5.3 Finding the Second Largest Key

- 方法：**Find Max**，删除**Max**,在剩余数据中找最大元

  2n-3次比较

第一趟找Max，没有获取关于“次大元”的有用信息

Useful Information: the key which lost to a key other than Max cannot be the second largest key

找次大元首先必须确定最大元

5个元素，找最大元:

- 比较　　　胜利者
- ---------------------------
- x_1, x_2　　　x_1
- x_1, x_3　　　x_1
- x_1, x_4　　　x_4
- x_4, x_5　　　x_4
- $max = x_4$并且次大元是x_5或者x_1，因为x_2和x_3都败给了非最大元的x_1.因此在本例中仅仅需要再多比较一次就可以找到次大元。

任何一个败给最大元以外其它数据元素的都不可能成为次大元
second-largest

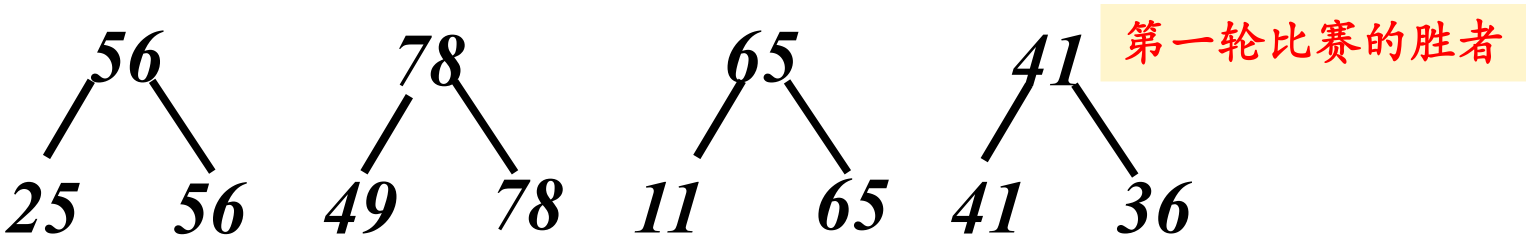
5.3 Finding the Second Largest Key

- 锦标赛法：
  - 多“轮”
  - 每一轮结束后，胜者进入下一轮
  - 每一轮中数据元素成对比较
  - 若某一轮中数据元素的个数为奇数，则其中一个数据元素直接进入下一轮

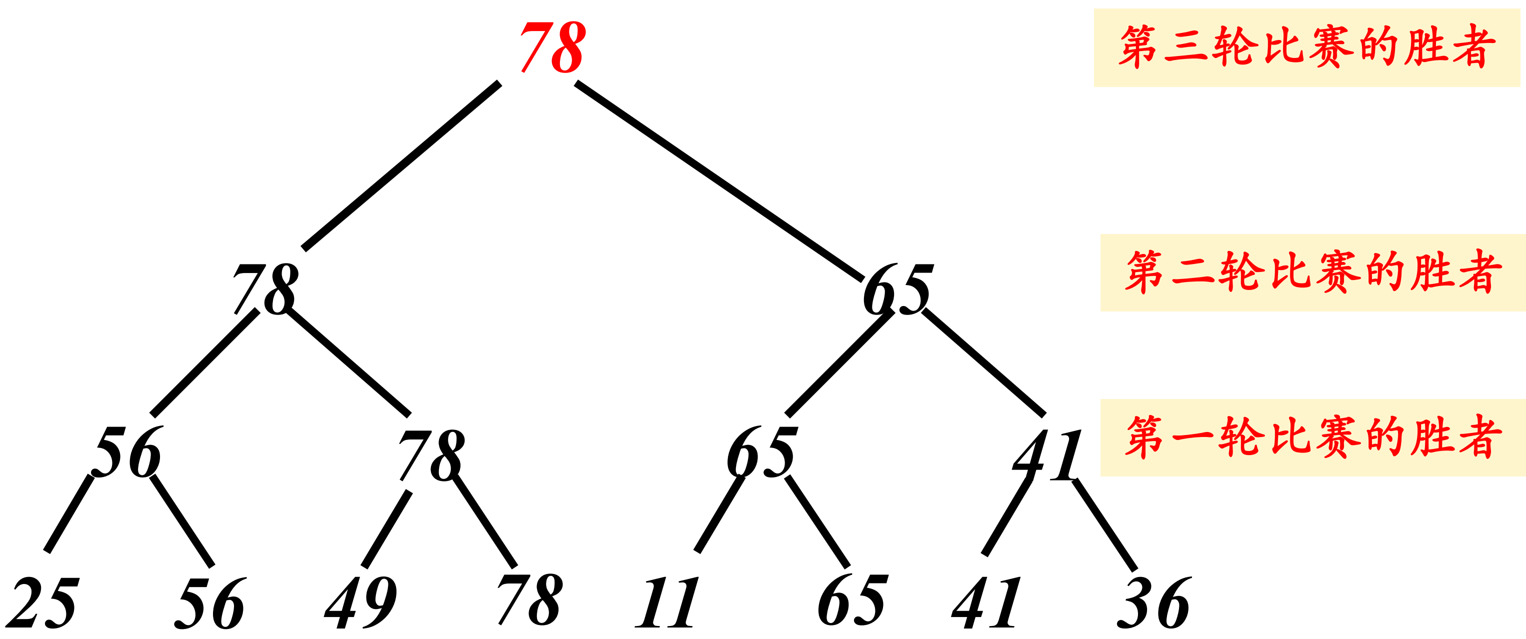
锦标赛法----二叉树

25　56　49　78　11　65　41　36

锦标赛法----二叉树



锦标赛法----二叉树


78
78
65
56
78
65
41
25
56
49
78
11
65
41
36
第三轮比赛的胜者
第二轮比赛的胜者
第一轮比赛的胜者

锦标赛法----二叉树

找最大元和次大元最坏需要比较n+⌈log n⌉-2次

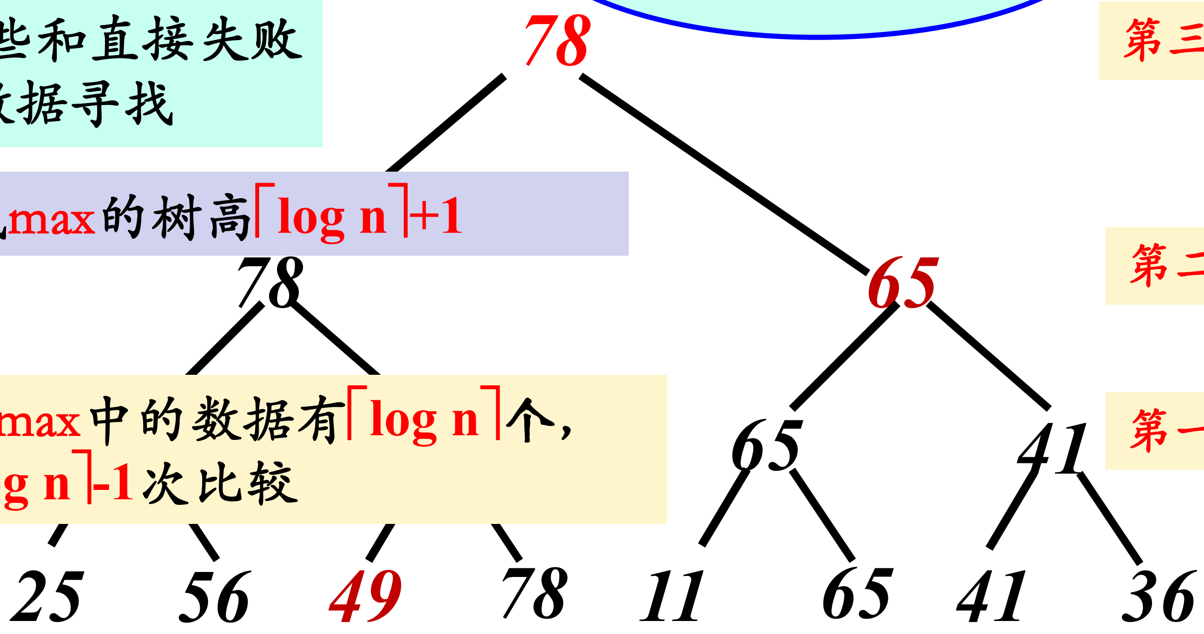
找次大元之前先找最大元：n-1次比较

次大元在那些和直接失败于max中的数据寻找

“锦标赛”找max的树高⌈log n⌉+1

直接失败于max中的数据有⌈log n⌉个，找次大元⌈log n⌉-1次比较

次大元:65,56,49
三个人中选



第三轮比赛的胜者

第二轮比赛的胜者

第一轮比赛的胜者

5.3 Finding the Second Largest Key

- 找次大元之前先找最大元：n-1次比较
- 次大元在那些和直接失败于max中的数据寻找
- “锦标赛”找max的树高$\lceil \log n \rceil+1$
- 直接失败于max中的数据有$\lceil \log n \rceil$个，找次大元$\lceil \log n \rceil-1$次比较

找次大元算法实现

- 存储方式：数组

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
25	56	49	78	11	65	41	36							

将输入数据移到叶子结点位置

```
for(i=n, j=2*n-1; i>=1; i--, j--) E[j]=E[i];
```

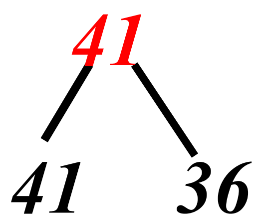
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
							25	56	49	78	11	65	41	36

找次大元算法实现

- 存储方式：数组

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
							25	56	49	78	11	65	41	36

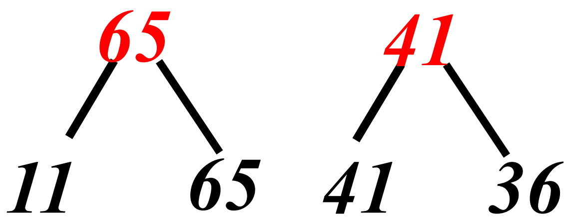
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
						41	25	56	49	78	11	65	41	36



找次大元算法实现

- 存储方式：数组

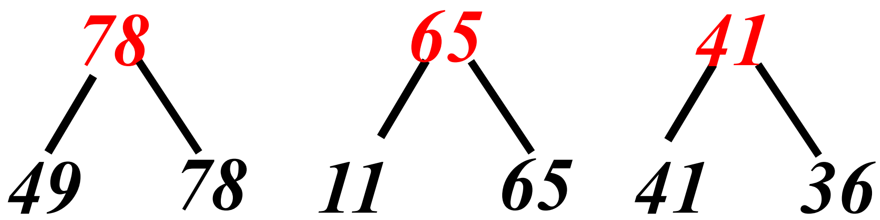
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
					65	41	25	56	49	78	11	65	41	36



找次大元算法实现

- 存储方式：数组

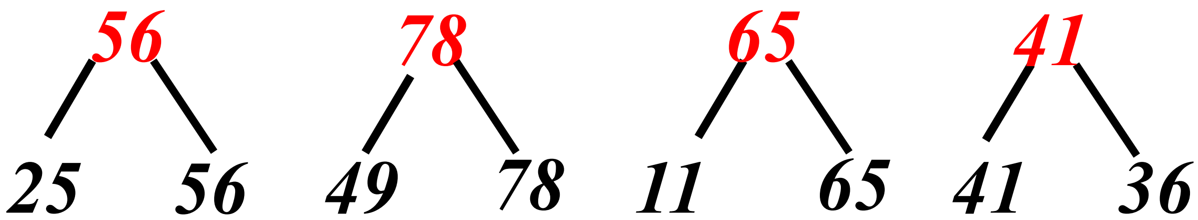
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
				78	65	41	25	56	49	78	11	65	41	36



找次大元算法实现

- 存储方式：数组

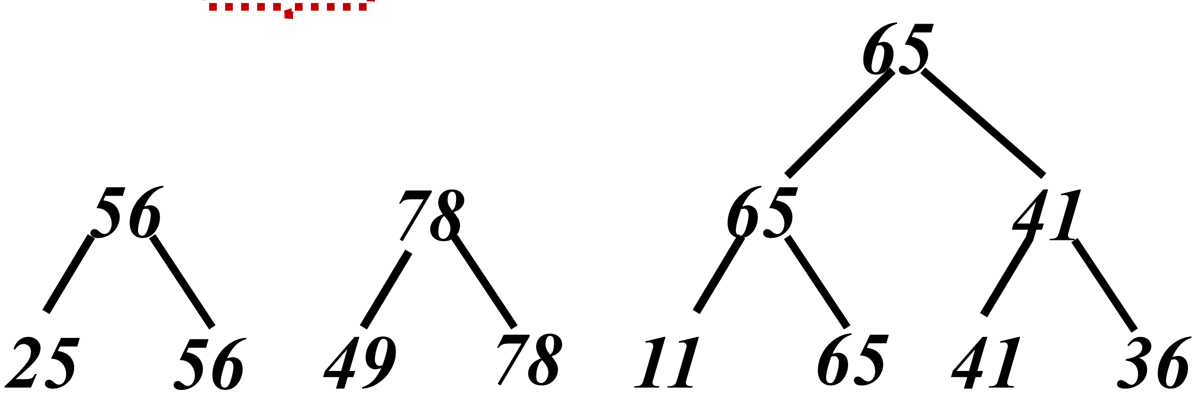
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
			56	78	65	41	25	56	49	78	11	65	41	36



找次大元算法实现

- 存储方式：数组

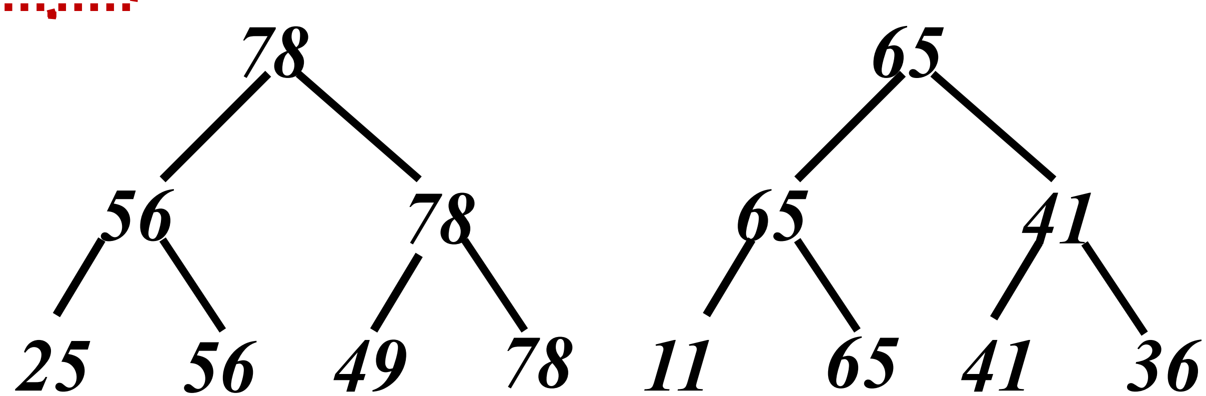
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
		65	56	78	65	41	25	56	49	78	11	65	41	36



找次大元算法实现

- 存储方式：数组

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	78	65	56	78	65	41	25	56	49	78	11	65	41	36

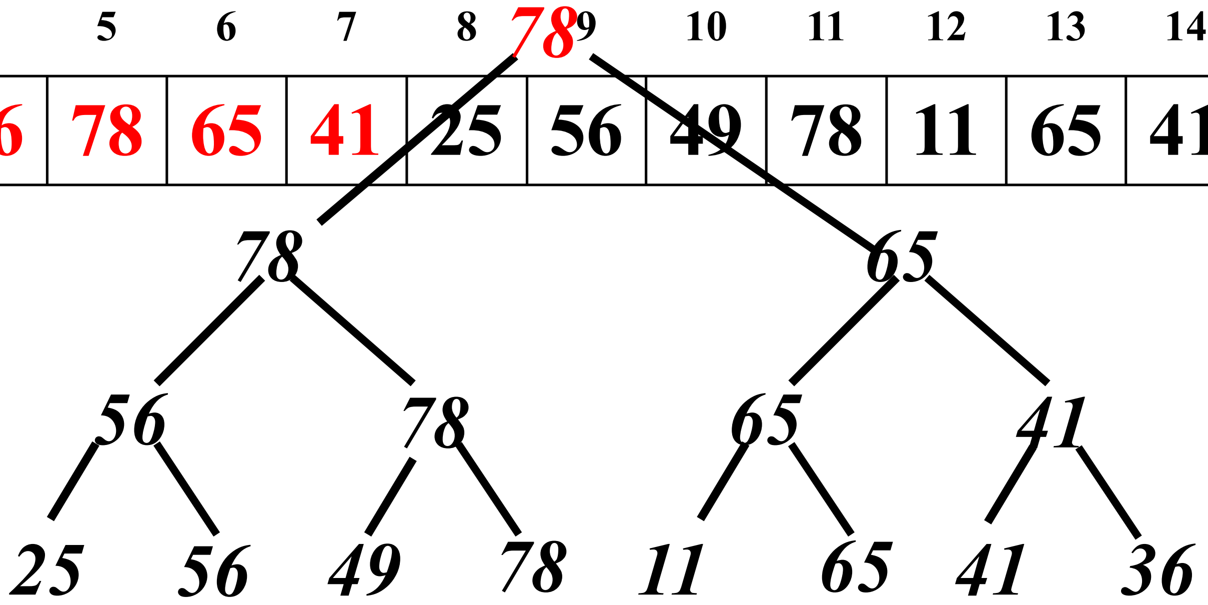


找次大元算法实现

- 存储方式：数组

```
for(last=2*n-2; last>=2; last-=2)
    if(E[last]>E[last+1])
  E[last/2]=E[last];
    else E[last/2]=E[last+1];
```

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
78	78	65	56	78	65	41	25	56	49	78	11	65	41	36



```
Element secondlarge(Element E[ ],int n)
{  for(i=n, j=2*n-1; i>=1; i--, j--) E[j]=E[i];
   for(last=2*n-2;last>=2;last-=2)
      if(E[last]>E[last+1]) E[last/2]=E[last];
      else E[last/2]=E[last+1];
    max = E[1];  secondLargest =- ∞; i = 1;

      补齐找次大元的伪码描述

   return secondLargest;
}
```

5.3 Finding the Second Largest Key

Theorem 5.2 Any algorithm (that works by comparing keys) to find the second largest in a set of n keys must do at least $n + \lceil \lg n \rceil - 2$ comparisons in the worst case.

- 定理5.2：n个数据元素中找次大元最坏情况下至少需要比较$n+\lceil \log n \rceil-2$次
- 证明：假设n个数据元素中不含重复数据。

  首先 必须找*max*，以确定*second-largest*不是最大的

  ----需要n-1次比较确定*max*

5.3 Finding the Second Largest Key

- 给每个数据元素赋权重$w(x)$,初始$w(x)=1$。
- 一次比较后，败者将权重全部转赠给赢者，其权重变为0。

对策策略

Case	Adversary reply	Updating of weights
$w(x) > w(y)$	$x > y$	New $w(x)$ = prior $(w(x) + w(y))$; new $w(y) = 0$.
$w(x) = w(y) > 0$	Same as above.	Same as above.
$w(y) > w(x)$	$y > x$	New $w(y)$ = prior $(w(x) + w(y))$; new $w(x) = 0$.
$w(x) = w(y) = 0$	Consistent with previous replies.	No change.

例子

Comparands	Weights	Winner	New Weights	Keys
x1,x2	w(x1)=w(x2)	x1	2,0,1,1,1	20,10,*,*,*
x1,x3	w(x1)>w(x3)	x1	3,0,0,1,1	20,10,15,*,*
x5,x4	w(x5)=w(x4)	x5	3,0,0,0,2	20,10,15,30,40
x1,x5	w(x1)>w(x5)	x1	5,0,0,0,0	41,10,15,30,40

根据多策论构造的该算法最坏情况输入：41，10，15，30，40

max：41； Second Largest Key 的候选人：x2, x3, x5

Case	Adversary reply	Updating of weights
$w(x) > w(y)$	$x > y$	New $w(x)$ = prior $(w(x) + w(y))$; new $w(y) = 0$.
$w(x) = w(y) > 0$	Same as above.	Same as above.
$w(y) > w(x)$	$y > x$	New $w(y)$ = prior $(w(x) + w(y))$; new $w(x) = 0$.
$w(x) = w(y) = 0$	Consistent with previous replies.	No change.

5.3 Finding the S

- 给每个数据元素赋权重$w(x)$,初始$w(x)$=1。
- 一次比较后，败者将权重全部转赠给赢者，其权重变为0。

1. A key has lost a comparison if and only if its weight is zero.
2. In the first three cases, the key chosen as the winner has nonzero weight, so it has not yet lost. The adversary can give it an arbitrarily high value to make sure it wins without contradicting any of its earlier replies.
3. The sum of the weights is always n. This is true initially, and the sum is preserved by the updating of the weights.
4. When the algorithm stops, only one key can have nonzero weight. Otherwise there would be at least two keys that never lost a comparison, and the adversary could choose values to make the algorithm's choice of secondLargest incorrect.

5.3 F

Case	Adversary reply	Updating of weights
$w(x) > w(y)$	$x > y$	New $w(x) =$ prior $(w(x) + w(y))$; new $w(y) = 0$.
$w(x) = w(y) > 0$	Same as above.	Same as above.
$w(y) > w(x)$	$y > x$	New $w(y) =$ prior $(w(x) + w(y))$; new $w(x) = 0$.
$w(x) = w(y) = 0$	Consistent with previous replies.	No change.

- 给每个
  - 一次比较后，败者将权重全部转赠给赢者，其权重变为0。
  - 任意时刻所有数据元素权的和为n,
  - 当算法停止时，只有max=x的权为n，其余均为0。
- 若x,y比较，x为胜者：

$$w(x) = w(x) + w(y), \begin{cases} w^{new}(x) = 2w(x), w(x) = w(y) \\ \cancel{w^{new}(x) > 2w(x), w(x) < w(y)} \\ w^{new}(x) < 2w(x), w(x) > w(y) \end{cases}$$

令$w_i(x)$为参加第i次比较后x的权，根据对策论$w_i(x) \le 2w_{i-1}(x)$,最大元x共参加k次比较，则：

$$n = w_k(x) \le 2w_{k-1}(x) \le 2^2 w_{k-2}(x) \le 2^3 w_{k-3}(x) \le \ldots \le 2^k w_0(x)$$

$$k \ge \lceil \log n \rceil.$$

max最坏情况下至少参加了$\lceil \log n \rceil$次比较，则再进行$\lceil \log n \rceil$-1次比较可找到second-largest.所以n个数据元素中找次大元最坏情况下至少需要比较$n+\lceil \log n \rceil-2$次。