



Data Structures

Ch7

查找 Searching

2023 年 11 月 23 日

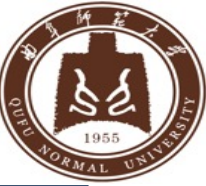
学而不厌 诲人不倦

- ➡ 7.1 概述
- ➡ 7.2 线性表查找技术：顺序查找与折半查找
- ➡ 7.3 树表的查找技术：二叉排序树与平衡二叉树
- ➡ 7.4 散列表查找技术
- ➡ 7.5 各种查找方法的比较
- ➡ 7.6 扩展与提高

本章的重点就是研究**查找表的存储方法**以及在此基础上的**查找方法**。

7.1 概述

7-1-1 查找的基本概念



7.1 概述

数据元素、结点、顶点

7-1-1 查找的基本概念



1. 关键码

- ✦ 关键码：可以标识一个记录_{记录}的某个数据项
- ✦ 键值：关键码的值
- ✦ 主关键码：可以唯一标识一个记录的关键码
- ✦ 次关键码：不能唯一标识一个记录的关键码

职工号	姓名	性别	年龄	工作时间
0001	王刚	男	48	1990.4
0002	张亮	男	35	2003.7
0003	刘楠	女	57	1979.9
0004	齐梅	女	35	2003.7
0005	李爽	女	56	1982.9



2. 查找

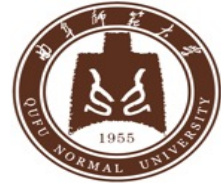
✦ **查找**：在相同类型的记录构成的集合中找出**满足给定条件**的记录

✍ 给定的查找条件可能是多种多样的

✍ 把查找条件限制为“**匹配**”，即查找**关键码等于给定值**的记录

✦ **查找的结果**：若在查找集合中找到了与给定值相匹配的记录，则称**查找成功**；否则，称**查找失败**

职工号	姓名	性别	年龄	工作时间
0001	王刚	男	38	1990.4
0002	张亮	男	25	2003.7
0003	刘楠	女	47	1979.9
0004	齐梅	女	25	2003.7
0005	李爽	女	50	1972.9



2. 查找

✦ **静态查找**：不涉及插入和删除操作的查找

静态查找只注重**查找**效率，适用于：

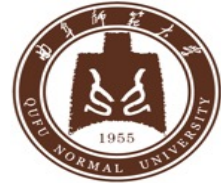
- (1) 查找集合一经生成，便只对其进行查找，而不进行插入和删除操作
- (2) 经过一段时间的查找之后，集中地进行插入和删除等修改操作

✦ **动态查找**：涉及插入和删除操作的查找

动态查找要求**插入**、**删除**、**查找**均有较好的效率，适用于：查找与插入和删除操作在同一个阶段进行

例如：当查找成功时，要删除查找到的记录

当查找不成功时，要插入被查找的记录



3. 查找结构

📌 **查找结构**：面向查找操作的数据结构，即查找基于的数据结构

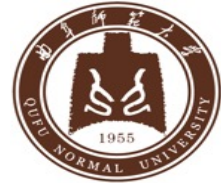
🕒 已经有了数据结构的概念，为什么要强调查找结构？

- (1) 几乎所有的数据结构都提供了查找作为基本操作，但对于数据结构整体来说，查找并不是最重要的操作
- (2) 对于查找结构来说，查找是最重要的基本操作，重要的是查找效率
- (3) **数据结构 + 算法 = 程序**：不同的查找结构，会获得不同的查找效率

查找结构



查找方法



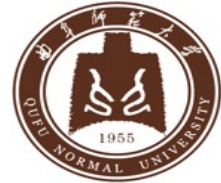
3. 查找结构

 查找基于的数据模型是什么？  集合


集合 {
 线性表：适用于静态查找，顺序查找、折半查找等技术
 树 表：适用于动态查找，二叉排序树的查找技术
 散列表：静态查找和动态查找均适用，采用散列技术

 注意到，都是把集合组织成XXX表，为什么？

理解起来，集合最常用的表示法是列举法，习惯上，也称为表



4. 查找算法的性能

 如何评价查找算法的效率呢? \Rightarrow 和关键码的比较次数

 平均查找长度: 查找算法进行的关键码比较次数的数学期望值

$$ASL = \sum_{i=1}^n p_i c_i$$

其中: n : 问题规模, 查找集合中的记录个数

p_i : 查找第 i 个记录的概率

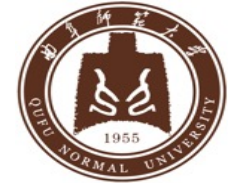
c_i : 查找第 i 个记录所需的关键码的比较次数

p_i 与算法无关, 取决于具体应用

c_i 取决于算法



如果 p_i 是已知的, 则平均查找长度只是问题规模的函数



7.2 线性表查找技术


7-2-1 顺序查找



7.2 线性表查找技术

7-2-1 顺序查找

1. 顺序查找的基本思想

 **顺序查找（线性查找）**：从线性表的一端向另一端**逐个**将记录与给定值进行比较，若相等，则**查找成功**，给出该记录在表中的位置；若整个表检测完仍未找到与给定值相等的记录，则**查找失败**，给出失败信息

```
int SeqSearch1 (int r[ ], int n, int k)
{
    int i = n;
    while (i > 0 && r[i] != k)
        i--;
    return i;
}
```

例：查找 35, 查找 25

0	1	2	3	4	5	6	7	8	9
	10	15	24	6	12	35	40	98	55

\uparrow
 i



7.2 线性表查找技术

7-2-1 顺序查找

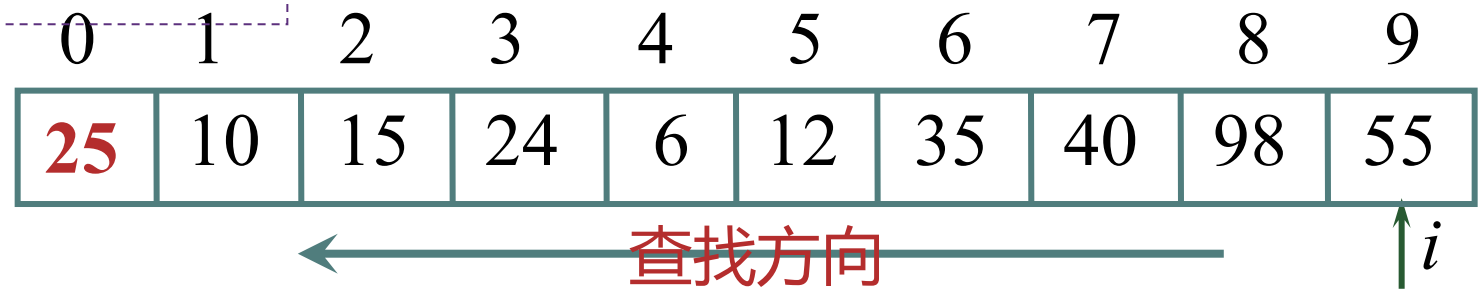
2. 顺序查找的改进

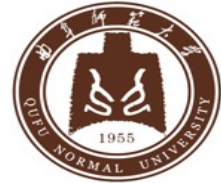
📎 顺序查找的改进：设置“哨兵”，就是待查值，放在查找方向的尽头处，免去了每一次比较后都要判断查找位置是否越界

```
int LineSearch :: SeqSearch2(int k)
{
    int i = n;
    data[0] = k;
    while (data[i] != k)
        i--;
    return i;
}
```

```
int LineSearch :: SeqSearch1 (int k)
{
    int i = n;
    while (i > 0 && data[i] != k)
        i--;
    return i;
}
```

例：查找 35，查找 25





3. 顺序查找的性能

```
int LineSearch :: SeqSearch(int k)
{
    int i = n;
    data[0] = k;
    while (data[i] != k)
        i--;
    return i;
}
```

查找成功:

$$\sum_{i=1}^n p_i c_i = \sum_{i=1}^n p_i (n - i + 1) = \frac{n+1}{2} = O(n)$$

查找不成功:

$$n + 1 = O(n)$$

0	1	2	3	4	5	6	7	8	9
	10	15	24	6	12	35	40	98	55

↑ i



7.2 线性表查找技术

7-2-1 顺序查找

4. 顺序查找的特点

📎 顺序查找的缺点：查找效率较低

特别是当待查找集合中元素较多时，不推荐使用顺序查找

📎 顺序查找的优点：算法简单而且使用面广

- (1) 对表中记录的存储没有任何要求，顺序存储和链接存储均可
- (2) 对表中记录的有序性也没有要求，无论记录是否按关键码有序均可

7.2 线性表查找技术

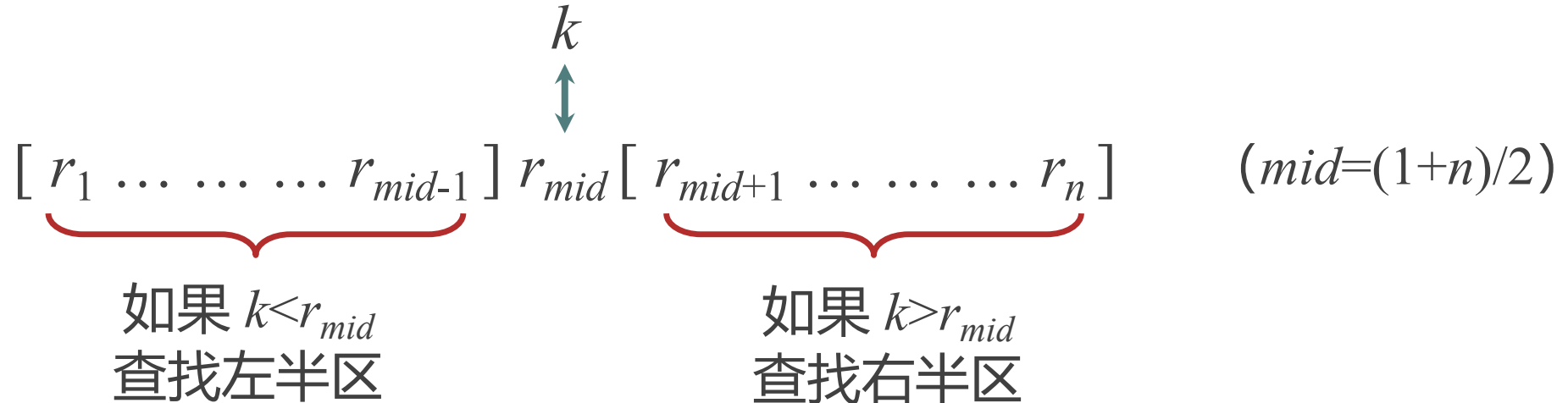
7-2-2 折半查找

7.2 线性表查找技术

7-2-2 折半查找

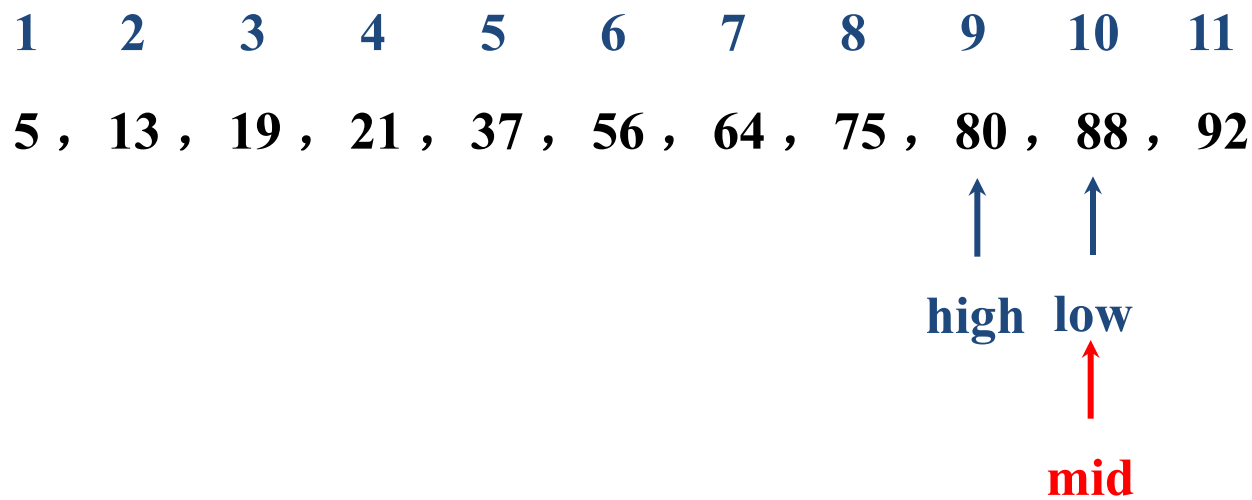
1. 折半查找

📎 折半查找（对半查找、二分查找）：在有序表（假设为递增）中，取中间记录作为比较对象，若给定值与中间记录相等，则查找成功；若给定值小于中间记录，则在有序表的左半区继续查找；若给定值大于中间记录，则在有序表的右半区继续查找。不断重复上述过程，直到查找成功，或查找区域无记录，查找失败



折半查找

$$\text{mid} = \lfloor (\text{low} + \text{high}) / 2 \rfloor$$



查找 21

mid = 6 high = mid - 1 = 5

mid = 3 low = mid + 1 = 4

mid = 4 找到

查找 85

mid = 6 low = mid + 1 = 7

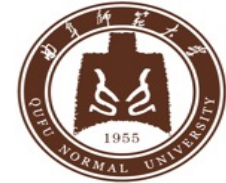
mid = 9 low = mid + 1 = 10

mid = 10 high = mid - 1 = 9

high < low 查找不成功

应用范围：顺序表，表内元素之间有序。不可直接用于线性链表。

7.2 线性表查找技术

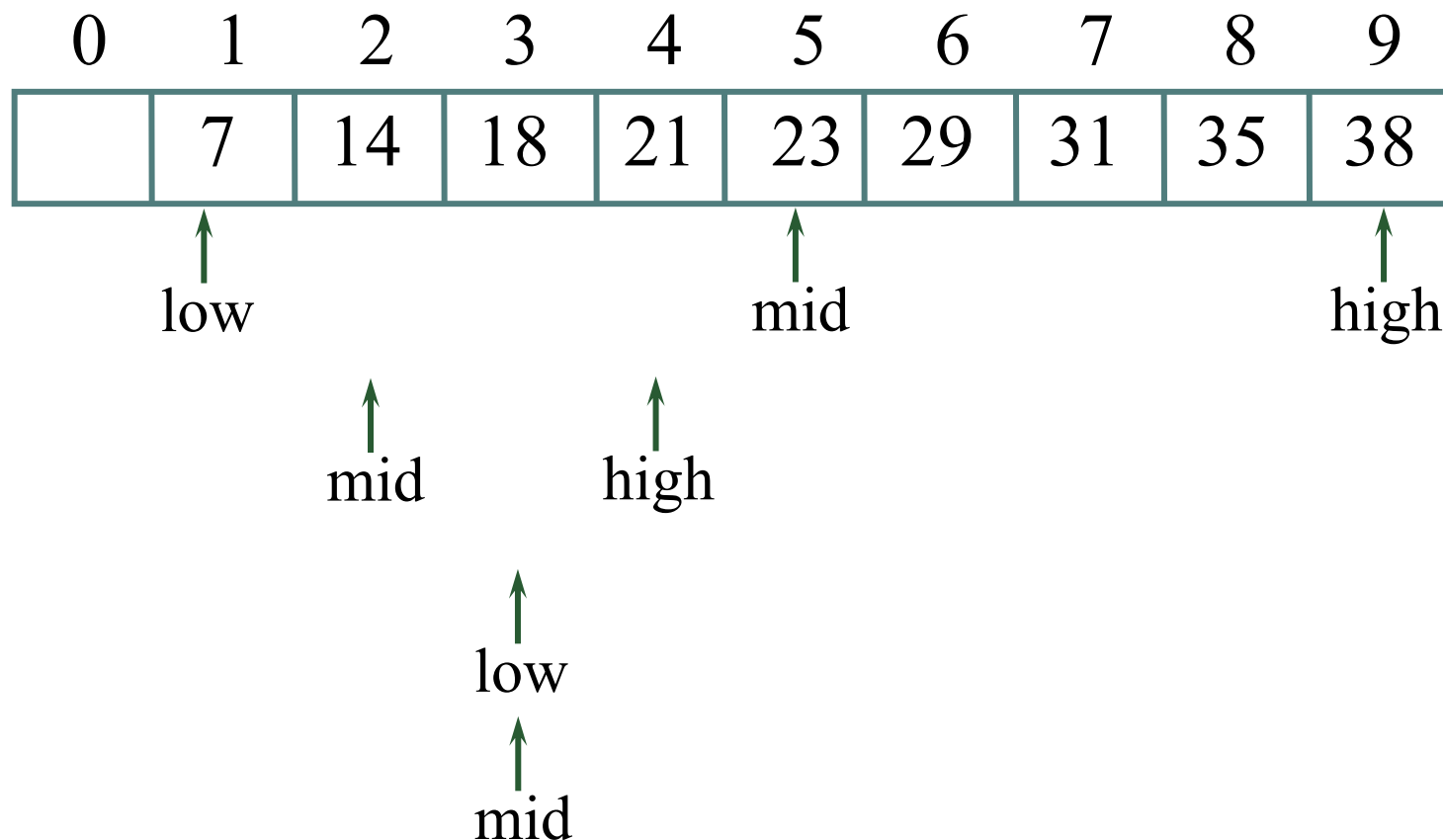


7-2-2 折半查找

1. 折半查找

例：查找 18

$$\text{mid} = \lfloor (\text{low} + \text{high}) / 2 \rfloor$$



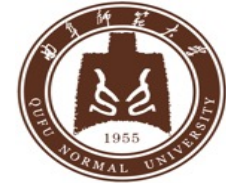
查找区间 [1, 9]

查找区间 [1, 4]

查找区间 [3, 4]

查找成功

7.2 线性表查找技术

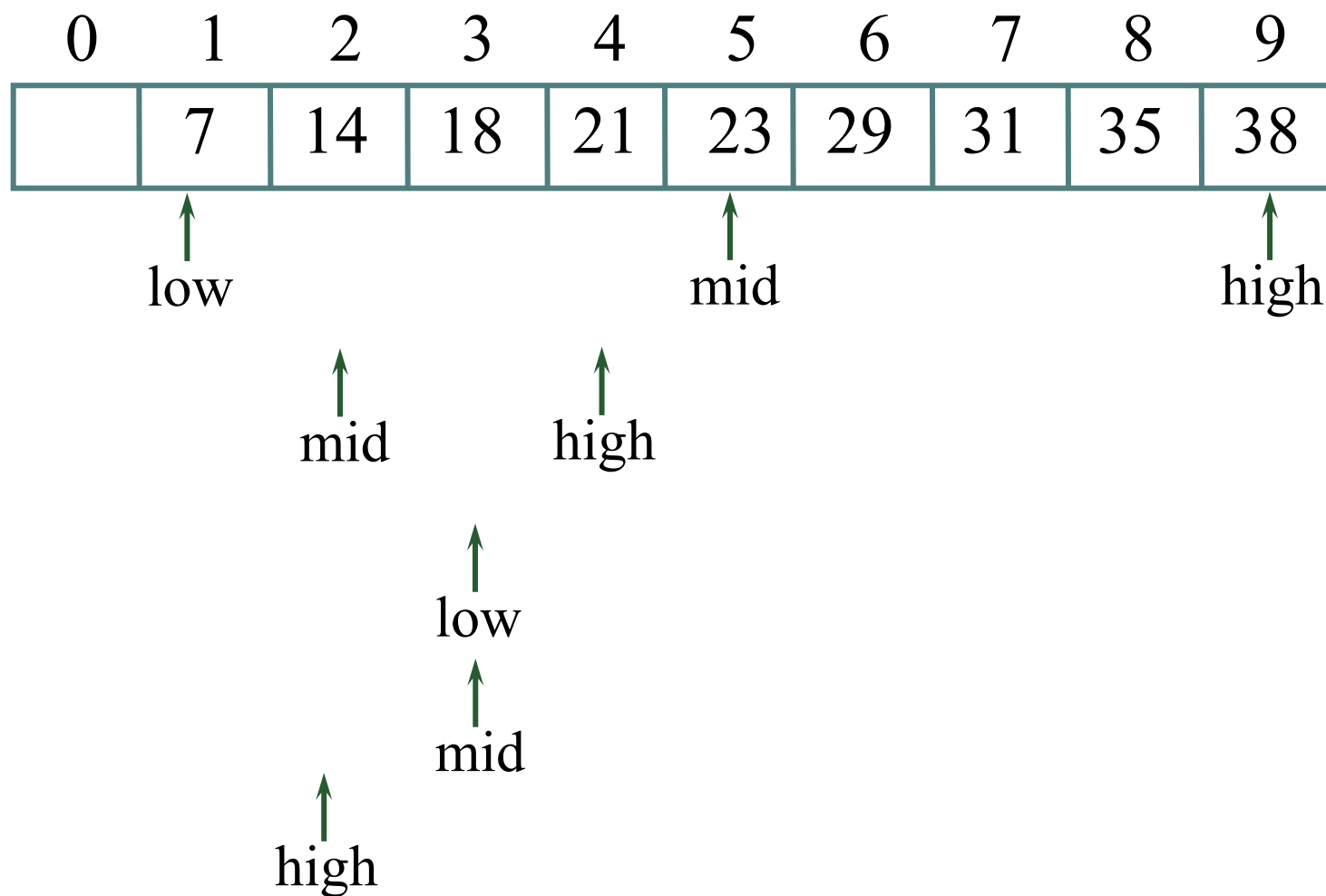


7-2-2 折半查找

1. 折半查找

例：查找 15

$$\text{mid} = \lfloor (\text{low} + \text{high}) / 2 \rfloor$$



查找区间 [1, 9]

查找区间 [1, 4]

查找区间 [3, 4]

查找区间 [3, 2]

low > high, 查找失败



2. 非递归算法

```
int LineSearch :: BinSearch1(int k)           /*查找集合存储在r[1]~r[n]*/
{
    int mid, low = 1, high = n;               /*初始查找区间是[1, n]*/
    while (low <= high)                        /*当区间存在时*/
    {
        mid = (low + high) / 2;
        if (k < data[mid]) high = mid - 1;
        else if (k > data[mid]) low = mid + 1;
        else return mid;                      /*查找成功，返回元素序号*/
    }
    return 0;                                 /*查找失败，返回0*/
}
```

7.2 线性表查找技术

7-2-2 折半查找

3. 递归算法

例：查找 18

0	1	2	3	4	5	6	7	8	9
	7	14	18	21	23	29	31	35	38

low ↑ mid ↑ high ↑

high ↑

查找区间 [1, 9]

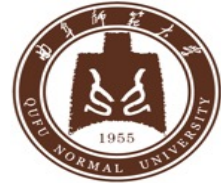
查找区间 [1, 4]

```
int LineSearch :: BinSearch2(int low, int high, int k)
{
    mid = (low + high) / 2;
    if (k < data[mid]) return BinSearch2(low, mid-1, k);
    else if (k > data[mid]) return BinSearch2(mid+1, high, k);
    else return mid;          /*查找成功，返回序号*/
}
```



3. 递归算法

```
int LineSearch :: BinSearch2(int low, int high, int k)
{
    int mid;
    if (low > high) return 0;           /*递归的边界条件*/
    else {
        mid = (low + high) / 2;
        if (k < data[mid]) return BinSearch2(low, mid-1, k);
        else if (k > data[mid]) return BinSearch2(mid+1, high, k);
        else return mid;               /*查找成功，返回序号*/
    }
}
```



4. 判定树

✦ 判定树（折半查找判定树）：描述折半查找判定过程的**二叉树**

📎 设查找区间是 $[low, high]$ ，判定树的构造方法：

- (1) 当 $low > high$ 时，判定树为空；
- (2) 当 $low \leq high$ 时，判定树的**根**结点是有序表中序号为 $mid = (low + high) / 2$ 的记录，根结点的**左子树**是与有序表 $r[low] \sim r[mid-1]$ 相对应的判定树，根结点的**右子树**是与有序表 $r[mid+1] \sim r[high]$ 相对应的判定树。

7.2 线性表查找技术

7-2-2 折半查找

4. 判定树

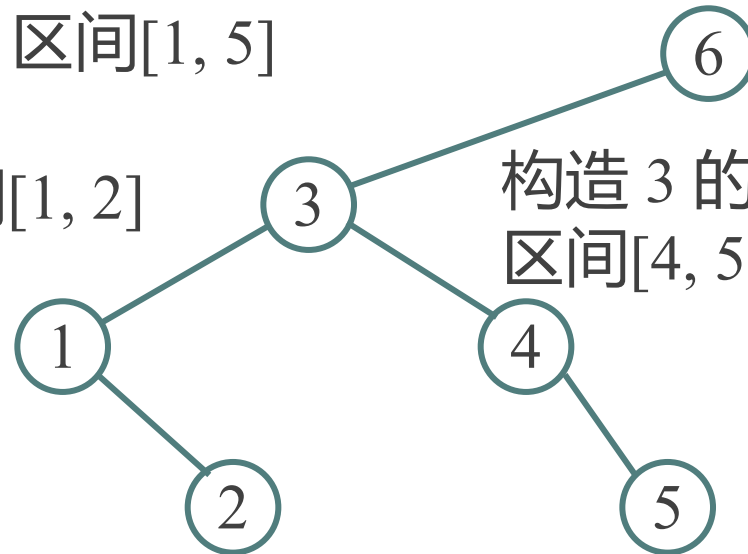
例如，结点个数（查找集合的记录个数）为11的判定树

查找区间 $[1, 11]$ ，中间记录的序号是6

构造 6 的左子树，区间 $[1, 5]$

构造 3 的左子树，区间 $[1, 2]$

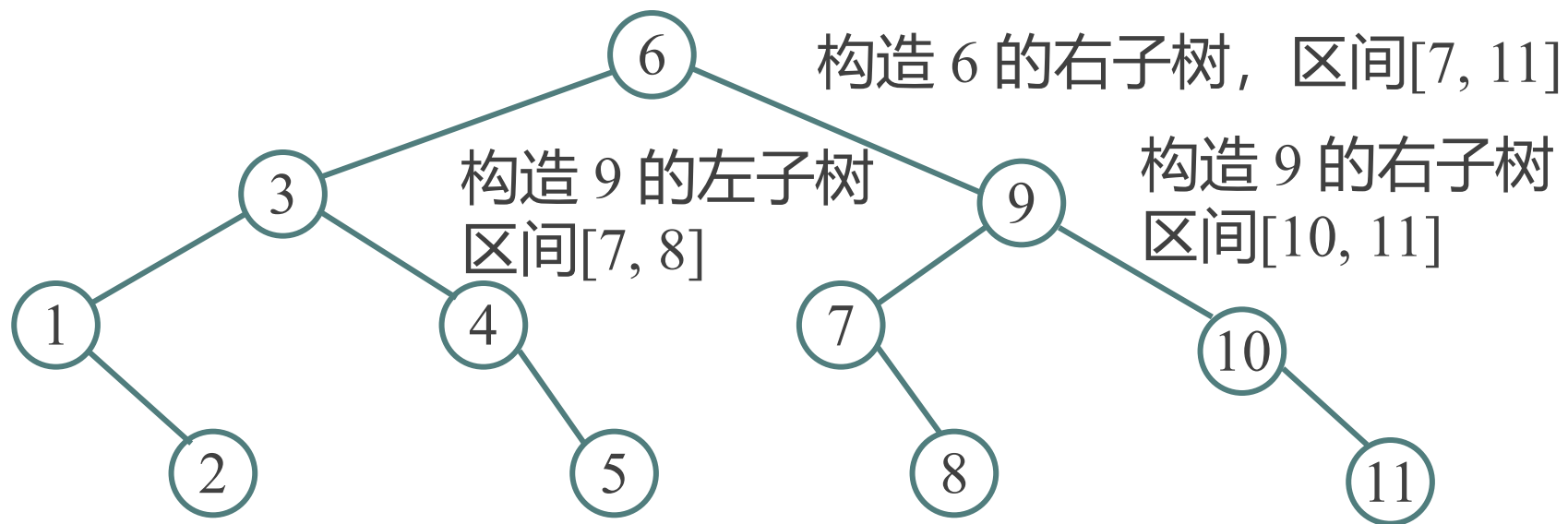
构造 3 的右子树
区间 $[4, 5]$





4. 判定树

例如，结点个数（查找集合的记录个数）为11的判定树



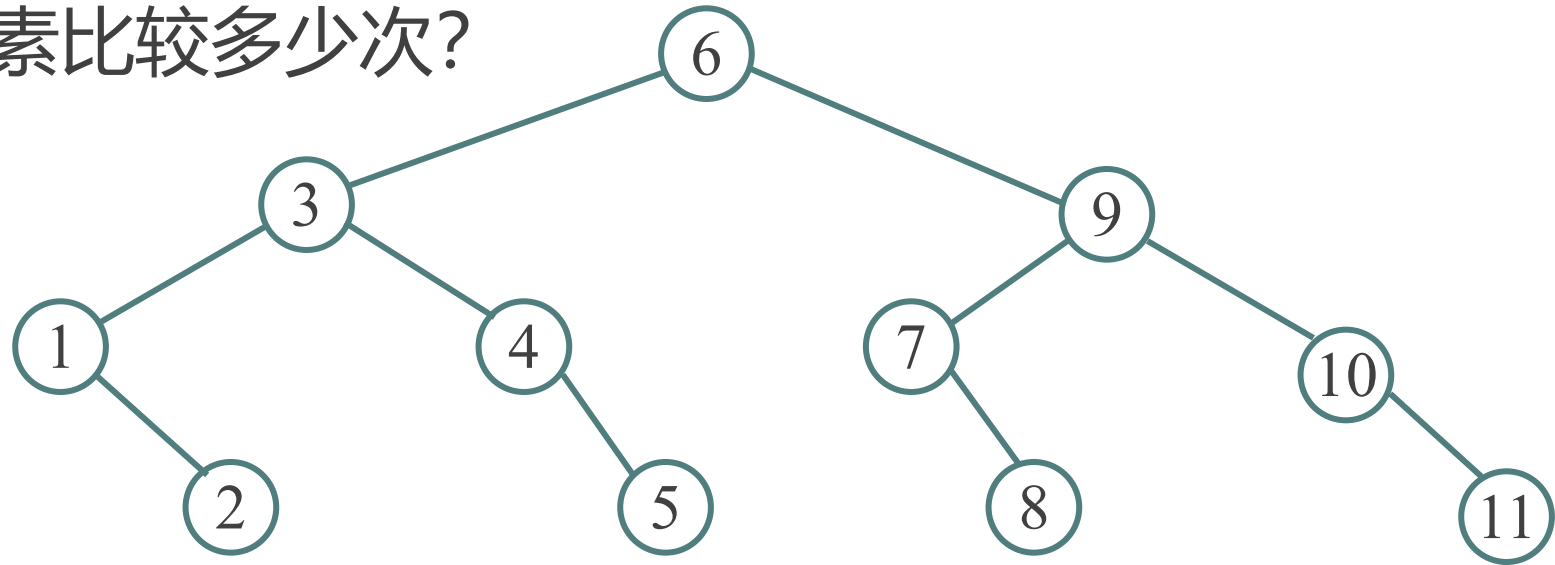
7.2 线性表查找技术

7-2-2 折半查找

5. 折半查找性能分析

例如，结点个数（查找集合的记录个数）为11的判定树

 查找第4个元素比较多少次？



折半查找任一记录的过程，即是判定树中从根结点到该记录结点的路径，和给定值的比较次数等于该记录结点在树中的层数

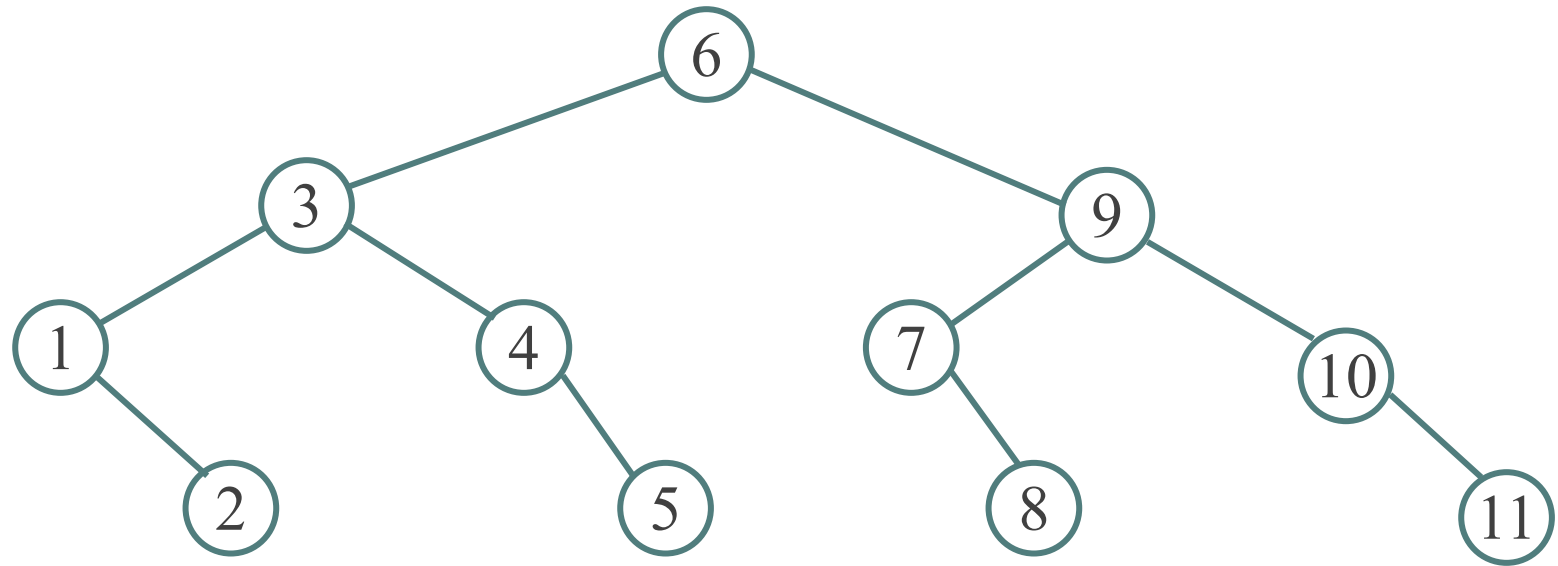
7.2 线性表查找技术

7-2-2 折半查找

5. 折半查找性能分析

例如，结点个数（查找集合的记录个数）为11的判定树

🕒 查找成功情况下，与判定树的深度有关，判定树的深度是多少呢？



判定树深度为 $\lfloor \log_2 n \rfloor + 1 \Rightarrow$ 比较次数至多为 $\lfloor \log_2 n \rfloor + 1 \Rightarrow$ 时间复杂度为 $O(\log_2 n)$

7.2 线性表查找技术

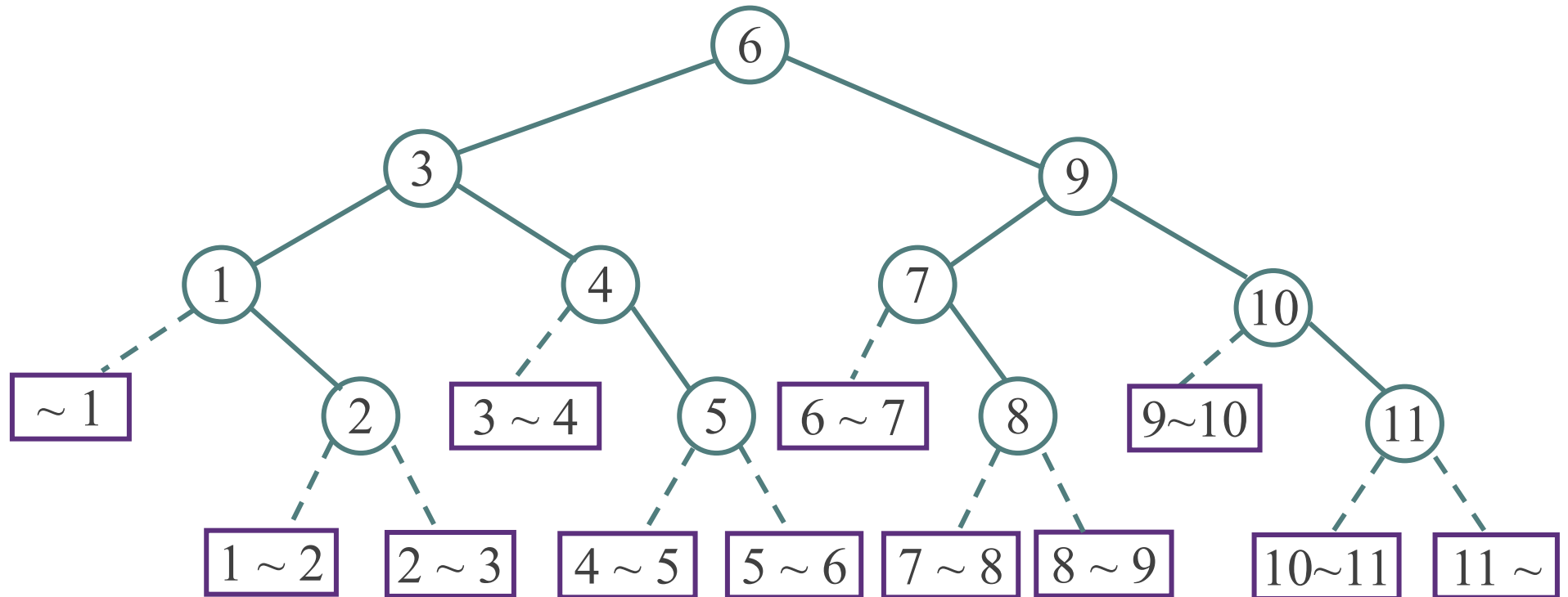
7-2-2 折半查找

5. 折半查找性能分析

例如，结点个数（查找集合的记录个数）为11的判定树

 如何确定查找失败呢？例如查找的元素比第3个元素大比第4个元素小？

查找不成功的过程是从根结点到**外部结点**的路径，和给定值进行的比较次数等于该路径上**内部结点的个数**。



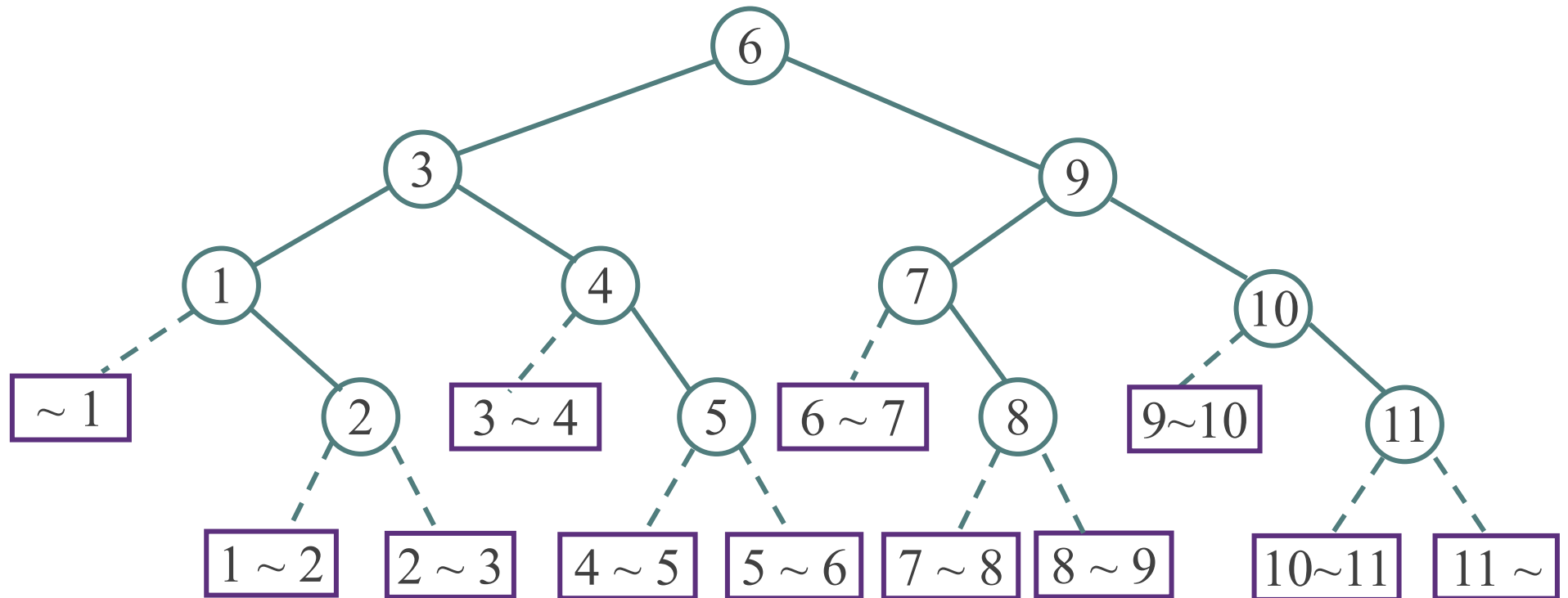
7.2 线性表查找技术

7-2-2 折半查找

5. 折半查找性能分析

✎ 查找成功的平均比较次数 = $(1 \times 1 + 2 \times 2 + 3 \times 4 + 4 \times 4) / 11 = 3$

✎ 查找不成功的平均比较次数 = $(3 \times 4 + 4 \times 8) / 12 = 11/3$





7.2 线性表查找技术

7-2-2 折半查找

5. 折半查找性能分析

设每个结点的查找概率相同都为 $1/n$ 。
设结点个数为 $n = 2^t - 1$ ($t = 1, 2, 3, \dots$)

平均情况分析 (在成功查找的情况下) :

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
4	8	9	10	11	13	19	29	32	47	65	77	81	93	99
4	3	4	2	4	3	4	1	4	3	4	2	4	3	4

∴ 经过 1 次比较确定的结点个数为 $1 = 2^0$ 个, 红色标识的结点。

经过 2 次比较确定的结点个数为 $2 = 2^1$ 个, 绿色标识的结点。

经过 3 次比较确定的结点个数为 $4 = 2^2$ 个, 灰色标识的结点。

经过 t 次比较确定的结点个数为 2^{t-1} 个, 蓝色标识的结点。

注意: ∴ $2^0 + 2^1 + 2^2 + \dots + 2^{t-1} = 2^t - 1$

∴ 最多经过 t 次比较可以找到有序表中的任何一个结点



7.2 线性表查找技术

7-2-2 折半查找

5. 折半查找性能分析

平均情况分析（在成功查找的情况下）：

$$\therefore ASL = (2^0 \times 1 + 2^1 \times 2 + 2^2 \times 3 + \dots + 2^{t-1} \times t) / n$$

$$= \left[\sum_{i=1}^t (i \times 2^{i-1}) \right] / n$$

$$= (n + 1) \times \log_2(n + 1) / n - 1$$

结论：在成功查找的情况下，

平均查找的代价约为 $ASL = \log_2(n + 1) - 1$

或者简单地记为： $ASL = \log_2 n - 1$

小结

1. 掌握查找的基本概念和算法评价方法
2. 掌握顺序查找和折半查找实现与性能分析方法



Thank You !

Q & A