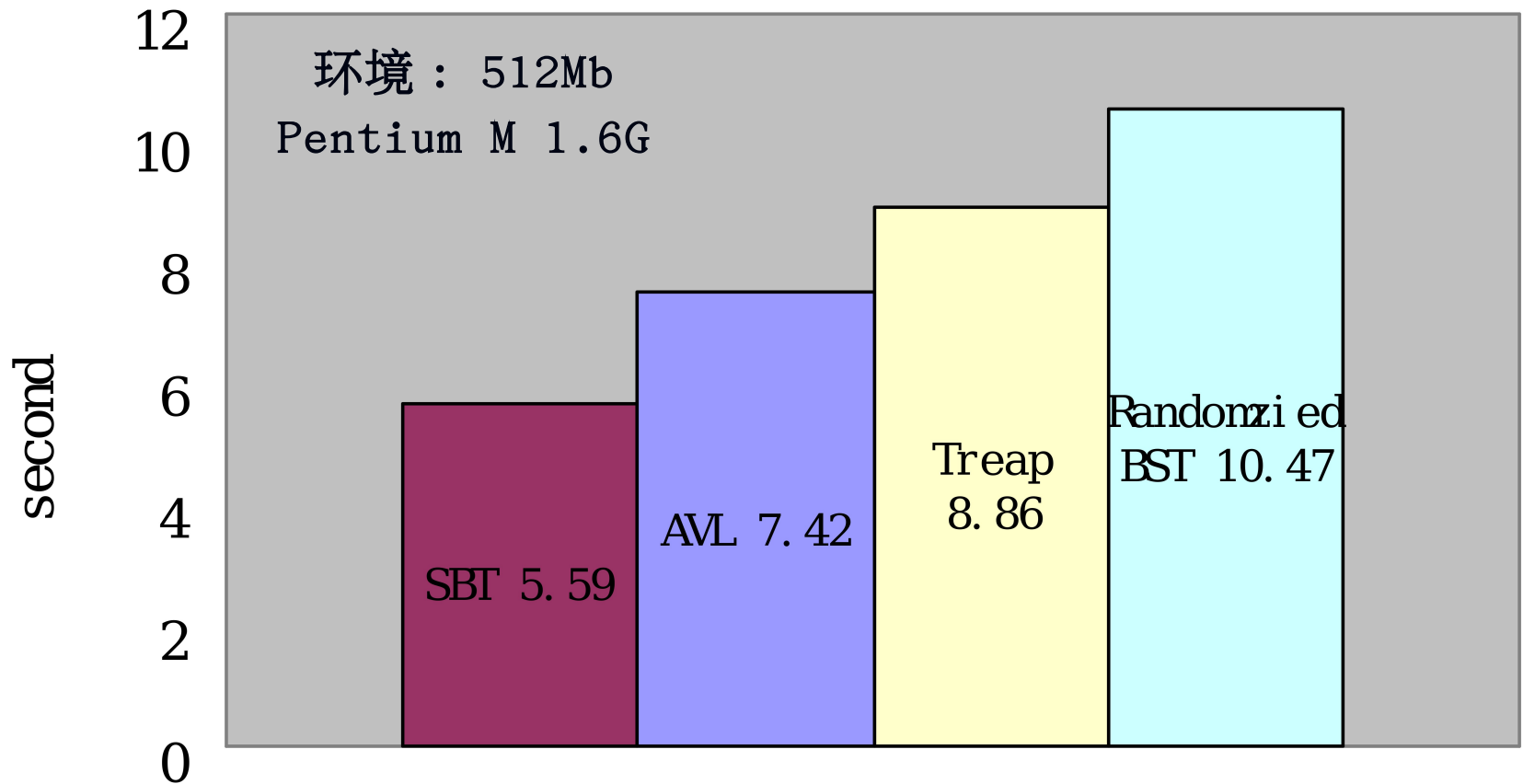
Advantage

Insert 2,000,000 nodes with random values



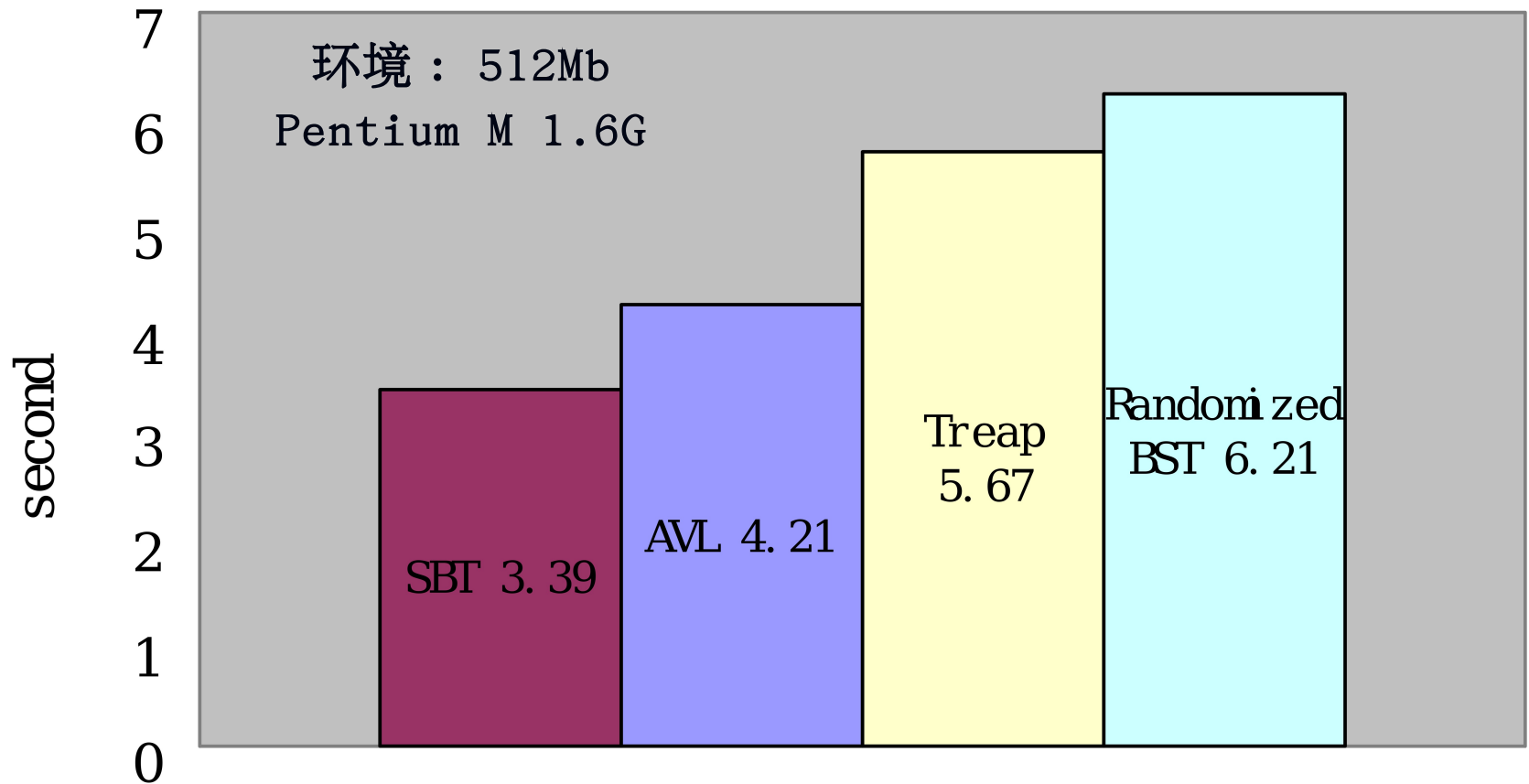
Advantage

Perform 2,000,000 operations of 66% insertion and 33% deletion with random values



Advantage

Perform 2,000,000 operations of 20% insertion, 10% deletion and 70% query with random values



Advantage

- 七大优点
- 1. 速度快
- 2. 性能高

Advantage

- 插入 2,000,000 个关键字随机的结点

类型	SBT	AVL	Treap	Random ized BST	Splay	Perfec t BST
平均深度	19.24	19.33	26.51	25.53	37.20	18.95
高度	24	24	50	53	78	20
旋转次数	157 万	140 万	399 万	400 万	2515 万	?

Advantage

- 插入 2,000,000 个关键字有序的结点

类型	SBT	AVL	Treap	Random ized BST	Splay	Perfec t BST
平均深度	18.95	18.95	25.7	26.29	999999.5	18.95
高度	20	20	51	53	1999999	20
旋转次数	200 万	200 万	200 万	200 万	0	?

Advantage

- 七大优点
- 1. 速度快
- 2. 性能高
- 3. 调试易

Advantage

■ 七大优点

1. 速度快
2. 性能高
3. 调试易
4. 代码短

Advantage

■ 七大优点

1. 速度快
2. 性能高
3. 调试易
4. 代码短
5. 空间少

Advantage

■ 七大优点

1. 速度快
2. 性能高
3. 调试易
4. 代码短
5. 空间少
6. 功能强

Advantage

■ 七大优点

1. 速度快
2. 性能高
3. 调试易
4. 代码短
5. 空间少
6. 功能强
7. 应用广

Advantage


在信息学竞赛中的应用

1. Happy Birthday(NOI2006)
2. 郁闷的出纳员 (NOI2004)
3. 营业额统计 (HNOI2002)
4. 宠物收养所 (HNOI2004)

.....

探索历程

感性与理性中螺旋前进




```
If s[left[t]]>s[right[t]]
then  right_rotate(t)
      简 单
```

效果不好，不一定
能降低平均深度

```
If s[left[t]]>s[right[t]]
then right_rotate(t)
```

简单



```
If
    s[left[left[t]]]>s[right[t]]
```

```
then right_rotate (t)
```

能降低平均深度。可能这是解决有序和随机数据的最好方法

效果不好，不一定
能降低平均深度

```
If s[left[t]] > s[right[t]]  
then right_rotate(t)
```

简单

效果不好，不一定
能降低平均深度

```
If  
s[left[left[t]]] > s[right  
t[t]]
```

then right_rotate (t)

能降低平均深度。可能这是解决有序和随机数据的最好方法

依赖数据。
对于人
字形数据，BST
会退化成 $O(N)$ 深

If

$s[\text{left}[\text{left}[t]]] > s[\text{right}[t]]$

then right_rotate (t)

能降低平均深度。这应该是解决有序和随机数据的很好方法

If

$s[\text{right}[\text{left}[t]]] > s[\text{right}[t]]$

then left_rotate (left[t])
right_rotate (t)

效果不好，不一定

能降低平均深度

依赖数据。对于
人

字形数据，BST

会退化成 $O(N)$ 深

If

```
s[left[left[t]]]>s[right[t]]
```

then right_rotate (t)

能降低平均深度。可能这是解决有序和随机数据的最好方法

If

```
s[right[left[t]]]>s[right[t]]
```

then left_rotate (left[t])
right_rotate (t)

依赖数据。对于
人
字形数据，BST
会退化成长度
为 $O(N)$ 的
深度，复杂度
不容易分析

添加

If

```
s[right[left[t]]]>s[right[t]]
```

then left_rotate(left[t])
right_rotate(t)

依赖数据 . 对于
人
字形数据 , B S
T
会退化成 $O(N)$ 深度

Maintain

探索历程

- 在此感谢
- 我的英语老师 Fiona
- 复旦大学的姚子渊
- 香港大学的麦原和 Professor Golin

附

- 想获得更详尽的内容，请参考我的论文。

Insertion

■ Simple-Insert (t, v)

1. If $t=0$ then

2. $t \leftarrow \text{NEW-NODE}(v)$

3. Else

4. $s[t] \leftarrow s[t] + 1$

5. If $v < \text{key}[t]$ then

6. Simple-Insert($\text{left}[t], v$)

7. Else

8. Simple-Insert($\text{right}[t], v$)

9. Maintain ($t, v \geq \text{key}[t]$)

Insert (t, v)

Deletion

- 为了方便，我增加了 Delete 的功能：
- 如果没有要删除的点就删除最后搜索到的结点。

Deletion

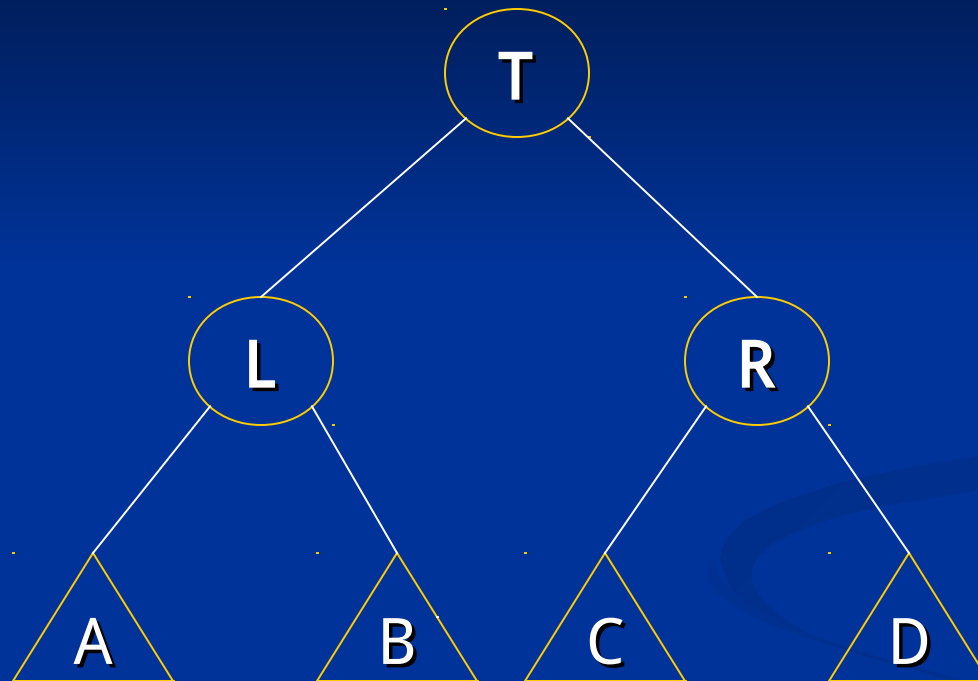
```
1. If  $s[t] \leq 2$  then
2.   record  $\leftarrow$  key[t]
3.    $t \leftarrow \text{left}[t] + \text{right}[t]$ 
4.   exit
5.  $s[t] \leftarrow s[t] - 1$ 
6. If  $v = \text{key}[t]$  then
7.   Delete( $\text{left}[t], v[t] + 1$ )
8.   Key[t]  $\leftarrow$  record
9.   Maintain( $t, \text{true}$ )
10. Else if  $v < \text{key}[t]$  then
11.   Delete( $\text{left}[t], v$ )
12. Else
13.   Delete( $\text{right}[t], v$ )
14.   Maintain( $t, v < \text{key}[t]$ )
```

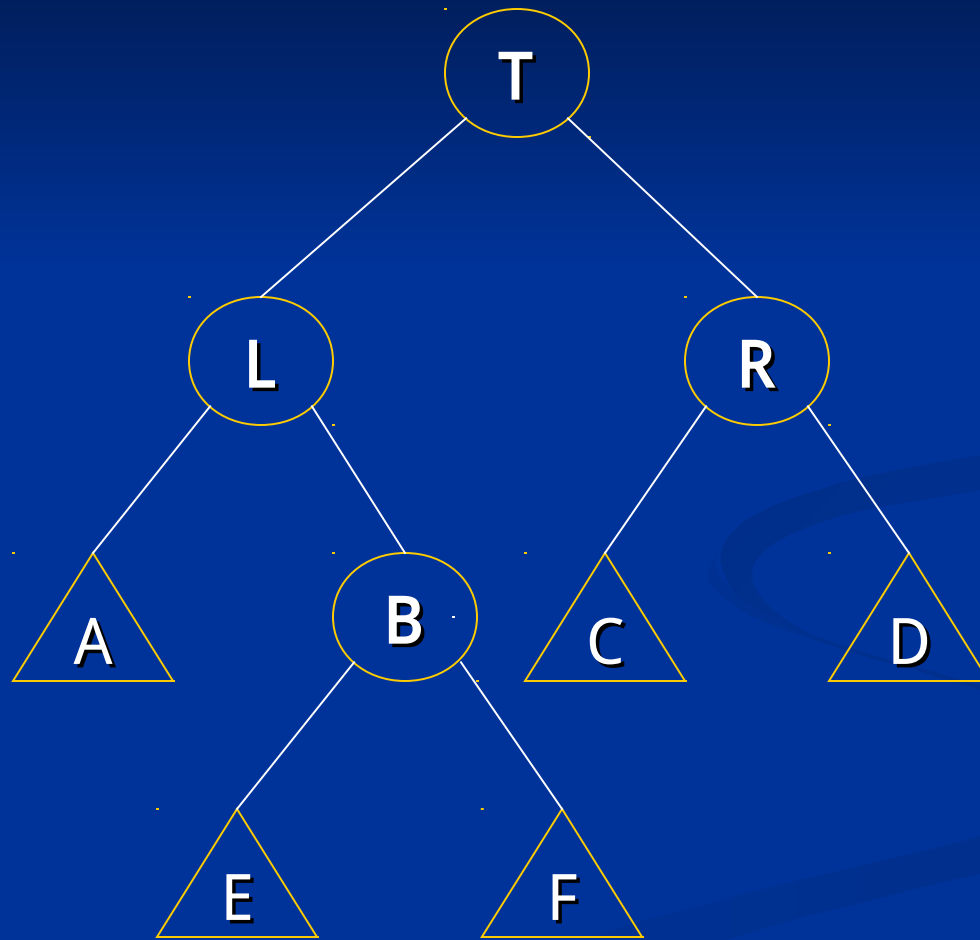
Delete(t, v)

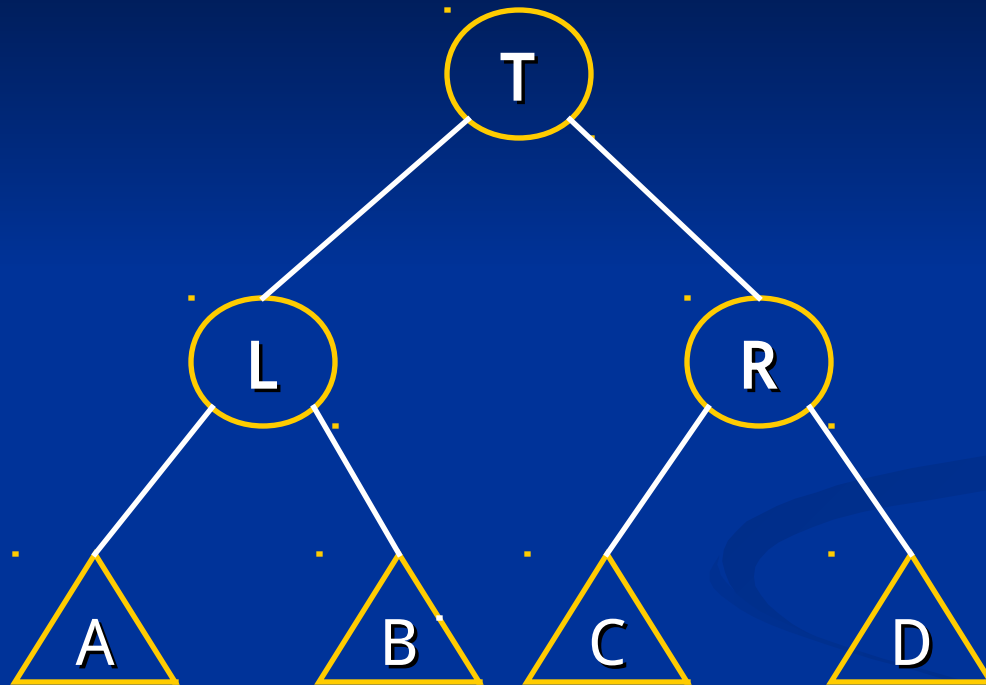
Deletion

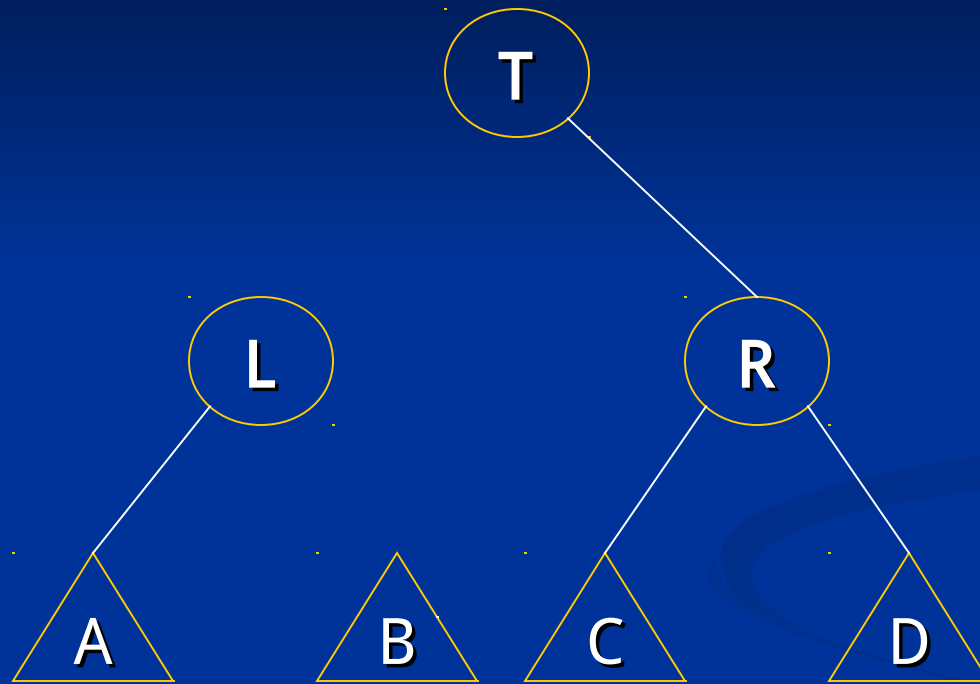
- 其实结合 SBT-Insert 地使用 Simple-Delete 时间复杂度仅仅是 $O(\log n^*)$ 。其中 n^* 是总的插入次数。
- Simple-Delete 不仅更简单而且常数非常小。

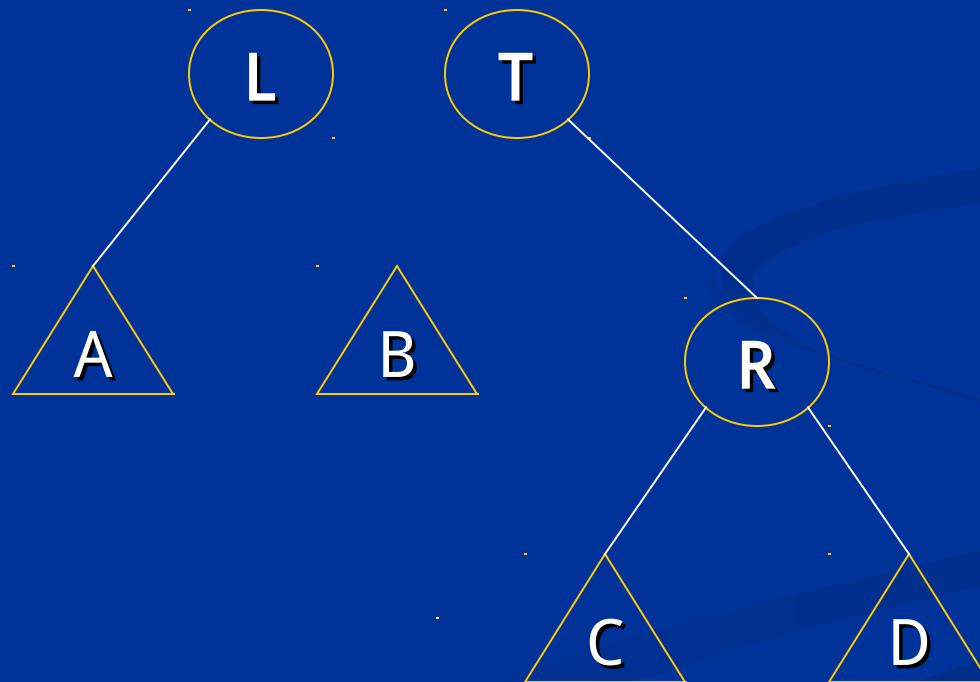
f

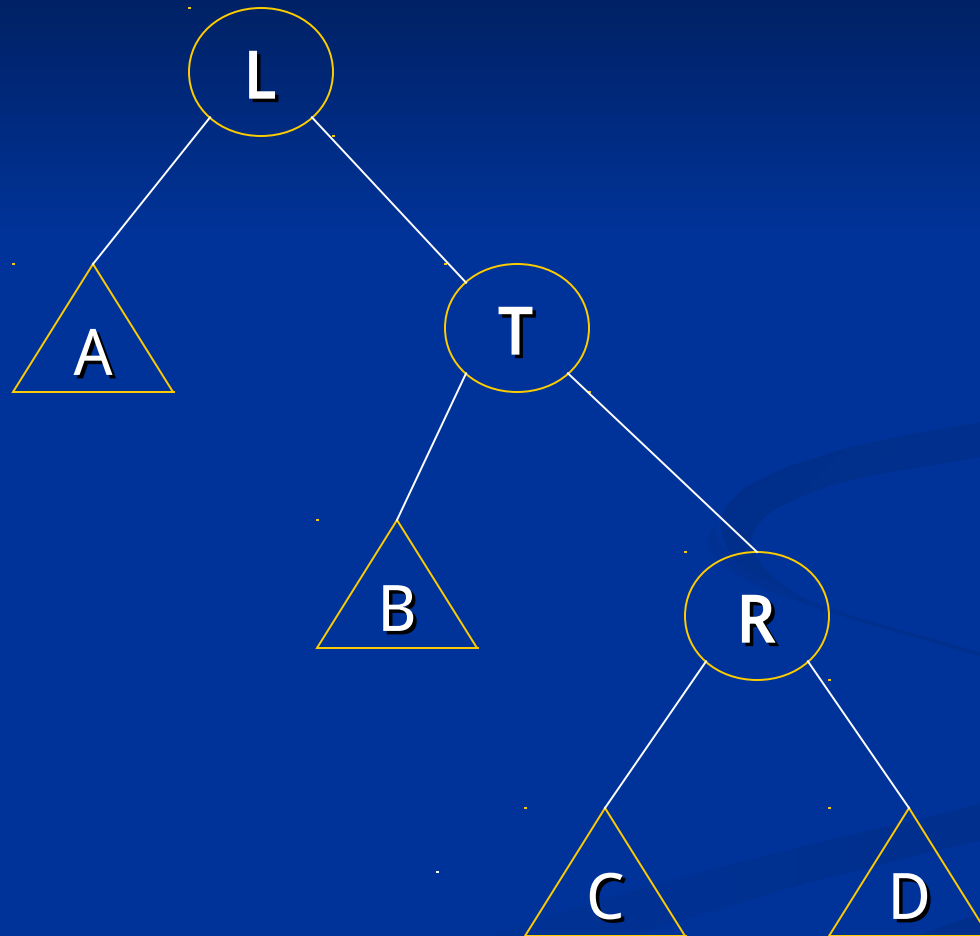


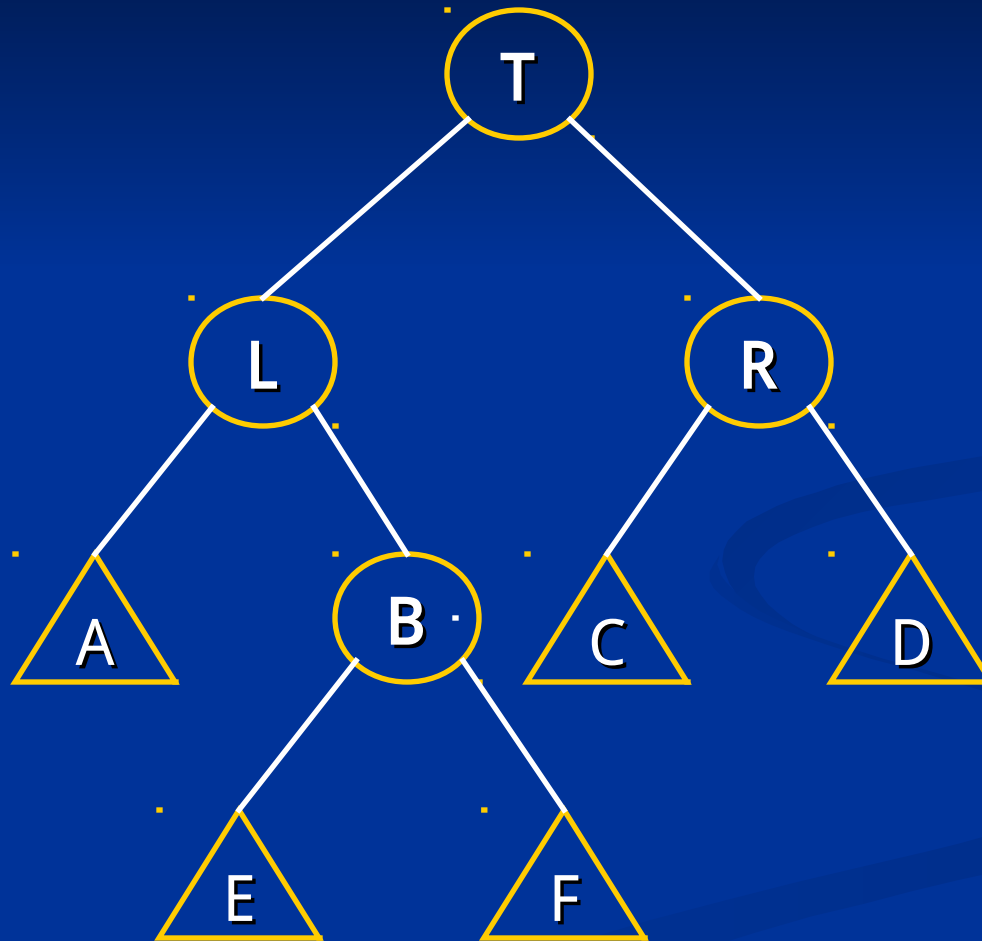


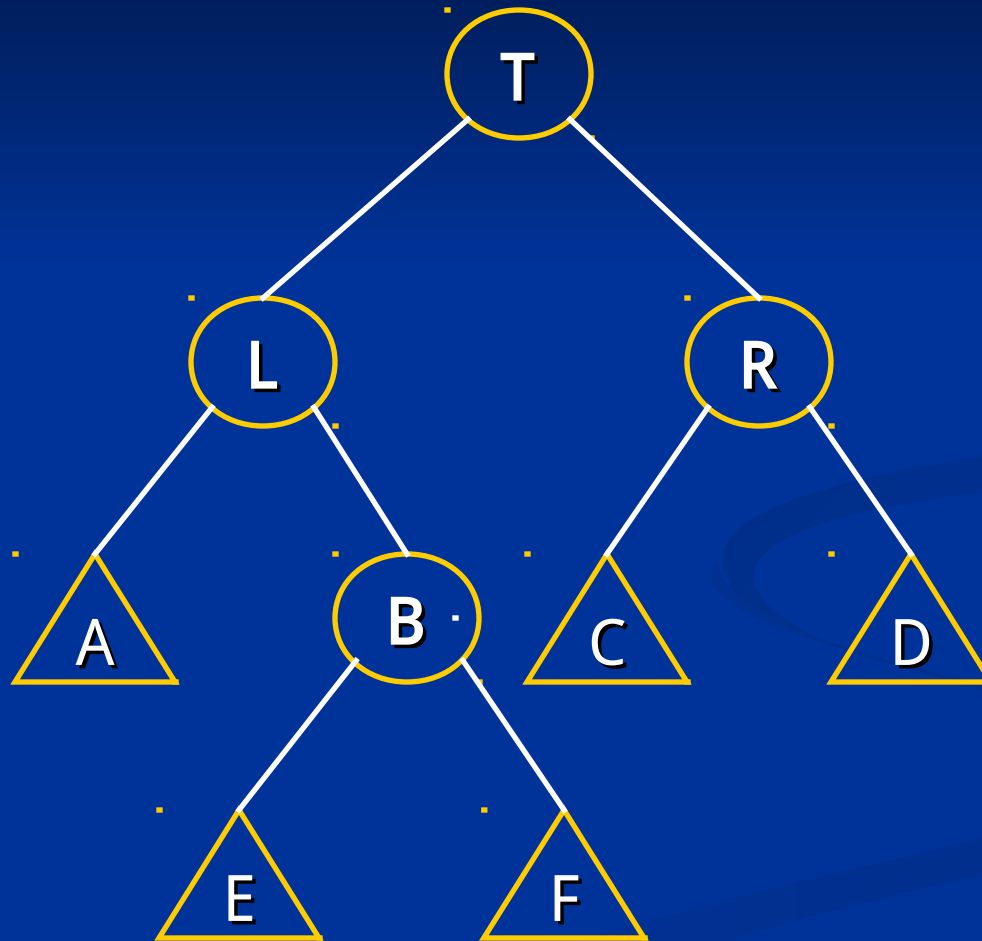


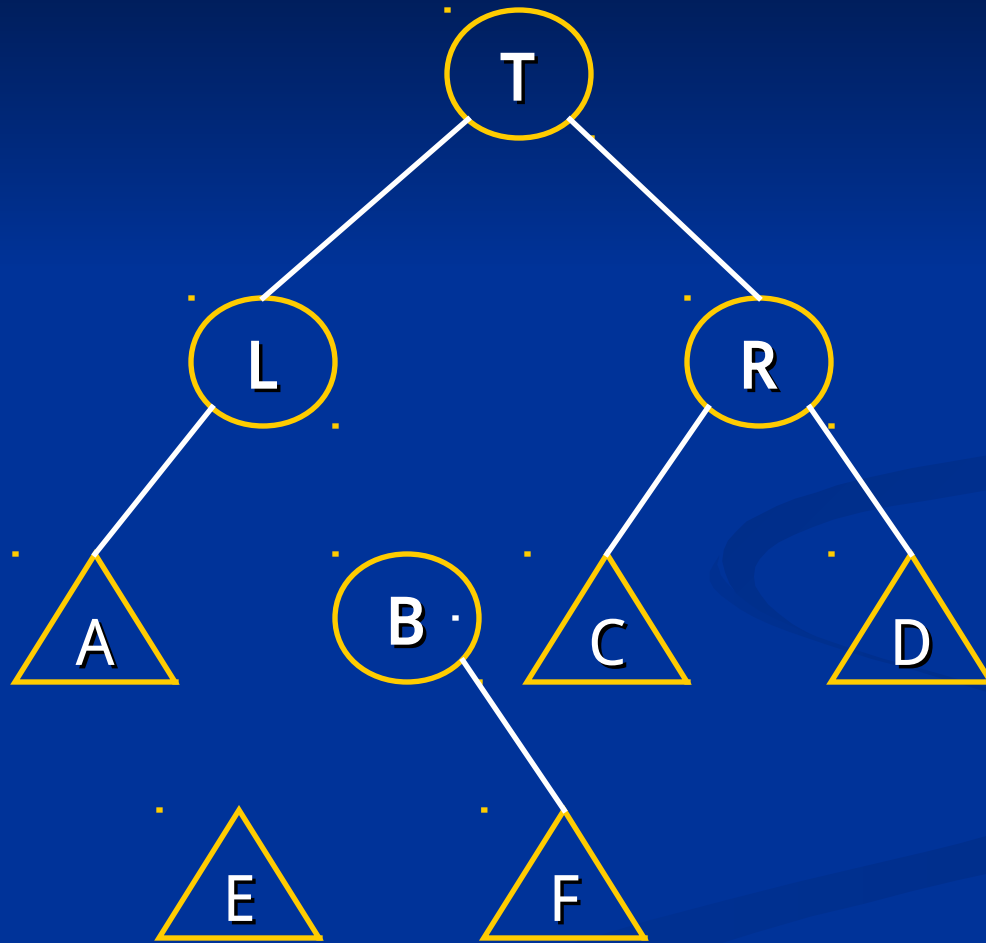


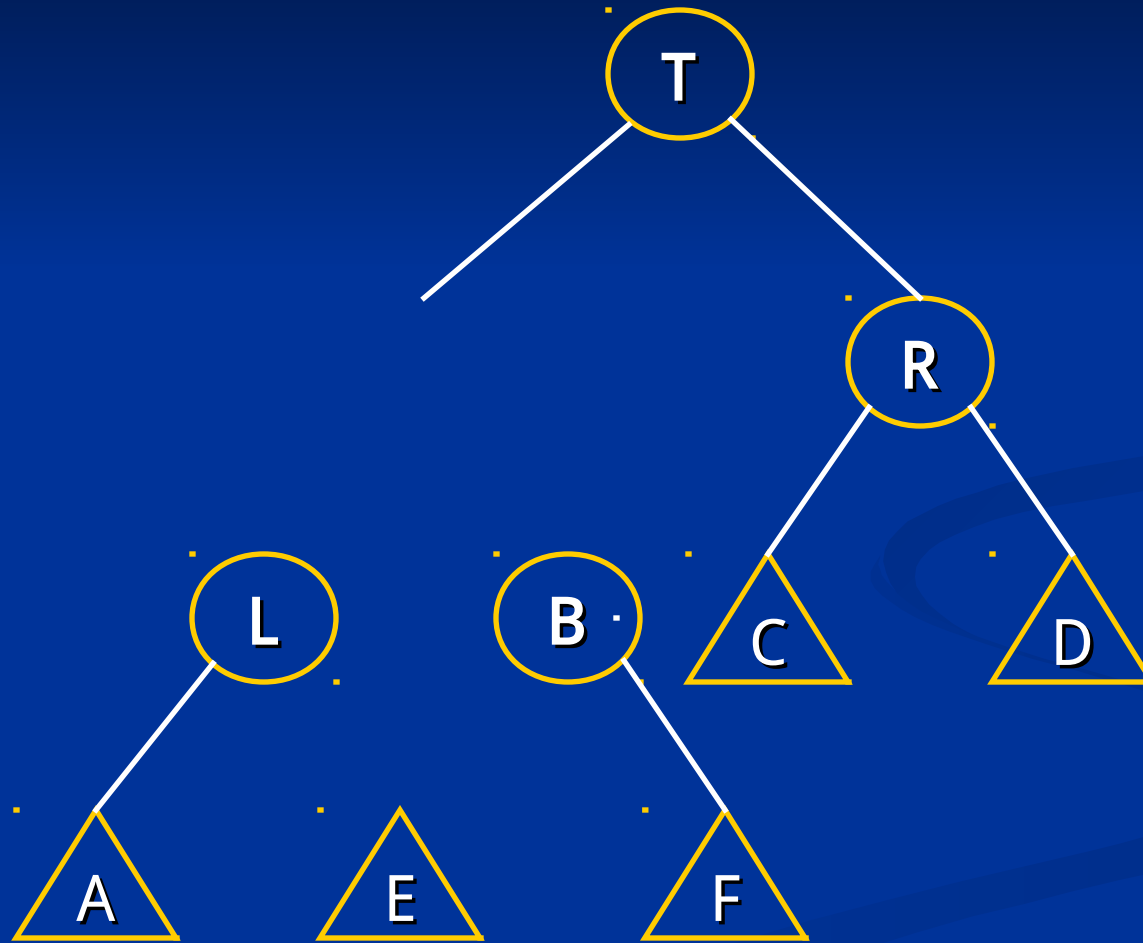


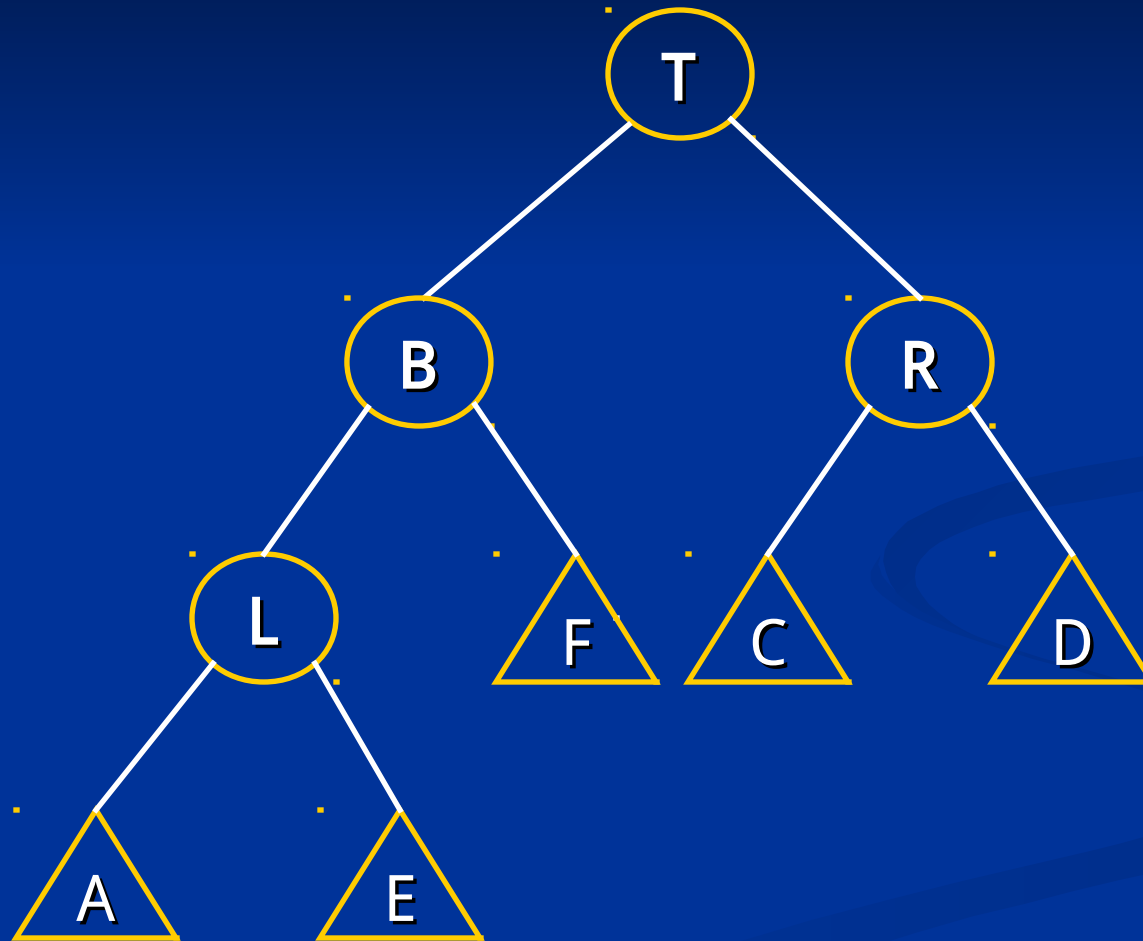


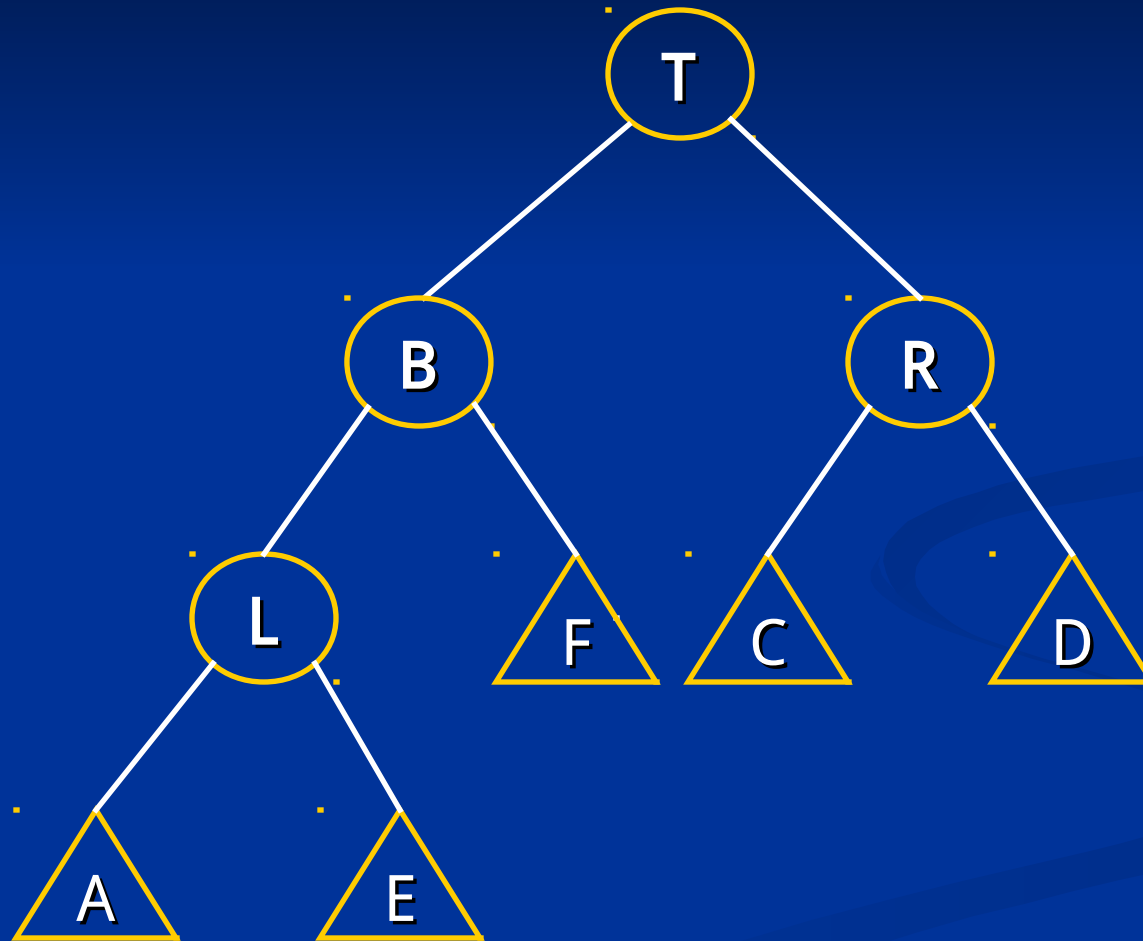


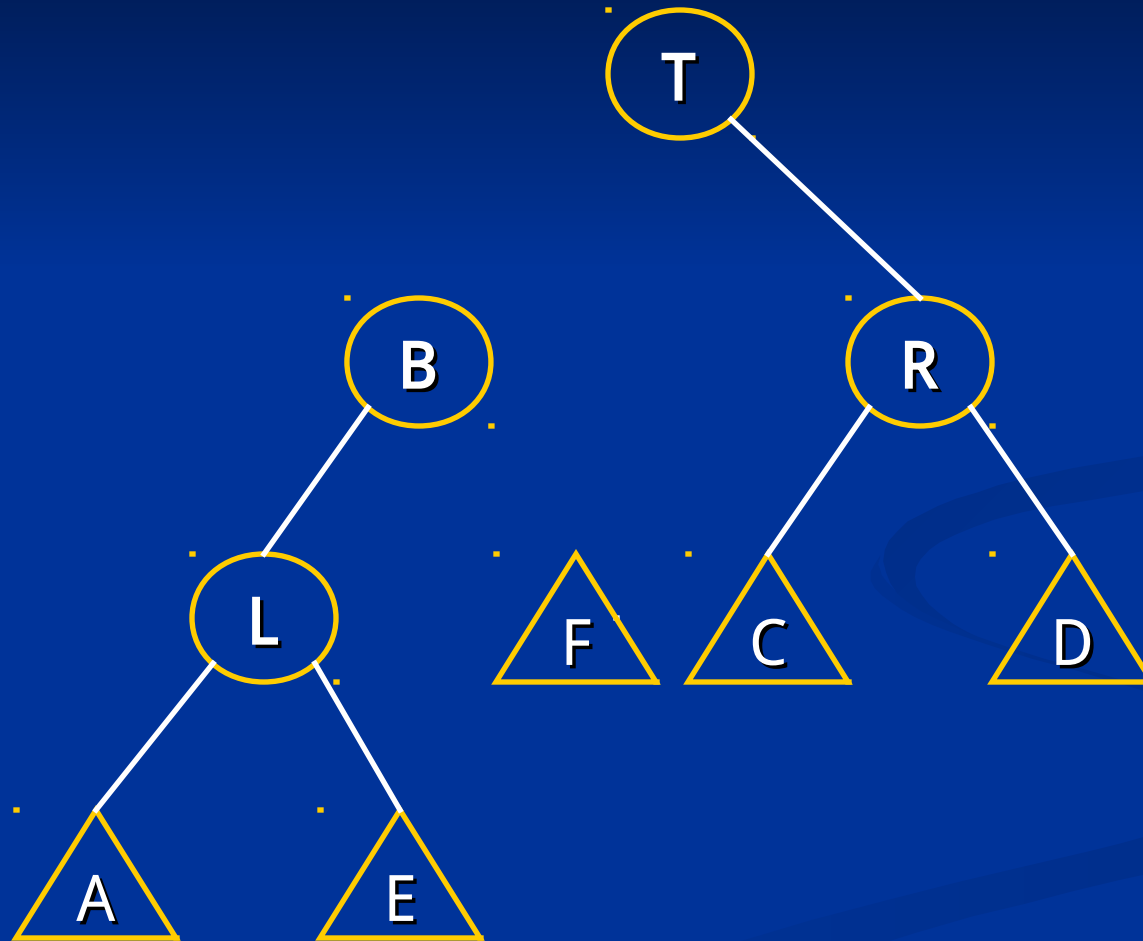


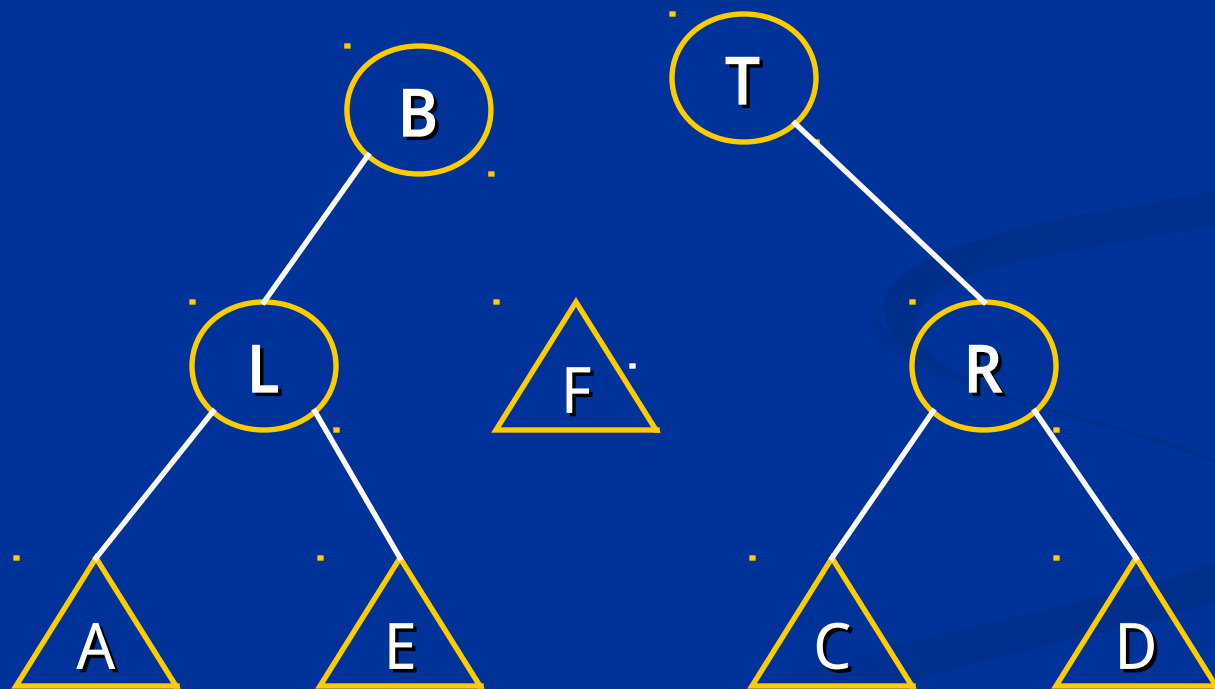


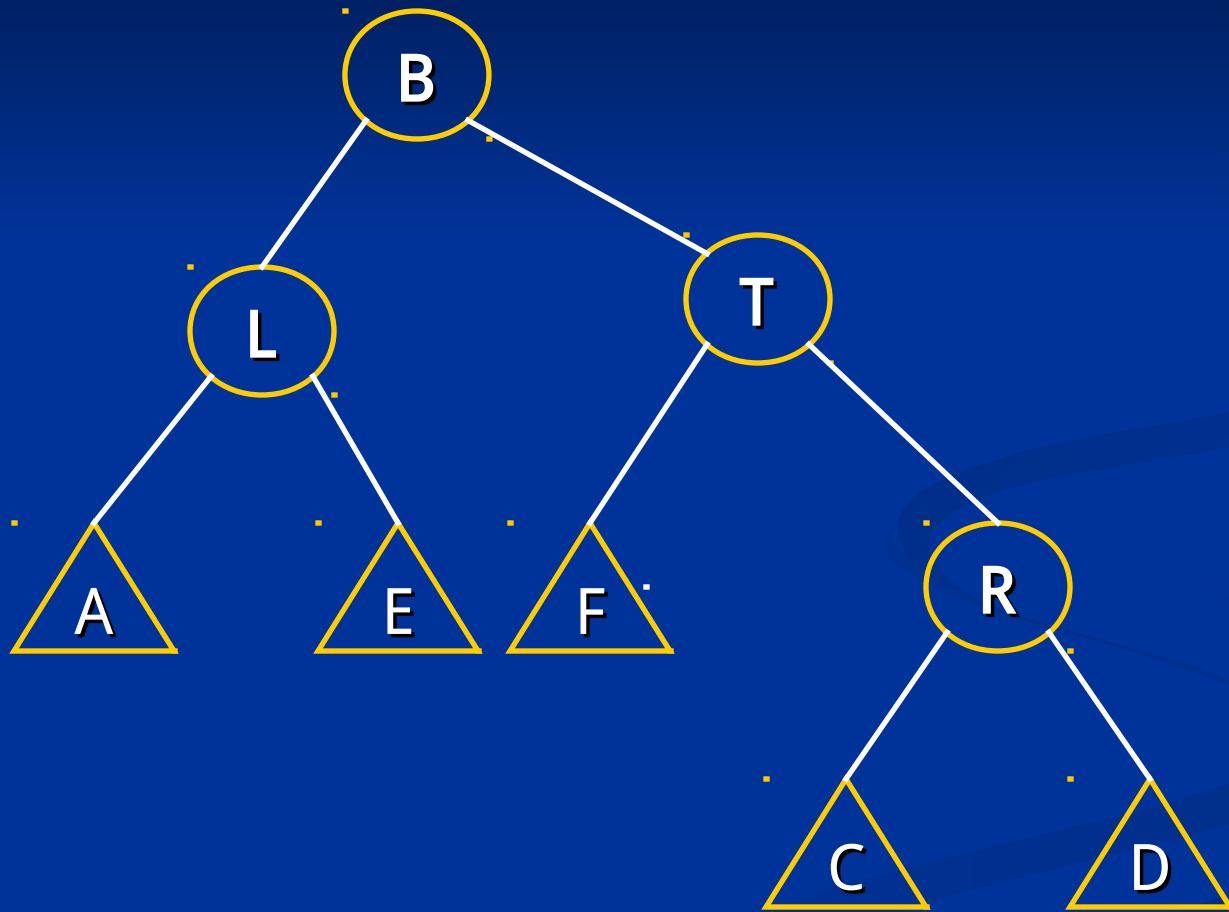


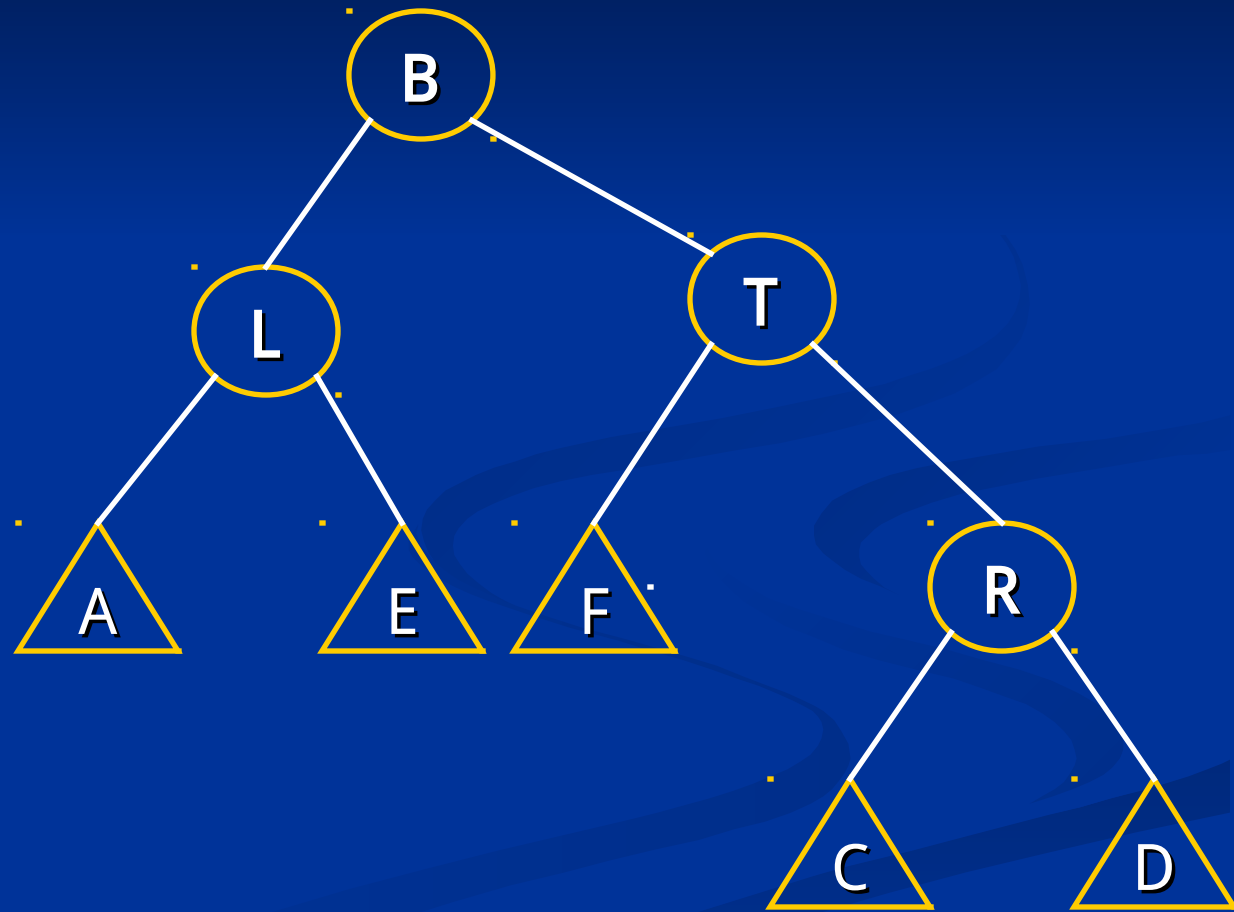


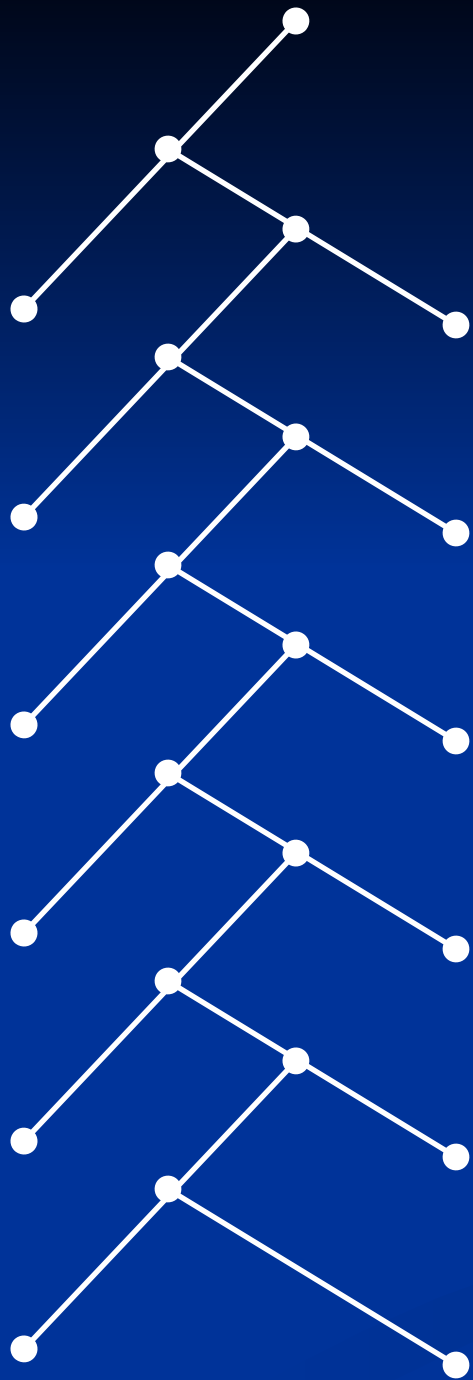












插入人字形数据后退化的

SBT V2.0

