

本节内容

顺序查找

知识总览

顺序查找

算法思想

算法实现

算法优化

顺序查找的算法思想

顺序查找，又叫“线性查找”，通常用于线性表。

算法思想：从头到 jio 挨个找（或者反过来也OK）

33	10	13	29	16	19	32	7	43	41	37
----	----	----	----	----	----	----	---	----	----	----



查找目标：

43

顺序查找的算法思想



顺序查找，又叫“线性查找”，通常用于线性表。

算法思想：从头到 jio 挨个找（或者反过来也OK）



查找目标：

43

顺序查找的实现

```
typedef struct{           //查找表的数据结构（顺序表）
    ElemType *elem;       //动态数组基址
    int TableLen;         //表的长度
}SSTable;

//顺序查找
int Search_Seq(SSTable ST,ElemType key){
    int i;
    for(i=0;i<ST.TableLen && ST.elem[i]!=key; ++i);
    //查找成功，则返回元素下标；查找失败，则返回-1
    return i==ST.TableLen? -1 : i;
}
```

查找目标:

43

33	10	13	29	16	19	32	7	43	41	37				
----	----	----	----	----	----	----	---	----	----	----	--	--	--	--

0

1

2

3

4

5

6

7

8

9

10

11

12

13

...

TableLen=11

顺序查找的实现

```
typedef struct{           //查找表的数据结构（顺序表）
    ElemType *elem;       //动态数组基址
    int TableLen;         //表的长度
}SSTable;

//顺序查找
int Search_Seq(SSTable ST,ElemType key){
    int i;
    for(i=0;i<ST.TableLen && ST.elem[i]!=key; ++i);
    //查找成功，则返回元素下标；查找失败，则返回-1
    return i==ST.TableLen? -1 : i;
}
```

查找目标:

43

33	10	13	29	16	19	32	7	43	41	37				
----	----	----	----	----	----	----	---	----	----	----	--	--	--	--

0 1 2 3 4 5 6 7 8 9 10 11 12 13 ...

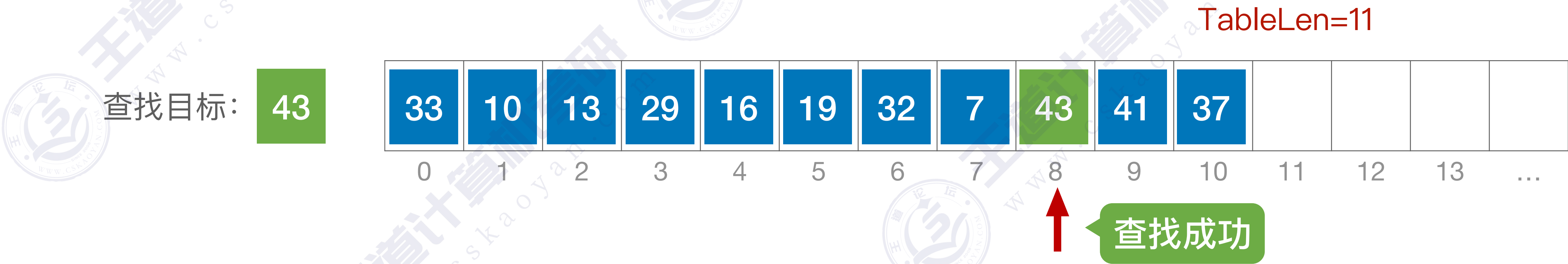


TableLen=11

顺序查找的实现

```
typedef struct{           //查找表的数据结构（顺序表）
    ElemType *elem;       //动态数组基址
    int TableLen;         //表的长度
}SSTable;

//顺序查找
int Search_Seq(SSTable ST,ElemType key){
    int i;
    for(i=0;i<ST.TableLen && ST.elem[i]!=key; ++i);
    //查找成功，则返回元素下标；查找失败，则返回-1
    return i==ST.TableLen? -1 : i;
}
```



顺序查找的实现

```
typedef struct{           //查找表的数据结构（顺序表）
    ElemType *elem;       //动态数组基址
    int TableLen;         //表的长度
}SSTable;

//顺序查找
int Search_Seq(SSTable ST,ElemType key){
    int i;
    for(i=0;i<ST.TableLen && ST.elem[i]!=key; ++i);
    //查找成功，则返回元素下标；查找失败，则返回-1
    return i==ST.TableLen? -1 : i;
}
```

查找目标:

66

33	10	13	29	16	19	32	7	43	41	37				
----	----	----	----	----	----	----	---	----	----	----	--	--	--	--

0 1 2 3 4 5 6 7 8 9 10 11 12 13 ...



TableLen=11

顺序查找的实现

```
typedef struct{           //查找表的数据结构（顺序表）
    ElemType *elem;       //动态数组基址
    int TableLen;         //表的长度
}SSTable;

//顺序查找
int Search_Seq(SSTable ST,ElemType key){
    int i;
    for(i=0;i<ST.TableLen && ST.elem[i]!=key; ++i);
    //查找成功，则返回元素下标；查找失败，则返回-1
    return i==ST.TableLen? -1 : i;
}
```

查找目标:

66

33	10	13	29	16	19	32	7	43	41	37				
0	1	2	3	4	5	6	7	8	9	10	11	12	13	...

TableLen=11

↑ 查找失败

顺序查找的实现（哨兵）

```
typedef struct{           //查找表的数据结构（顺序表）
    ElemType *elem;       //动态数组基址
    int TableLen;         //表的长度
}SSTable;
```

```
//顺序查找
int Search_Seq(SSTable ST,ElemType key){
    ST.elem[0]=key;       //“哨兵”
    int i;
    for(i=ST.TableLen;ST.elem[i]!=key;--i); //从后往前找
    return i;             //查找成功，则返回元素下标；查找失败，则返回0
}
```

0号位置存“哨兵”



数据从下标1开始存

16	33	10	13	29	16	19	32	7	43	41	37			
0	1	2	3	4	5	6	7	8	9	10	11	12	13	...

TableLen=11



顺序查找的实现（哨兵）

```
typedef struct{           //查找表的数据结构（顺序表）
    ElemType *elem;       //动态数组基址
    int TableLen;         //表的长度
}SSTable;

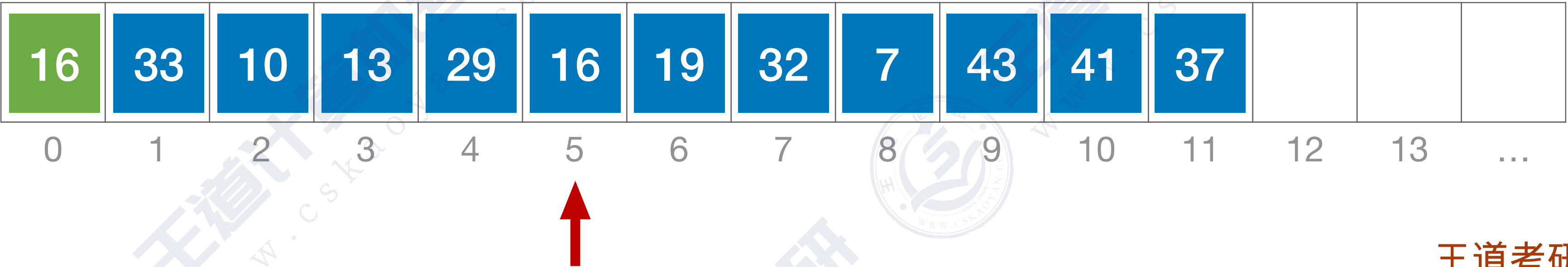
//顺序查找
int Search_Seq(SSTable ST,ElemType key){
    ST.elem[0]=key;       //“哨兵”
    int i;
    for(i=ST.TableLen;ST.elem[i]!=key;--i); //从后往前找
    return i;             //查找成功，则返回元素下标；查找失败，则返回0
}
```

0号位置存“哨兵”



数据从下标1开始存

TableLen=11



顺序查找的实现（哨兵）

```
typedef struct{           //查找表的数据结构（顺序表）
    ElemType *elem;       //动态数组基址
    int TableLen;         //表的长度
}SSTable;

//顺序查找
int Search_Seq(SSTable ST,ElemType key){
    ST.elem[0]=key;       //“哨兵”
    int i;
    for(i=ST.TableLen;ST.elem[i]!=key;--i); //从后往前找
    return i;             //查找成功，则返回元素下标；查找失败，则返回0
}
```

0号位置存“哨兵”



数据从下标1开始存

TableLen=11

16	33	10	13	29	16	19	32	7	43	41	37			
0	1	2	3	4	5	6	7	8	9	10	11	12	13	...



查找成功

顺序查找的实现（哨兵）

```
typedef struct{           //查找表的数据结构（顺序表）
    ElemType *elem;       //动态数组基址
    int TableLen;         //表的长度
}SSTable;

//顺序查找
int Search_Seq(SSTable ST,ElemType key){
    ST.elem[0]=key;       //“哨兵”
    int i;
    for(i=ST.TableLen;ST.elem[i]!=key;--i); //从后往前找
    return i;             //查找成功，则返回元素下标；查找失败，则返回0
}
```

0号位置存“哨兵”



数据从下标1开始存

66	33	10	13	29	16	19	32	7	43	41	37			
0	1	2	3	4	5	6	7	8	9	10	11	12	13	...

TableLen=11



顺序查找的实现（哨兵）

```
typedef struct{           //查找表的数据结构（顺序表）
    ElemType *elem;       //动态数组基址
    int TableLen;         //表的长度
}SSTable;
```

```
//顺序查找
int Search_Seq(SSTable ST,ElemType key){
    ST.elem[0]=key;       //“哨兵”
    int i;
    for(i=ST.TableLen;ST.elem[i]!=key;--i); //从后往前找
    return i;             //查找成功，则返回元素下标；查找失败，则返回0
}
```

优点：无需判断是否越界，效率更高

0号位置存“哨兵”



数据从下标1开始存

TableLen=11

66	33	10	13	29	16	19	32	7	43	41	37			
0	1	2	3	4	5	6	7	8	9	10	11	12	13	...



查找失败

查找效率分析



16	33	10	13	29	16	19	32	7	43	41	37			
0	1	2	3	4	5	6	7	8	9	10	11	12	13	...

TableLen=11



查找成功

$$ASL = \sum_{i=1}^n P_i C_i$$

$$ASL_{成功} = \frac{1 + 2 + 3 + \dots + n}{n} = \frac{n + 1}{2}$$

$$ASL_{失败} = n + 1$$

TableLen=11

66	33	10	13	29	16	19	32	7	43	41	37			
0	1	2	3	4	5	6	7	8	9	10	11	12	13	...



查找失败

顺序查找的优化（对有序表）

查找表中元素有序存放（递增/递减）

查找目标：

21

7

13

19

29

37

43

顺序查找的优化（对有序表）

查找表中元素有序存放（递增/递减）

查找目标：

21

7

13

19

29

37

43

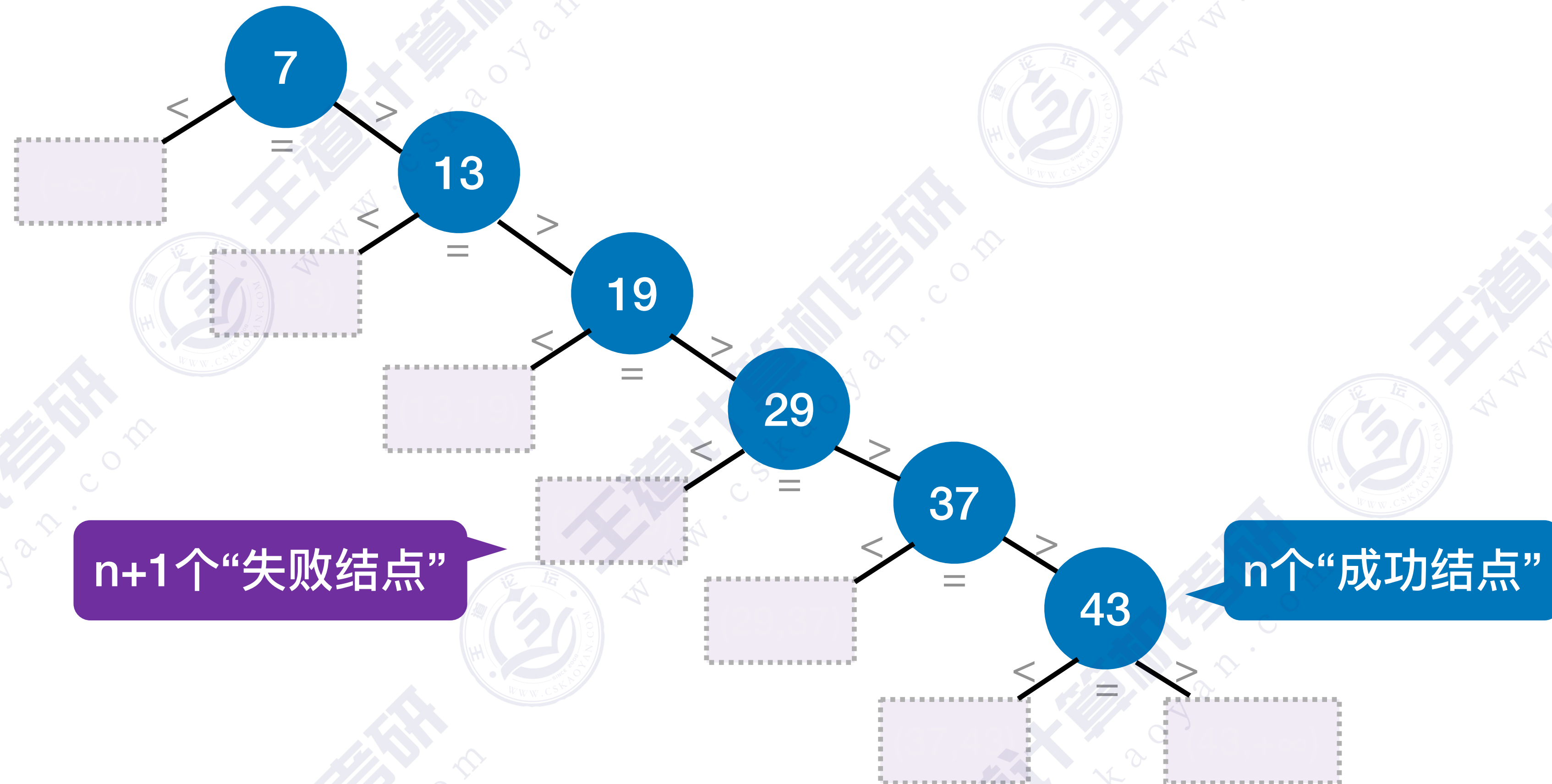
29>21查找失败

$$ASL_{\text{失败}} = \frac{1 + 2 + 3 + \dots + n + n}{n + 1} = \frac{n}{2} + \frac{n}{n + 1}$$

$$ASL_{\text{成功}} = \frac{1 + 2 + 3 + \dots + n}{n} = \frac{n + 1}{2}$$

共有n+1种查找失败的情况

用查找判定树分析ASL



一个成功结点的查找长度 = 自身所在层数
一个失败结点的查找长度 = 其父节点所在层数
默认情况下, 各种失败情况或成功情况都等概率发生

顺序查找的优化（被查概率不相等）

被查概率

7: 15%

13: 5%

19: 10%

29: 40%

37: 28%

43: 2%



$$ASL_{\text{成功}} = 1 \times 0.15 + 2 \times 0.05 + 3 \times 0.1 + 4 \times 0.4 + 5 \times 0.28 + 6 \times 0.02 = 3.67$$

被查概率大的放在靠前位置




$$ASL_{\text{成功}} = 1 \times 0.4 + 2 \times 0.28 + 3 \times 0.15 + 4 \times 0.1 + 5 \times 0.05 + 6 \times 0.02 = 2.18$$

$$ASL = \sum_{i=1}^n P_i C_i$$

知识回顾与重要考点

顺序查找

算法实现

从头到尾（或者从尾到头）挨个找

适用于顺序表、链表，表中元素有序无序都 OK

可在 0 号位置存“哨兵”，从尾部向头部挨个查找
优点：循环时无需判断下标是否越界

优化

若表中元素有序

当前关键字大于（或小于）目标关键字时，查找失败

优点：查找失败时 ASL 更少

查找判定树

成功结点的关键字对比次数 = 结点所在层数

失败结点的关键字对比次数 = 其父节点所在层数

若各个关键字被查概率不同

可按被查概率降序排列

优点：查找成功时 ASL 更少

时间复杂度

$O(n)$