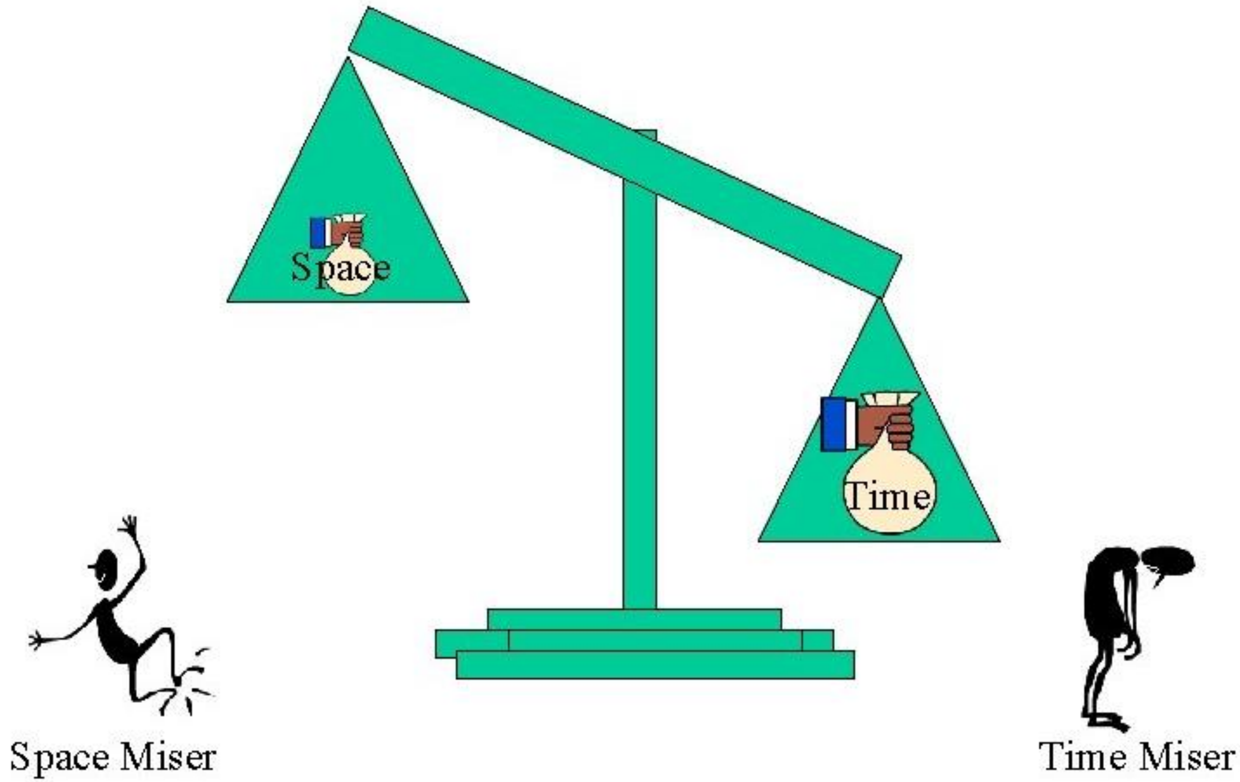
数据分析与算法设计
第7章 时空权衡
（Space and Time Tradeoffs）

李旻
百人计划研究员
浙江大学 信息与电子工程学院
Email: min.li@zju.edu.cn



Things which matter most must never be at the mercy of things which matter less.

舍卒保车，壮士断腕 —Johann Wolfgang von Göethe (1749–1832)

时空权衡

- 两种基本思想
  - 输入增强（Input Enhancement）：对问题的部分或全部输入做预处理，然后将获得的额外信息进行存储，以加速后面问题的求解
    - 计数法排序
    - Boyer-Moore字符串匹配及简化版本
  - 预构造(Prestructuring)：使用额外空间来实现更快和更方便的输入数据存储，从而提高算法的时间效率
    - 散列法（Hashing）
    - B树

目录

- 计数排序
- 字符串匹配中的输入增强技术
  - Horspool算法
  - Boyer-Moore算法
- 散列法
- B树

比较计数排序

- 算法思想：针对待排序数组中的每一个元素，统计数组中小于该元素的元素个数，并把结果记录在一张表中，则这个“个数”指出了该元素在有序表中的位置

数组 $A[0..5]$		62	31	84	96	19	47
初始	Count []	0	0	0	0	0	0
$i=0$ 遍之后	Count []	3	0	1	1	0	0
$i=1$ 遍之后	Count []		1	2	2	0	1
$i=2$ 遍之后	Count []			4	3	0	1
$i=3$ 遍之后	Count []				5	0	1
$i=4$ 遍之后	Count []					0	2
最终状态	Count []	3	1	4	5	0	2
数组 $S[0..5]$		19	31	47	62	84	96

比较计数排序

- 算法伪代码

算法 ComparisonCountingSort ($A[0..n-1]$)

//用比较计数法对数组排序

//输入：可排序数组 $A[0..n-1]$

//输出：将 A 中元素按照升序排列的数组 $S[0..n-1]$

初始化 → **for** $i \leftarrow 0$ **to** $n-1$ **do** $Count[i] \leftarrow 0$

for $i \leftarrow 0$ **to** $n-2$ **do**

  for $j \leftarrow i+1$ **to** $n-1$ **do**

    if $A[i]<A[j]$

      $Count[j] \leftarrow Count[j]+1$

    else $Count[i] \leftarrow Count[i]+1$

通过比较操作来统计小于第i个元素的元素个数

排序结果 → **for** $i \leftarrow 0$ **to** $n-1$ **do** $S[Count[i]] \leftarrow A[i]$

比较计数排序

- 算法效率
  - 基本操作：比较
  - 执行次数：

$$C(n)=\sum_{i=0}^{n-2}\sum_{j=i+1}^{n-1}1=\sum_{i=0}^{n-2}[(n-1)-(i+1)+1]=\sum_{i=0}^{n-2}(n-1-i)=\frac{n(n-1)}{2}$$

  - 效率类型：$O(n^2)$

增加了空间复杂度，但时间效率并没有提升！☹

分布计数排序

待排序的**整数数组**：假设最小值(l)和最大值(u)已知

13	11	12	13	12	12

统计每个元素在数组中出现的**频率**及**分布值**

数组值	11	12	13
频率	1	3	2
分布值	1	4	6

基于分布反向填充目标数组

	D[0..2]			S[0..5]					
$A[5] = 12$	1	**4**	6				12		
$A[4] = 12$	1	**3**	6			12			
$A[3] = 13$	1	2	**6**						13
$A[2] = 12$	1	**2**	5		12				
$A[1] = 11$	**1**	1	5	11					
$A[0] = 13$	0	1	**5**					13	

分布计数排序

- 算法伪代码

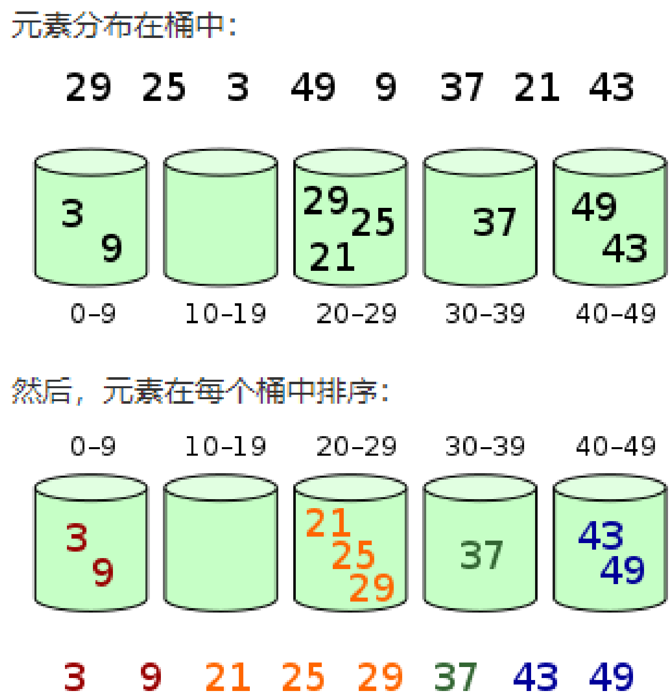
算法 DistributionCountingSort ($A[0..n-1], l, u$)
//用分布计数法，对来自于有限范围整数的一个数组进行排序
//输入：数组 $A[0..n-1]$，数组中的整数位于 l 和 u 之间($l \leqslant u$)
//输出：A 中元素构成的非降序数组 $S[0..n-1]$

for $j \leftarrow 0$ **to** $u-l$ **do** $D[j] \leftarrow 0$ // 初始化频率数组
for $i \leftarrow 0$ **to** $n-1$ **do** $D[A[i]-l] \leftarrow D[A[i]-l]+1$ // 计算频率值
for $j \leftarrow 1$ **to** $u-l$ **do** $D[j] \leftarrow D[j-1]+D[j]$ // 重用于分布
for $i \leftarrow n-1$ **downto** 0 **do**
    $j \leftarrow A[i]-l$
    $S[D[j]-1] \leftarrow A[i]$
    $D[j] \leftarrow D[j]-1$
return S

- 时间效率：O(n+k)（k为输入数组元素的范围）

桶排序

- 桶排序：将待排序数组n个元素均匀分布到K个“桶”中，然后对每个桶中的元素进行排序
  - 分布计数排序只使用了一个“桶”



基排序

- 基排序：将待排序数组元素统一为同样的数位长度（数位较短的元素前面补零），从最低位到最高位，依次进行一次排序（如计数排序），最后即可得到相应的排序数组

初始状态	按“个位”排序	按“十位”排序	按“百位”排序
053	542	003	003
542	053	014	014
003	003	214	053
063	063	616	063
014	014	542	154
214	214	748	214
154	154	053	542
748	616	154	616
616	748	063	748

十大排序算法

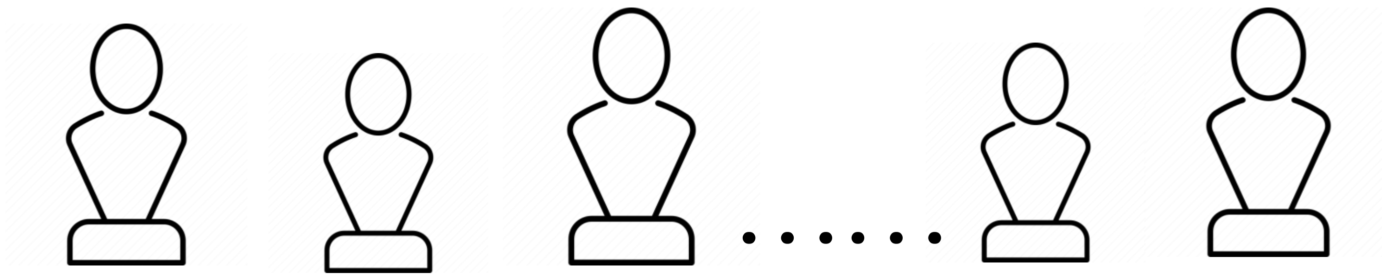
比较型排序

非比较型排序

排序方法	时间复杂度（平均）	时间复杂度（最坏）	时间复杂度（最好）	空间复杂度	稳定性
插入排序	$O(n^2)$	$O(n^2)$	$O(n)$	$O(1)$	稳定
希尔排序	$O(n^{1.3})$	$O(n^2)$	$O(n)$	$O(1)$	不稳定
选择排序	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(1)$	不稳定
堆排序	$O(nlog_2n)$	$O(nlog_2n)$	$O(nlog_2n)$	$O(1)$	不稳定
冒泡排序	$O(n^2)$	$O(n^2)$	$O(n)$	$O(1)$	稳定
快速排序	$O(nlog_2n)$	$O(n^2)$	$O(nlog_2n)$	$O(nlog_2n)$	不稳定
归并排序	$O(nlog_2n)$	$O(nlog_2n)$	$O(nlog_2n)$	$O(n)$	稳定
计数排序	$O(n+k)$	$O(n+k)$	$O(n+k)$	$O(n+k)$	稳定
桶排序	$O(n+k)$	$O(n^2)$	$O(n)$	$O(n+k)$	稳定
基数排序	$O(n*k)$	$O(n*k)$	$O(n*k)$	$O(n+k)$	稳定

讨论：最小距离排序问题

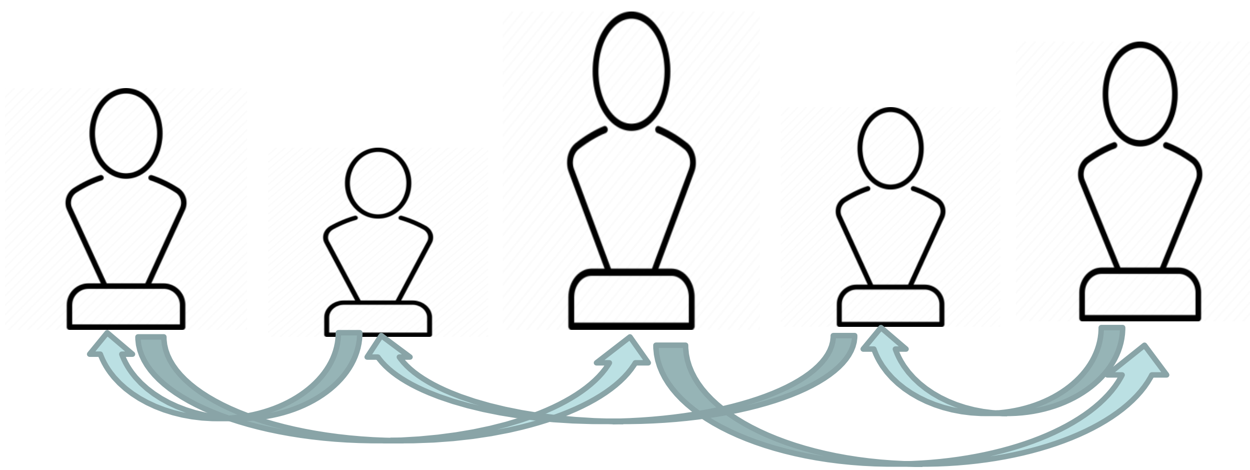
- 在美术馆大厅，有10个高度不同的石像放成一排。新的馆长希望移动它们使得它们按照高度从高到低顺序放置。请设计一种算法来完成该任务，使得所有石像的移动距离总和最小。



求解思路

- 按照比较计数排序的方法，计算比每个石像矮的石像总数，并记录在count数组中
- 基于这些计数，移动当前的位置为i的石像到它的目标位置count[i]
- 将原本位于count[i]处的石像移动到count[count[i]]位置处
- 重复上述两个步骤，直到所有石像都移动到正确的位置上

Count[0:4]　2　0　4　1　3



目录

- 计数排序
- 字符串匹配中的输入增强技术
  - Horspool算法
  - Boyer-Moore算法
- 散列法
- B树

字符串匹配

- 蛮力法（举例）
  - 文本：NOBODY_NOTICED_HIM
  - 模式：NOT

N	O	B	O	D	Y	_	N	O	T	I	C	E	D	_	H	I	M
N	O	~~T~~															
	~~N~~	O	T														
		~~N~~	O	T													
			~~N~~	O	T												
				~~N~~	O	T											
					~~N~~	O	T										
						~~N~~	O	T									
							N	O	T								

- 模式跟文本按从左到右的顺序做比较，若不匹配，则将模式右移一个字符，再做比较

是否可以动态调整模式的移动幅度，从而提高时间效率？

字符串匹配

- 基于输入增强技术的改进：对于模式进行预处理得到它的一些信息，储存在表中，然后在查找过程中策略性地使用这些信息，使得模式往后移动的距离尽可能大，从而加快匹配查找过程

- 著名的算法
  – KMP算法：Knuth-Morris-Pratt
  （模式匹配时，从左到右比较）
  – BM算法：Boyer-Moore
  （模式匹配时，从右到左比较）
  – Horspool算法：BM算法的简化版

Horspool算法

- 举例：在某个文本中查找模式BARBER：

$$\begin{array}{cccccccccc} s_0 & \ldots & & & & & c & \ldots & s_{n-1} \\ & & \mathrm{B} & \mathrm{A} & \mathrm{R} & \mathrm{B} & \mathrm{E} \; \mathrm{R} & & \end{array}$$

- 模式匹配查找方向：从右向左
- 根据模式中是否在c字符，调整右移距离
- 可分4种情况来讨论

Horspool算法

Case 1：模式中不存在c（c=S）

$s_0 \quad \ldots \quad S \quad \ldots \quad s_{n-1}$

$\not\parallel$

B A R B E R

B A R B E R

移动距离：右移模式长度m

Case 2:模式中存在唯一的c, 且为它的最后一个字符（c=R）

$s_0 \quad \ldots \quad M \;\; E \;\; R \quad \ldots \quad s_{n-1}$

$\not\parallel \;\; \parallel \;\; \parallel$

L E A D E R

L E A D E R

移动距离：右移模式长度m

Horspool算法

Case 3：模式中存在c，但不是它的最后一个字符（c=B）

$$
\begin{array}{ccccccccccccc}
s_0 & \ldots & & & & & \mathrm{B} & & & & & \ldots & s_{n-1}\\
 & & & & & & \not\parallel & & & & & & \\
 & & \mathrm{B} & \mathrm{A} & \mathrm{R} & \mathrm{B} & \mathrm{E} & \mathrm{R} & & & & & \\
 & & & & & \mathrm{B} & \mathrm{A} & \mathrm{R} & \mathrm{B} & \mathrm{E} & \mathrm{R} & &
\end{array}
$$

移动距离：右移模式直到最右边的c和文本的c对齐

Case 4：模式中存在多个c，包括它的最后一个字符（c=R）

$$
\begin{array}{cccccccccccc}
s_0 & \ldots & & & & & \mathrm{A} & \mathrm{R} & & & \ldots & s_{n-1}\\
 & & & & & & \not\parallel & \parallel & & & & \\
 & \mathrm{R} & \mathrm{E} & \mathrm{O} & \mathrm{R} & \mathrm{D} & \mathrm{E} & \mathrm{R} & & & & \\
 & & & & \mathrm{R} & \mathrm{E} & \mathrm{O} & \mathrm{R} & \mathrm{D} & \mathrm{E} & \mathrm{R} &
\end{array}
$$

移动距离：右移直到前m-1个字符中最右边的c和文本的c对齐

Horspool算法

- 空间换时间：给定模式P，及匹配查找时可能碰到任意的字符c，预先计算出模式的右移距离t(c)，并把它们存储在索引表中

$$t(c)=\begin{cases}\text{模式的长度}m(\text{如果}c\text{不包含在模式的前}m-1\text{个字符中})\\ \text{模式前}m-1\text{个字符中最右边的}c\text{到模式最后一个字符的距离(在其他情况下)}\end{cases}$$

算法 ShiftTable ($P[0..m-1]$)

//用 Horspool 算法和 Boyer-Moore 算法填充移动表

//输入：模式 $P[0..m-1]$以及一个可能出现字符的字母表

//输出：以字母表中字符为索引的数组 $Table[0..size-1]$，

//表中填充的移动距离是通过公式(7.1)计算出来的

for $i \leftarrow 0$ **to** $size-1$ **do** $Table[i] \leftarrow m$

for $j \leftarrow 0$ **to** $m-2$ **do** $Table[P[j]] \leftarrow m-1-j$

return *Table*

Horspool算法

算法 HorspoolMatching ($P[0..m-1]$, $T[0..n-1]$)

```
//实现 Horspool 字符串匹配算法
//输入：模式 P[0..m−1]和文本 T[0..n−1]
//输出：第一个匹配子串最左端字符的下标，如果没有匹配子串，则返回 −1
ShiftTable(P[0..m−1])                    //生成移动表
i ← m−1                                   //模式最右端的位置
while i ≤ n−1 do
    k ← 0                                 //匹配字符的个数
    while k ≤ m−1 and P[m−1−k] = T[i−k] do
        k ← k+1
    if k = m
        return i−m+1
    else i ← i + Table[T[i]]
return −1
```

Horspool算法

- 应用示例：在一个由英文字母和空格构成的文本中查找模式BARBER

ShiftTable

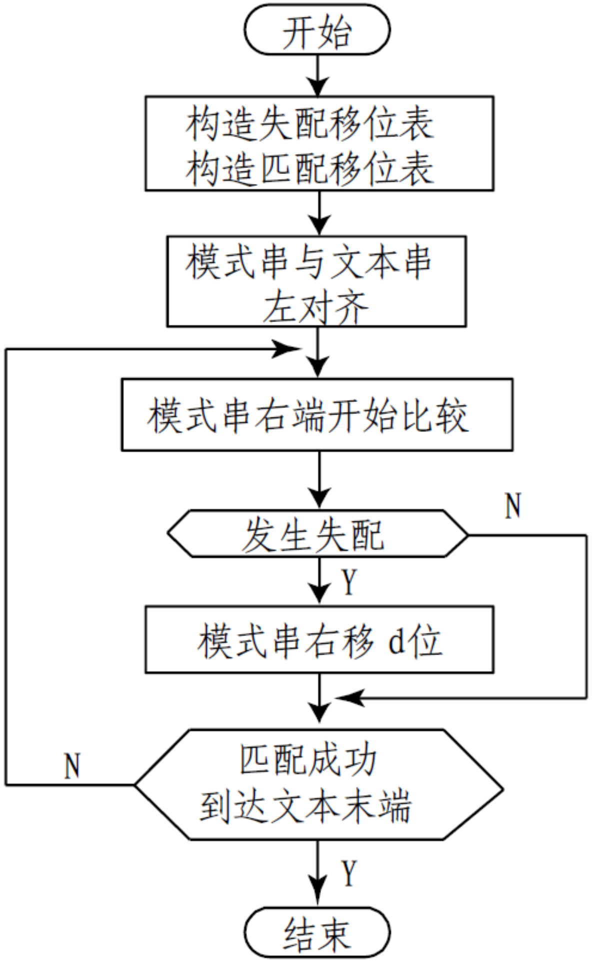
字符 c	A	B	C	D	E	F	…	R	…	Z	_
移动距离 $t(c)$	4	2	6	6	1	6	6	3	6	6	6

```
J I M _ S A W _ M E _ I N _ A _ B A R B E R S H O P
B A R B E R                 B A R B E R
        B A R B E R             B A R B E R
          B A R B E R                 B A R B E R
```

- 算法效率：对于随机文本来说，效率类型为 $\Theta(n)$

Boyer-Moore算法

- 每次匹配，从模式的最右边字符开始，向左扫描
- 如果模式最右边字符和文本中对应字符c不匹配，BM算法采用的移动距离与Horspool算法一致
- 如果已经有k个字符成功匹配，BM算法则采用不同的移动距离



Boyer-Moore算法

- 当有k个字符成功匹配时，可基于以下两种方式计算移动距离
  - 坏符号移动
  - 好后缀移动

$$
\begin{array}{ccccccccccc}
s_0 & \cdots & & \boxed{c} & \boxed{s_{i-k+1} \quad \cdots \quad s_i} & & & \cdots & s_{n-1} & \text{text} \\
 & & & \nparallel & \parallel \qquad\qquad \parallel & & & & & \\
 & p_0 & \cdots & p_{m-k-1} & p_{m-k} \quad \cdots \quad p_{m-1} & & & & & \text{pattern}
\end{array}
$$

Boyer-Moore算法

- 坏符号移动

$$
\begin{array}{cccccccccccc}
s_0 & \ldots & & & S & E & R & & & & \ldots & s_{n-1} \\
 & & & & \nparallel & \parallel & \parallel & & & & & \\
 & & B & A & R & B & E & R & & & & \\
 & & & & & B & A & R & B & E & R &
\end{array}
$$

$$\boxed{t_1(\mathrm{S}) - 2 = 6 - 2 = 4}$$

$$
\begin{array}{ccccccccccc}
s_0 & \ldots & & & A & E & R & & & \ldots & s_{n-1} \\
 & & & & \nparallel & \parallel & \parallel & & & & \\
 & & B & A & R & B & E & R & & & \\
 & & & B & A & R & B & E & R & &
\end{array}
$$

$$\boxed{t_1(\mathrm{A}) - 2 = 4 - 2 = 2}$$

  - 移动距离公式：$\boxed{d_1 = \max\{t_1(c) - k, 1\}}$

Boyer-Moore算法

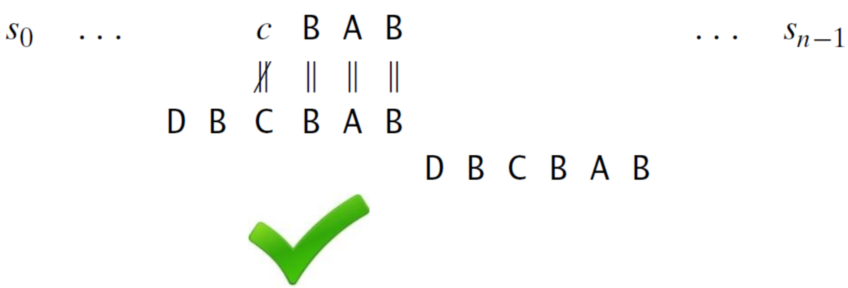
- 好后缀移动
  - 模式结尾长度为k的匹配部分记为suff(k)
  - 移动距离为d_2:
    - Case 1：若存在两个suff(k)且它们的前驱字符不同, 则d_2为从右数第二个suff(k)到最右边的suff(k)之间的距离

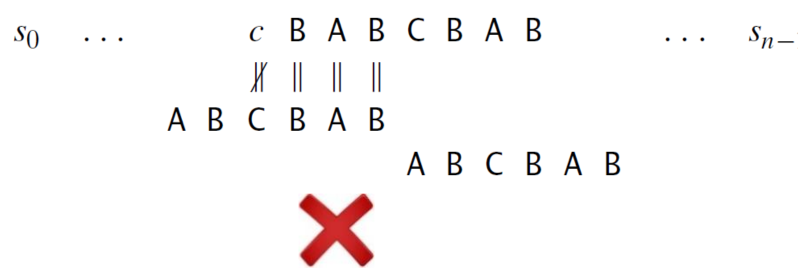
k	模 式	d_2
1	$\text{ABC}\overline{\text{B}}\text{A}\underline{\text{B}}$	2
2	$\overline{\text{AB}}\text{CB}\underline{\text{AB}}$	4

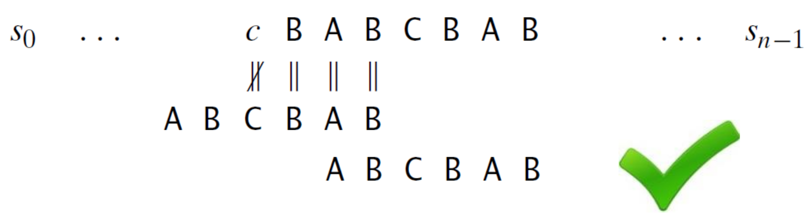
    - Case 2：若不存在另一个不同前驱字符的suff(k)，则需根据模式中的前缀（开头部分）和后缀（结尾部分）是否包含相同子串来做决定

Boyer-Moore算法

- Case 2：若不存在另一个不同前驱字符的suff(k)，则需根据模式中的前缀和后缀是否包含相同子串来做决定
  - 若不包含，则$d_2 = m$（模式长度）
  - 若包含，则找到长度为$l < k$的最长前缀，则d_2为对应的前缀与后缀之间的距离







Boyer-Moore算法

Boyer-Moore 算法

第一步： 对于给定的模式和在模式及文本中用到的字母表，按照给出的描述构造坏符号移动表。

第二步： 按照之前给出的描述，利用模式来构造好后缀移动表。

第三步： 将模式与文本的开始处对齐。

第四步： 重复下面的过程，直到发现了一个匹配子串或者模式到达了文本的最后一个字符以外：从模式的最后一个字符开始，比较模式和文本中的相应字符，直到要么所有 m 个字符都匹配(然后停止)，要么在 $k \geqslant 0$ 对字符成功匹配以后，遇到了一对不匹配的字符。在后一种情况下，如果 c 是文本中的不匹配字符，我们从坏符号移动表的第 c 列中取出单元格 $t_1(c)$的值。如果 $k > 0$，还要从好后缀移动表中取出相应的 d_2 的值。然后将模式沿着文本向右移动 d 个字符的距离。d 是按照以下公式计算出来的：

$$d = \begin{cases} d_1, & k = 0 \\ \max\{d_1, d_2\}, & k > 0 \end{cases}$$

其中，$d_1 = \max\{t_1(c) - k, 1\}$。

Boyer-Moore算法

- 举例：在一个由字母和空格构成的文本中查找BAOBAB
  - 坏符号移动表

c	A	B	C	D	…	O	…	Z	_
$t_1(c)$	1	2	6	6	6	3	6	6	6

  - 好后缀移动表

k	模　式	d_2
1	$\text{BAO}\overline{\text{B}}\text{A}\underline{\text{B}}$	2
2	$\overline{\text{B}}\text{AOB}\underline{\text{AB}}$	5
3	$\overline{\text{B}}\text{AO}\underline{\text{BAB}}$	5
4	$\overline{\text{B}}\text{A}\underline{\text{OBAB}}$	5
5	$\overline{\text{B}}\underline{\text{AOBAB}}$	5

Boyer-Moore算法

B E S S _ K N E W _ A B O U T _ B A O B A B S

B A O B A B

$d_1 = t_1(\text{K}) - 0 = 6$ B A O B A B

$d_1 = t_1(\_) - 2 = 4$ B A O B A B

$d_2 = 5$

$d = \max\{4, 5\} = 5$

$d_1 = t_1(\_) - 1 = 5$

$d_2 = 2$

$d = \max\{5, 2\} = 5$

B A O B A B

Horspool和Boyer-Moore算法比较

- Horspool算法
  - 最差效率：$O(nm)$
  - 随机文本：$\Theta(n)$
- Boyer-Moore算法
  - 最差效率：$\Theta(n)$
- Horspool算法更实用

目录

- 计数排序
- 字符串匹配中的输入增强技术
  - Horspool算法
  - Boyer-Moore算法
- 散列法
- B树

散列法（Hashing）

- 数据的存放方式直接影响相应的元素查找、插入和删除等操作的效率
- 记录的集合是一种常见的数据类型
  - 记录是由若干字段组成的，每个字段负责保存记录对象的一段特殊类型的信息
    - 举例：学生记录表的字段，分别代表学号、姓名、性别、专业等信息
  - 在记录的字段，通常至少会有一个被称为键的字段，来标识该条记录的对象
  - 如何有效地存放记录，以实现快速查找呢？

散列法（Hashing）

- 散列法：采用预先定义的函数对键进行计算，将键分布在一个称为散列表的一维数组中
  - 散列函数（hash function）

$$h:\mathbf{K}\rightarrow[0,1,2,...,m-1]$$

    - 函数值h(K)为键K指定一个散列地址

例：人口统计记录

编号	地区	总人口	汉族	回族	……

$h_1(key)$：取地区关键字第一个字母在字母表中的序号

$h_2(key)$：取第一个和最后一个字母在字母表中的序号之和，再除以30之后的余数

Key	BEIJING（北京）	TIANJIN（天津）	HEBEI（河北）	SHANXI（山西）	SHANGHAI（上海）	SHANDONG（山东）	HENAN（河南）
$h_1(key)$	02	20	08	19	19	19	08
$h_2(key)$	09	04	17	28	28	26	22

散列函数的构造

- 碰撞：散列表长度往往小于键的数量（“压缩”），可能存在两个或多个键分配到散列表的同一个地址
- “好”散列函数的构造准则：
  - 简单：函数易于计算
  - 均匀：把键在散列表的单元格中尽可能均匀地分布
- 常用构造方法
  - 直接定址法：h(key)= key; h(key)= a*key + b
  - 质数除余法：h(key)= key mod m
  - 平方取中法：key^2中间的几位数字（根据表长决定）
  - 折叠法、数字分析法、随机数法

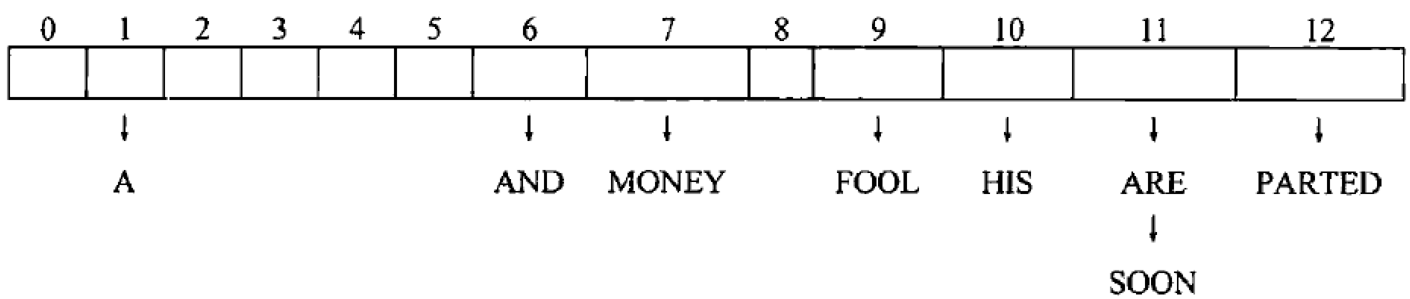
开散列（分离链）

- 开散列：分配到同一散列表单元格中的键用链表表示
- 示例

K 是一个字符串 $c_0c_1\cdots c_{s-1}$

$$h(K) = \left(\sum_{i=0}^{s-1} ord(c_i)\right) \bmod m$$

键	A	FOOL	AND	HIS	MONEY	ARE	SOON	PARTED
散列地址	1	9	6	10	7	11	11	12



开散列（分离链）

- 查找的效率
  - 取决于每个单元格的链表长度
  - 链表长度又取决于散列表的长度及散列函数的质量
  - 散列表的负载因子$\alpha = n/m$(n:键数；m:散列表长度)
    - 假设散列函数把键均匀分布在表的单元格中，对于随机选定的元素，在成功查找和不成功查找时平均需要检查的指针数分别为：

      $\mathrm{S} \approx 1 + \frac{\alpha}{2}$; $\mathrm{U} \approx \alpha$
- 插入、删除的效率：与查找的效率类型一致（$\Theta(1)$）

闭散列（开式寻址）

- 闭散列
  - 所有的键都存储在散列表中
  - 采用“线性探查”解决碰撞：如果发生碰撞，则检查碰撞处后面的单元格；若该单元格为空，则直接存储新的键；若该单元格不为空，则再检查它的后继，以此类推；若查找到达了散列表的尾部，则重新折回散列表的开始处

keys	A	FOOL	AND	HIS	MONEY	ARE	SOON	PARTED
hash addresses	1	9	6	10	7	11	11	12

0	1	2	3	4	5	6	7	8	9	10	11	12
	A											
	A								FOOL			
	A					AND			FOOL			
	A					AND			FOOL	HIS		
	A					AND	MONEY		FOOL	HIS		
	A					AND	MONEY		FOOL	HIS	ARE	
	A					AND	MONEY		FOOL	HIS	ARE	SOON
PARTED	A					AND	MONEY		FOOL	HIS	ARE	SOON

闭散列（开式寻址）

- 查找效率
  - 在成功查找和不成功查找的情况下，算法访问单元格的平均次数为

$$S \approx \frac{1}{2}(1+\frac{1}{1-\alpha}) \quad \text{and} \quad U \approx \frac{1}{2}(1+\frac{1}{(1-\alpha)^2})$$

不同的负载因子

α	$\frac{1}{2}(1+\frac{1}{1-\alpha})$	$\frac{1}{2}(1+\frac{1}{(1-\alpha)^2})$
50%	1.5	2.5
75%	2.5	8.5
90%	5.5	50.5

- 当散列表接近于满的时候，线性探查的性能会恶化

讨论：生日悖论问题

- 当一个房间里有多少人时，其中两个人生日（月和日）相同的概率大于1/2？这个问题的答案十分出人意料，请试着求解。对于散列来说这个结论意味着什么？

- （假设一年365天）

求解思路

1. 首先我们求所有人生日都不同的概率
2. 第一个人的生日可为$1 \sim 365$中的任意一个
3. 第二个人可选日期的个数为364个
4. ……
5. 第n个人的生日则可为$365-n+1$中的任意一个
6. 那么可得存在生日相同的概率为：

$$1-\frac{365}{365}*\frac{364}{365}*\cdots*\frac{365-n+1}{365}$$

求解过程

1. 令$E(n)$表示所有人生日都不同的概率
2. 则

$$E(n)=\prod_{i=1}^{n-1}(1-\frac{i}{365}).$$

3. 由均值不等式

$$H_n=\frac{n}{\sum\limits_{i=1}^{n}\frac{1}{x_i}}=\frac{n}{\frac{1}{x_1}+\frac{1}{x_2}+\cdots+\frac{1}{x_n}}$$，被称为调和平均数。

$$G_n=\sqrt[n]{\prod_{i=1}^{n}x_i}=\sqrt[n]{x_1x_2\cdots x_n}$$，被称为几何平均数。

$$A_n=\frac{\sum\limits_{i=1}^{n}x_i}{n}=\frac{x_1+x_2+\cdots+x_n}{n}$$，被称为算术平均数。

$$Q_n=\sqrt{\frac{\sum\limits_{i=1}^{n}x_i^2}{n}}=\sqrt{\frac{x_1^2+x_2^2+\cdots+x_n^2}{n}}$$，被称为平方平均数。

$$\prod_{i=1}^{n-1}(1-\frac{i}{365})<\left(\frac{1}{n-1}\sum_{i=1}^{n-1}\left(1-\frac{i}{365}\right)\right)^{n-1}$$

求解过程

4. 由等差数列

- $\left(\frac{1}{n-1}\sum_{i=1}^{n-1}\left(1-\frac{i}{365}\right)\right)^{n-1}=\left(1-\frac{n}{730}\right)^{n-1}$

5. 由不等式$1-x<e^{-x}$，可得

- $\left(1-\frac{n}{730}\right)^{n-1}<\left(e^{-\frac{n}{730}}\right)^{n-1}=e^{-n(n-1)/730}$

6. 当$n(n-1)>730\ln 2\approx 505.997$时

- $\left(1-\frac{n}{730}\right)^{n-1}<0.5$

求解过程

7. 取$n(n-1)=506$，此时$n=23$
8. 当一个房间里有23人时，其中两个人生日（月和日）相同的概率大于1/2

对散列来说，即使散列表大小比键值数大很多（大10倍以上），也会存在碰撞的可能性

目录

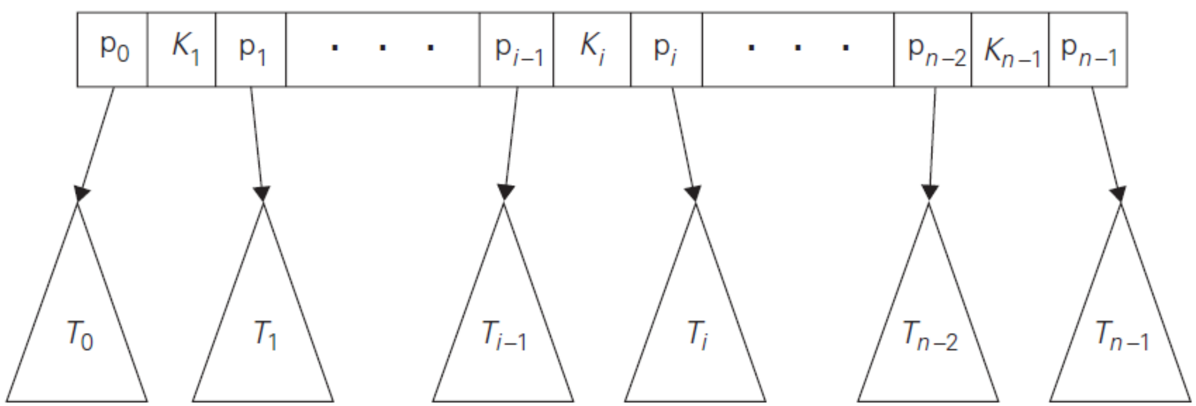
- 计数排序
- 字符串匹配中的输入增强技术
  - Horspool算法
  - Boyer-Moore算法
- 散列法
- B树

B树

- 当查找的文件较大，且存放在磁盘等直接存取设备中时，为了减少查找过程中对磁盘的读写次数，提高查找效率，基于直接存取设备的读写操作以“页”为单位的特征
- 1972年R.Bayer和E.M.McCreight提出一种称之为B−树的多路平衡查找树。它适合在磁盘等直接存取设备上组织动态的查找表
- B树的概念是一个描述型的“概念”，是B树能够运行从而表现出来的一种现象

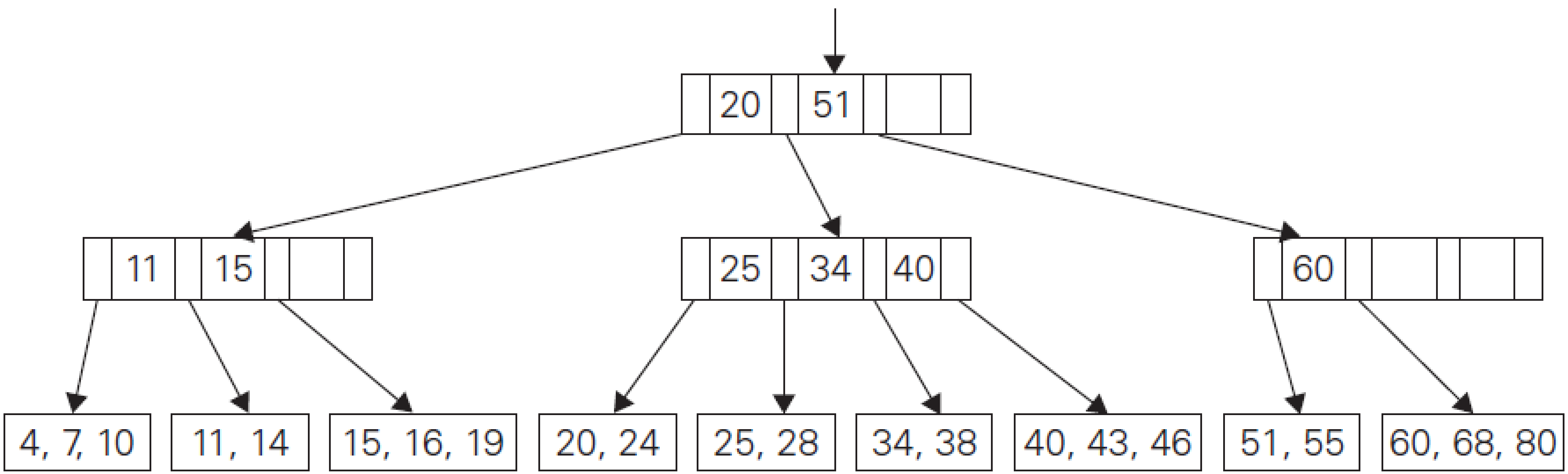
B树

- 基本结构
  - 所有的数据记录按照键的升序存储在叶子中
  - 以父母节点为索引，每个父母节点包含n-1个有序的键$K_1 < \cdots < K_{n-1}$
  - 这些键之间插入了n个指向节点子女的指针，使得子树T_0中的所有键都小于K_1，子树T_1中的所有键都大于等于K_1且小于K_2，以此类推，直到最后一棵子树T_{n-1}，它的键大于等于K_{n-1}，而K_{n-1}等于T_{n-1}中的最小键



B树

- 一棵次数(order)为m的B树满足以下特性
  - 根要么是一个叶子，要么是有2到m个子女
  - 除根和叶子以外的每个节点，有$\lceil m/2 \rceil$到m个子女
  - 为完美平衡树，所有叶子都在同一层



B树：查找操作

- B树的查找与二叉查找树的查找类似：从根开始，顺着一根指针链条前进，收敛到可能包含查找键的叶子

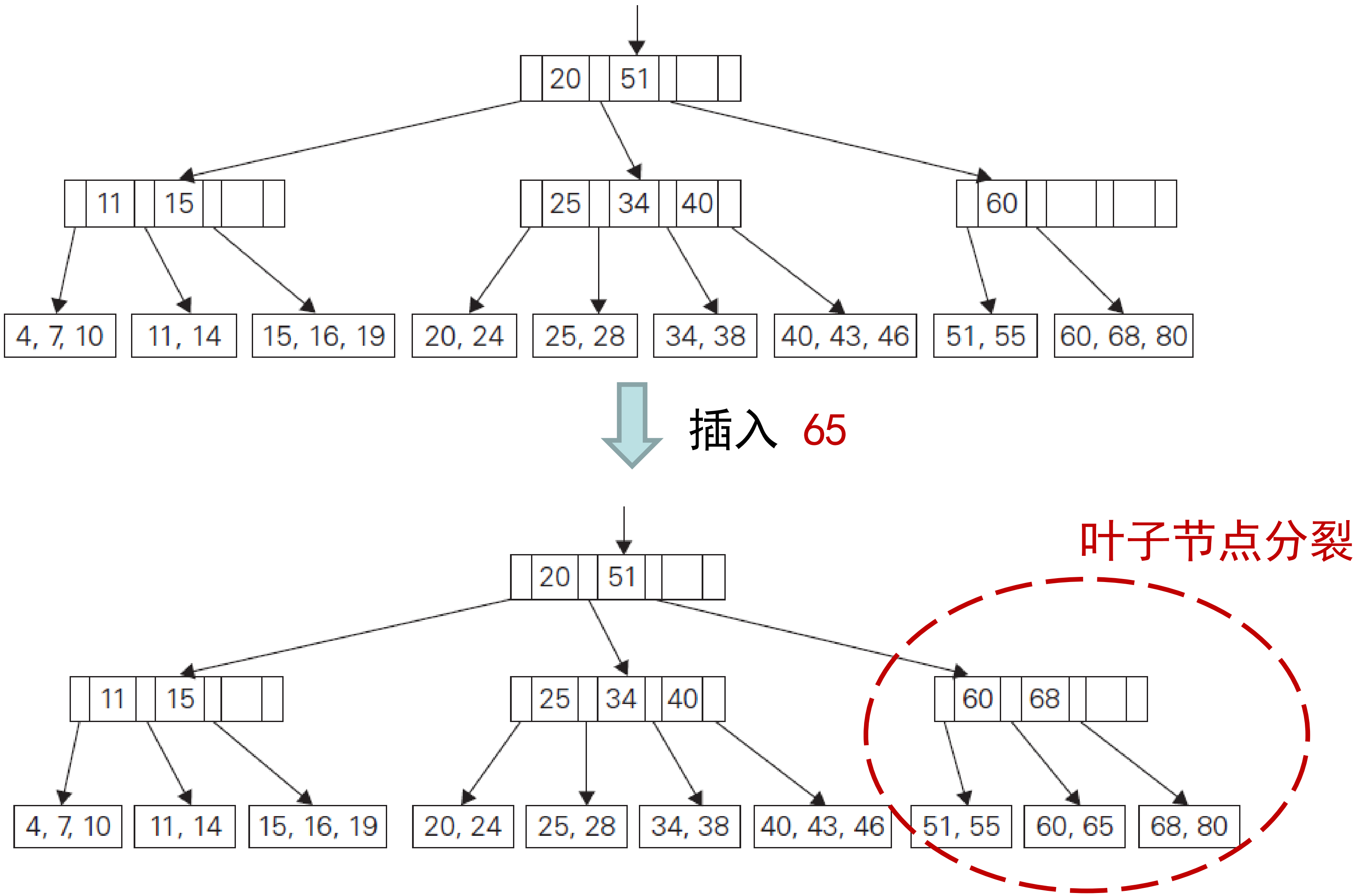
- 查找的效率取决于B树的高度
  - n个节点、次数为m、高度h>0的B树满足以下不等式

$$n \geq 1 + \sum_{i=1}^{h-1} 2\lceil m/2 \rceil^{i-1}(\lceil m/2 \rceil - 1) + 2\lceil m/2 \rceil^{h-1}$$

根至少包含1个键　　第i层节点至少包含的键数目　　最后一层至少包含的键数目

$$\Rightarrow \quad h \leq \left\lfloor \log_{\lceil m/2 \rceil} \frac{n+1}{4} \right\rfloor + 1$$

B树：插入操作


20 51
11 15
25 34 40
60
4, 7, 10
11, 14
15, 16, 19
20, 24
25, 28
34, 38
40, 43, 46
51, 55
60, 68, 80
插入 65
叶子节点分裂
20 51
11 15
25 34 40
60 68
4, 7, 10
11, 14
15, 16, 19
20, 24
25, 28
34, 38
40, 43, 46
51, 55
60, 65
68, 80

课后作业

章 X	节 X.Y	课后作业题 Z	思考题 Z
7	7.1	6	-
	7.2	2, 4	11
	7.3	1, 2	-
	7.4	6	-

注：只需上交“课后作业题”；以“学号姓名_chX.pdf ”规范命名，提交到“学在浙大”指定文件夹。DDL：2024年4月16日