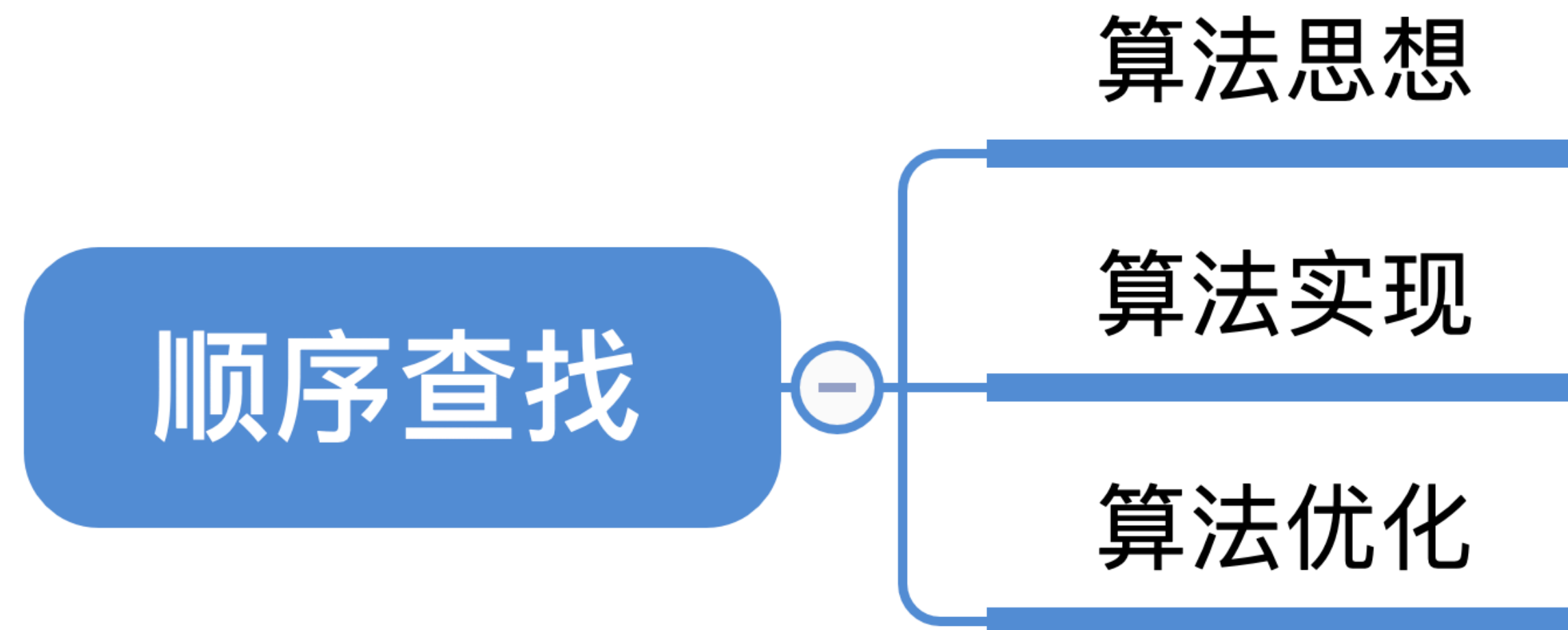


本节内容

顺序查找

知识总览



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：从头到尾挨个找（或者反过来也OK）



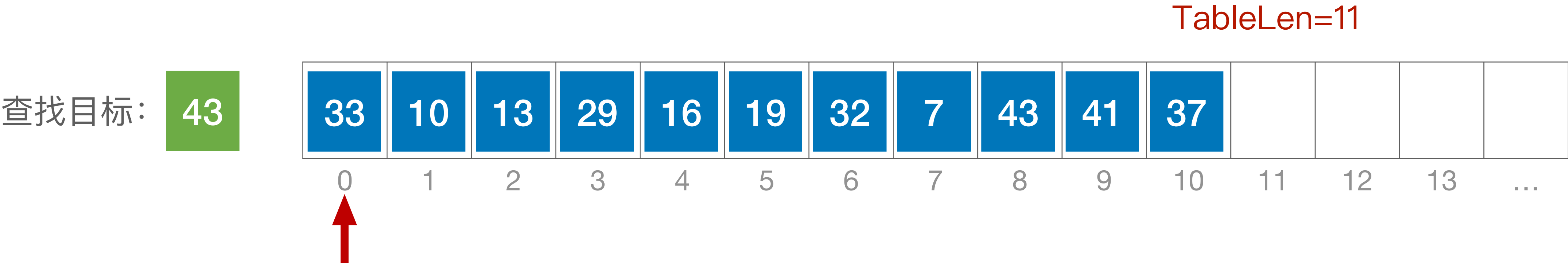
查找目标：

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;

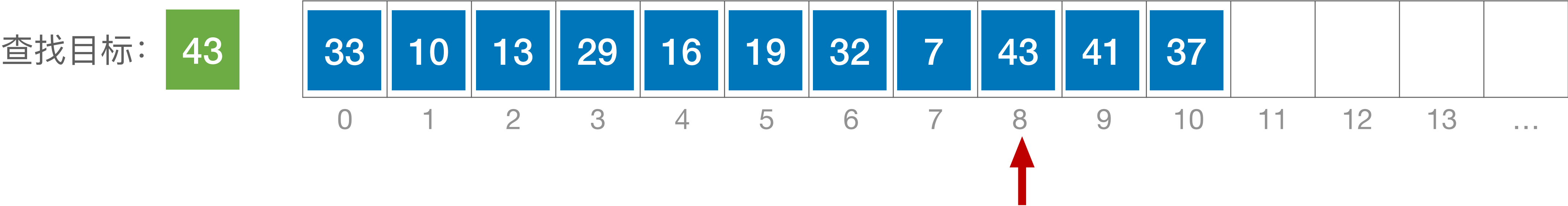
//顺序查找
int Search_Seq(SSTable ST,ElemType key){
    int i;
    for(i=0;i<ST.TableLen && ST.elem[i]!=key; ++i);
    //查找成功，则返回元素下标；查找失败，则返回-1
    return i==ST.TableLen? -1 : i;
}
```



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;

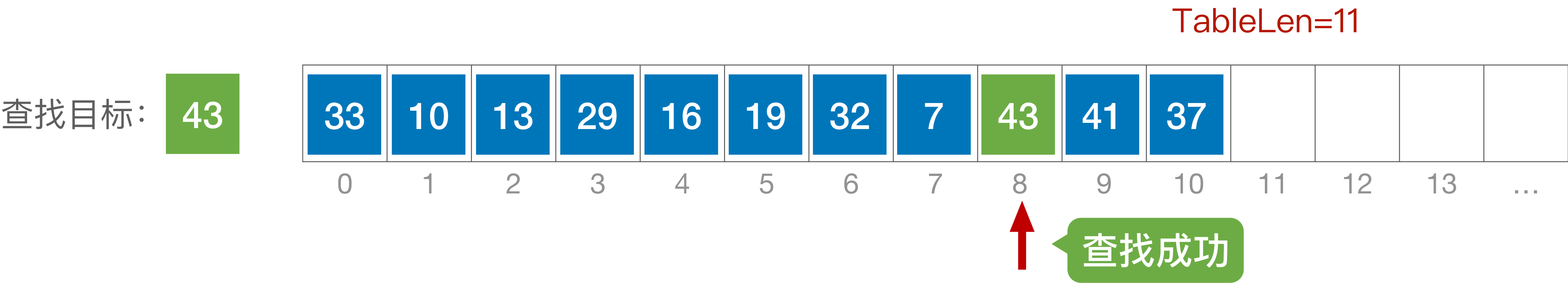
//顺序查找
int Search_Seq(SSTable ST,ElemType key){
    int i;
    for(i=0;i<ST.TableLen && ST.elem[i]!=key; ++i);
    //查找成功，则返回元素下标；查找失败，则返回-1
    return i==ST.TableLen? -1 : i;
}
```



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;

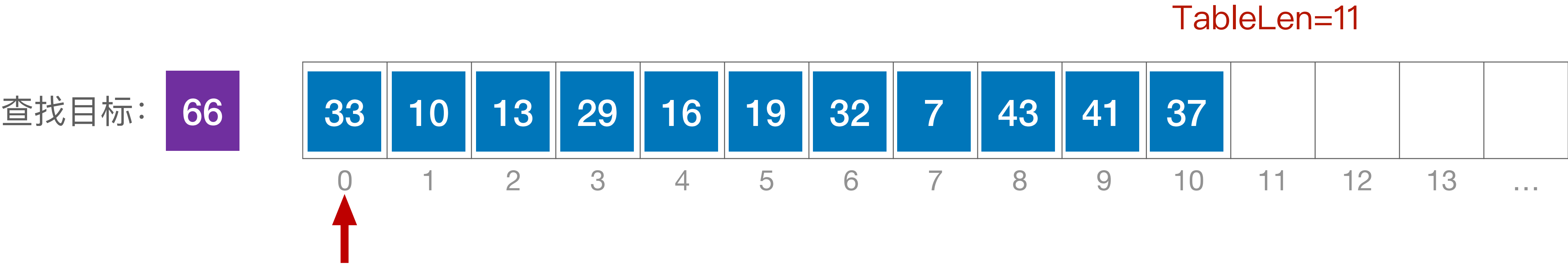
//顺序查找
int Search_Seq(SSTable ST,ElemType key){
    int i;
    for(i=0;i<ST.TableLen && ST.elem[i]!=key; ++i);
    //查找成功，则返回元素下标；查找失败，则返回-1
    return i==ST.TableLen? -1 : i;
}
```



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;

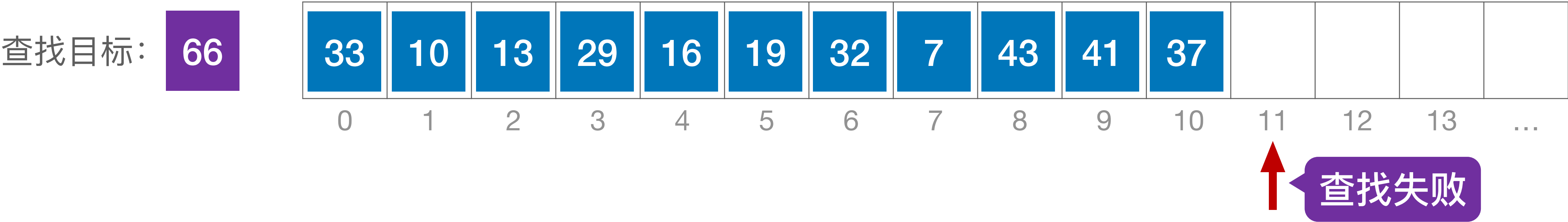
//顺序查找
int Search_Seq(SSTable ST,ElemType key){
    int i;
    for(i=0;i<ST.TableLen && ST.elem[i]!=key; ++i);
    //查找成功，则返回元素下标；查找失败，则返回-1
    return i==ST.TableLen? -1 : i;
}
```



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;
}
```

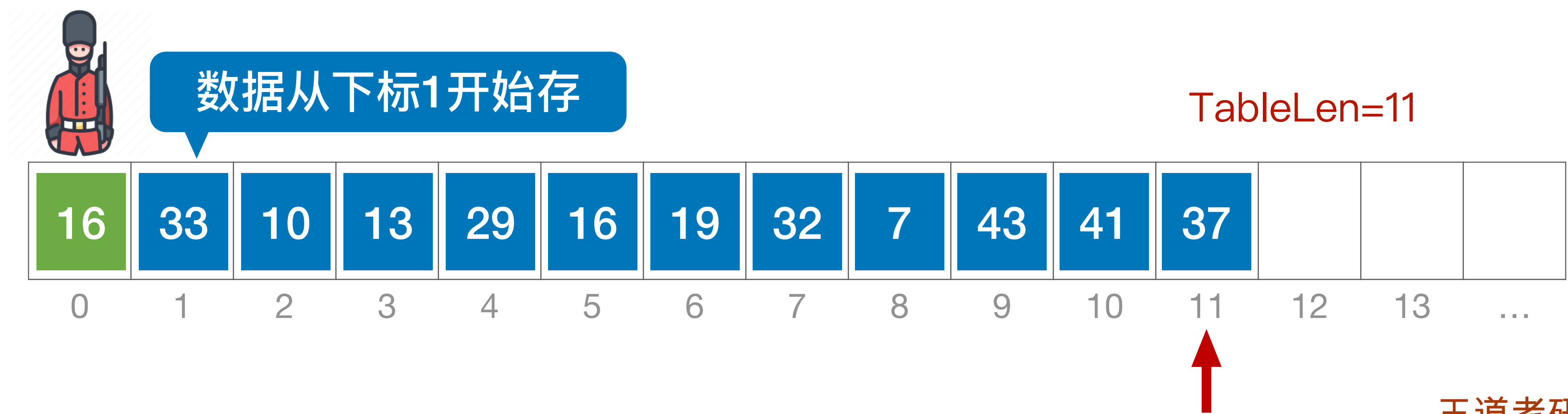


顺序查找的实现（哨兵）

```
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”



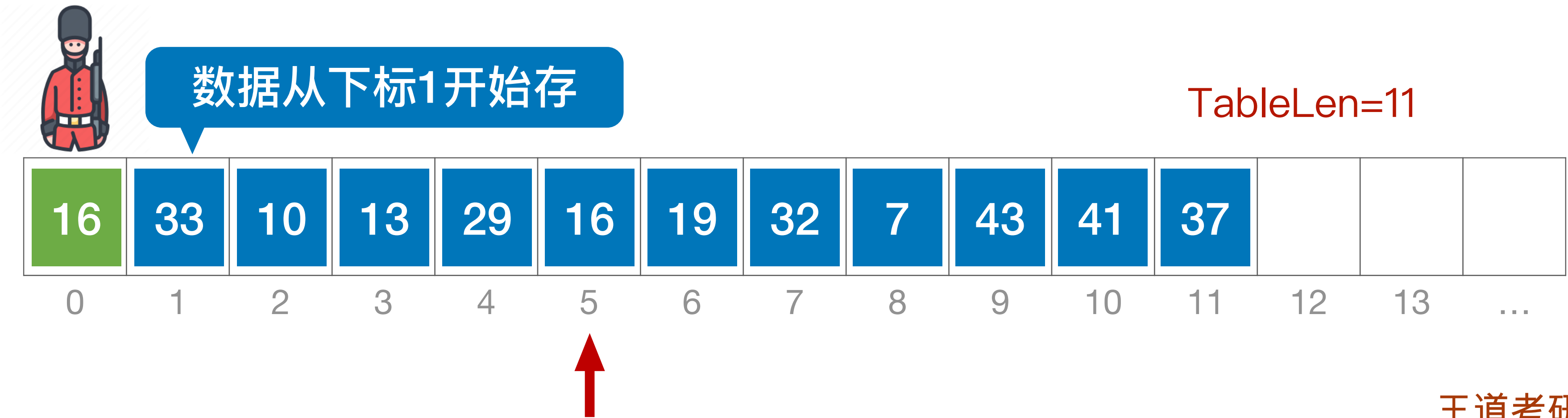
顺序查找的实现（哨兵）

```
typedef struct{
    ElemType *elem;
    int TableLen;
}SSTable;

//查找表的数据结构（顺序表）
//动态数组基址
//表的长度

//顺序查找
int Search_Seq(SSTable ST,ElemType key){
    ST.elem[0]=key;
    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”



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”

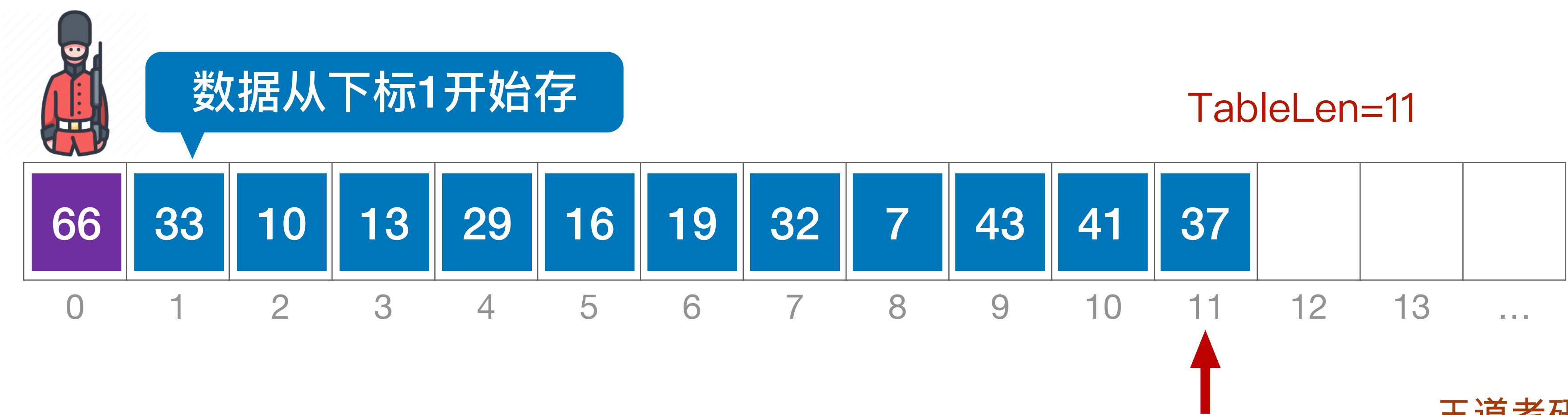


顺序查找的实现（哨兵）

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

//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”



顺序查找的实现（哨兵）

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

优点：无需判断是否越界，效率更高



```
//顺序查找  
int Search_Seq(SSTable ST,ElemType key){  
    ST.elem[0]=key;  
    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

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



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

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

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

$$ASL_{\text{失败}} = n + 1$$

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



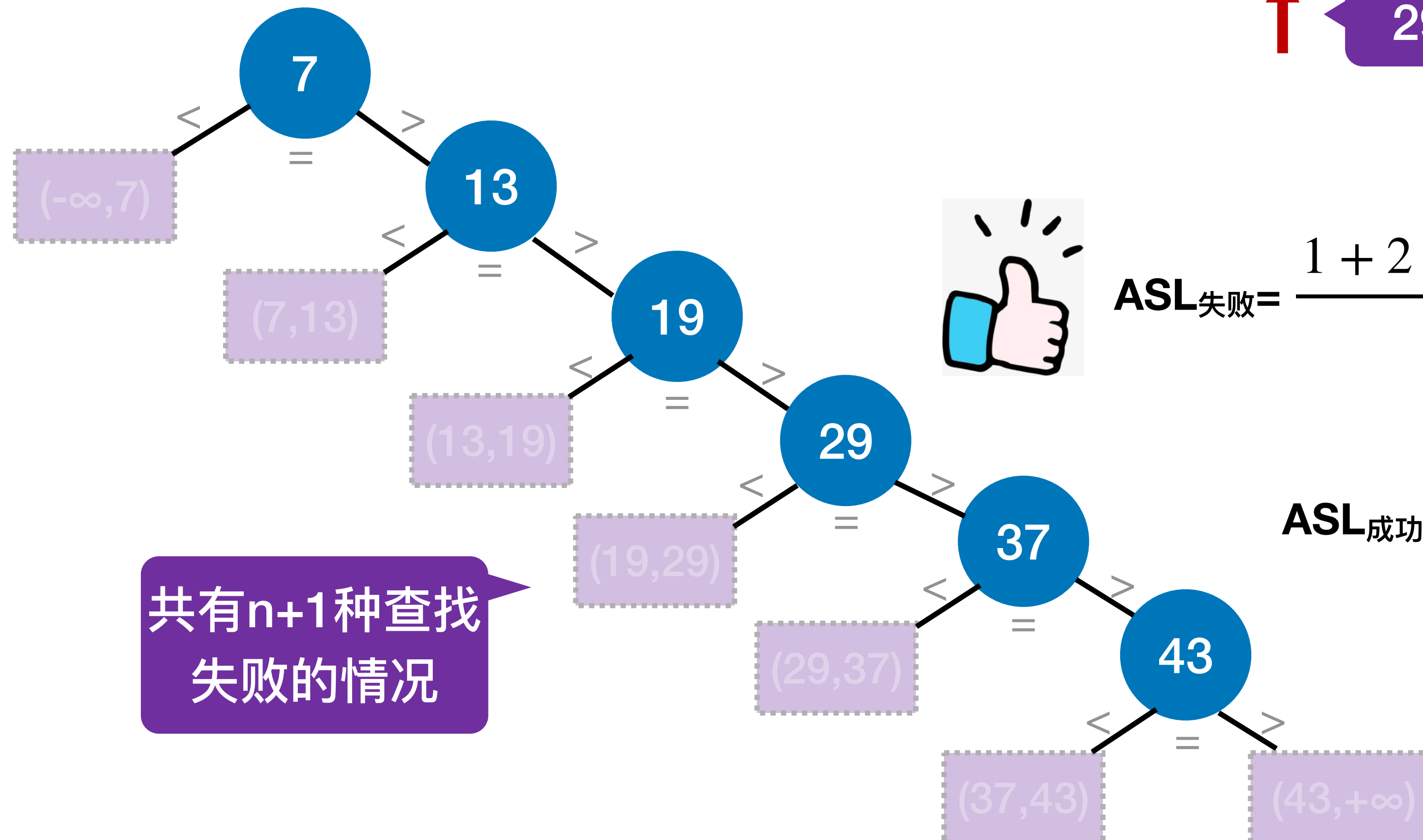
顺序查找的优化（对有序表）

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

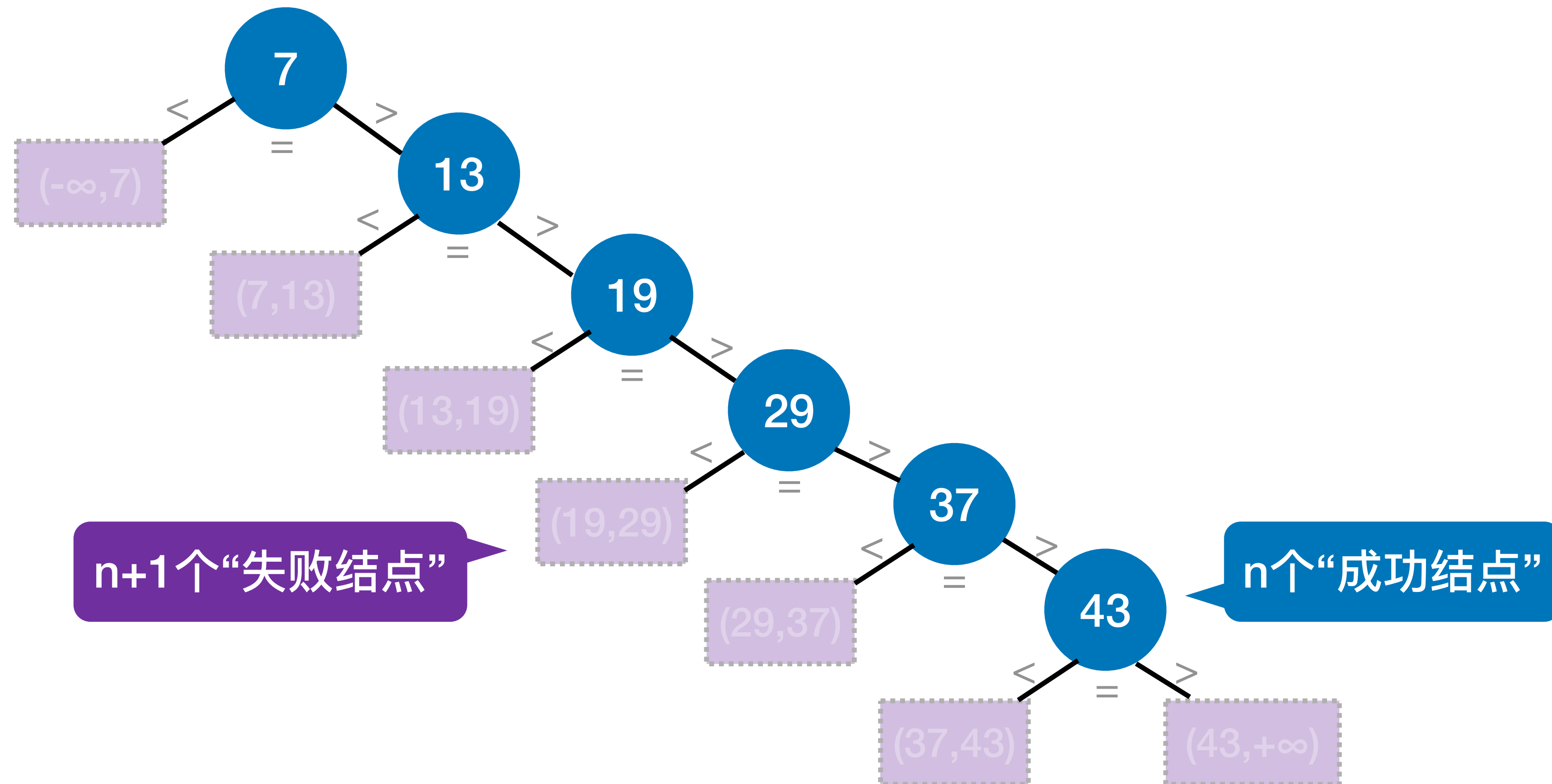
29 > 21 查找失败



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

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

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%



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

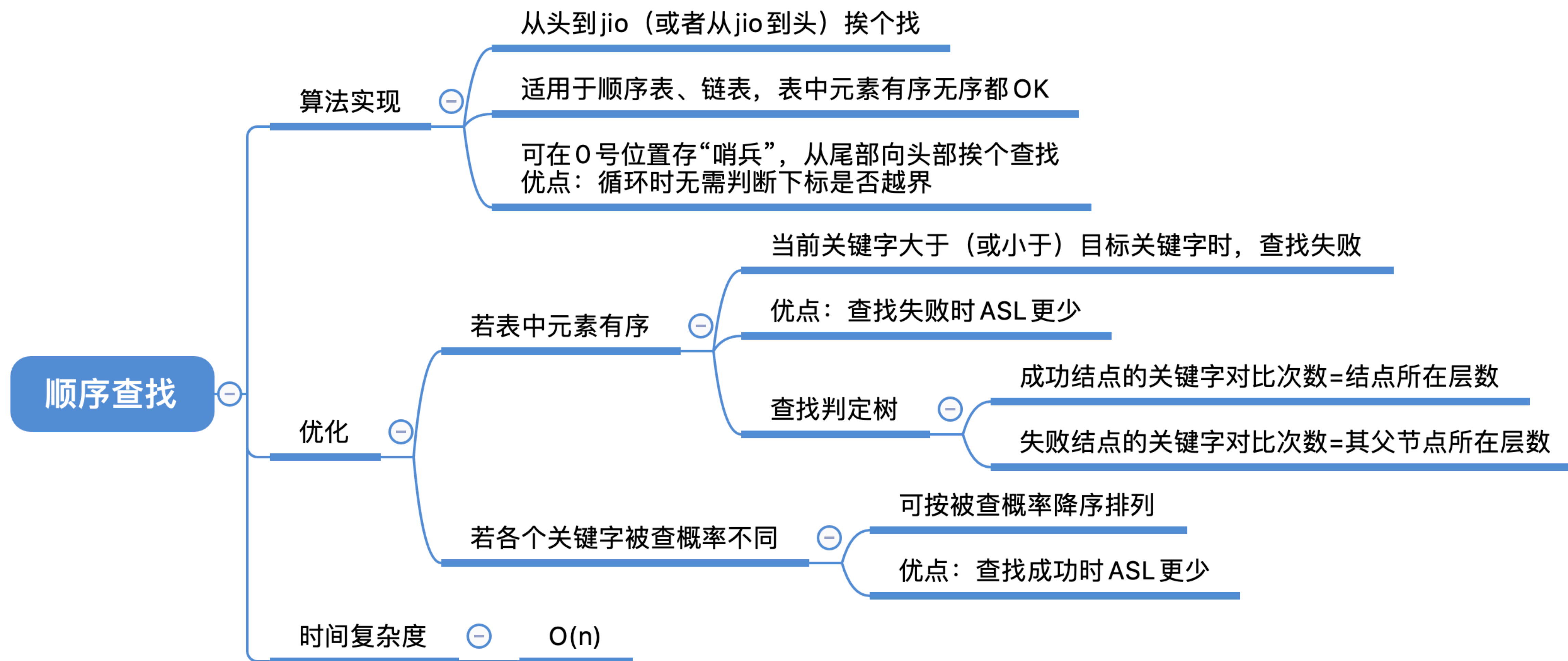
被查概率大的放在靠前位置




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

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

知识回顾与重要考点

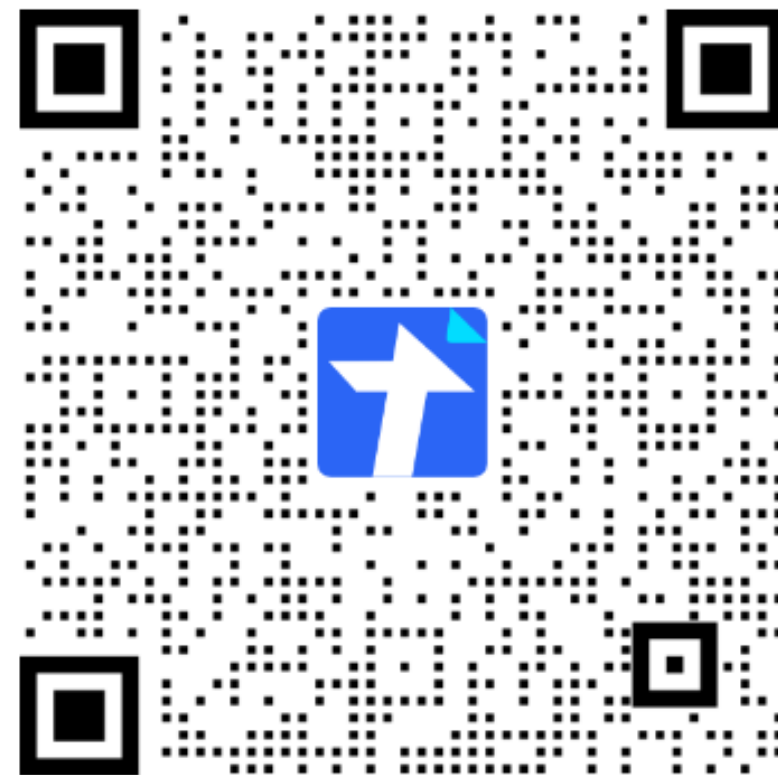


欢迎大家对本节视频进行评价~



学员评分：7.2.1 顺序查找

扫一扫二维码打开或分享给好友



— 腾讯文档 —

可多人实时在线编辑，权限安全可控



公众号：王道在线



b站：王道计算机教育



抖音：王道计算机考研