# 数据结构

## 第 4 周 – 字符串：表示、编码、操作、模式匹配

Koczka Ferenc 博士 · 埃斯特哈齐·卡罗伊大学 · 2026

# 复习

- 数组是**同构、静态、顺序存储、直接访问**的
- 地址计算 → 访问 **O(1)** · 在中间插入/删除 **O(n)**
- **动态数组：**容量加倍 → 追加操作均摊 O(1)

## 今天的问题

- 机器如何存储文本 – 为什么 `len("世界") == 6`？
- 为什么在循环中用 `+` 拼接很慢?
- 怎样在文本中找到一个词?

# 字符串的概念

**字符串**（字符串、文本）是字符的有限序列。

- 本质上是一个**字符数组** – 再加上针对文本的操作
- **长度**：字符的个数 · **空串**：`""`（长度为 0）
- **子串**：连续的一段字符（`"data"` 是 `"database"` 的一部分）

| 性质 | 字符串 |
| --- | --- |
| 元素 | 同构（字符） |
| 存储 | 顺序 |
| 访问 | 直接（按下标） |
| 可变性 | Java、Go：**不可变** |

# 表示方法

**C：以 \0 结尾的字符数组**



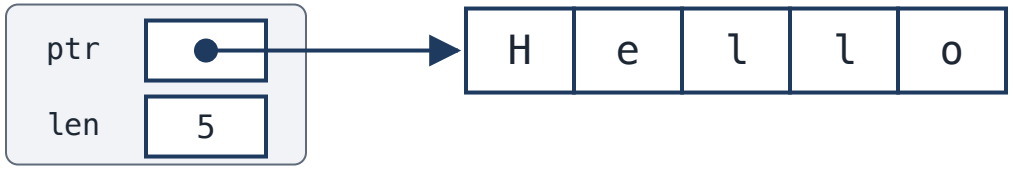

*求长度 O(n)：数到 \0 为止*

**Pascal：长度前缀**




*求长度 O(1)，但最多 255 个字符*

**Java / Go：头部（长度 + 指针）+ 字符数组**




*求长度 O(1)，内容不可变*

# 字符编码

# ASCII、Unicode、UTF-8

- **ASCII**(1963):7 位，128 个字符 – 英文字母、数字、标点
- **Unicode**：为每一种文字的每一个字符分配一个编号，即**码点**
  `A` = U+0041 · `é` = U+00E9 · `世` = U+4E16 · `😀` = U+1F600
- 码点必须转换成字节：这就是**编码**

| 编码 | 一个字符的大小 | 使用者 |
| --- | --- | --- |
| **UTF-8** | 1–4 字节（ASCII = 1 字节） | Web、Linux、**Go** |
| **UTF-16** | 2 或 4 字节 | **Java**、Windows、JavaScript |
| UTF-32 | 固定 4 字节 | 很少使用 |
| GB 2312 / GBK / GB 18030 | 1–4 字节 | 中国国家标准 |

# UTF-8：变长编码




Go:len ("Aé世😀") = 10 字节 · 4 个 rune Java:"Aé世😀".length() = 5 个 char (UTF-16) · 4 个码点

> “有多少个字符？”的答案取决于我们数的是什么：**字节**、**UTF-16 单元**还是**码点**。

# Java 和 Go 中的字符

**Java – UTF-16**

```
String s = "Aé世😀";
s.length();                    // 5(char)
s.codePointCount(0, s.length()); // 4
s.charAt(2);                   // '世'
s.getBytes(UTF_8).length;      // 10

// 按码点遍历:
s.codePoints().forEach(cp -> ...);
```

- `char` = 16 位 → 一个表情符号占**两个** `char`（代理对）
- 大多数汉字占**一个** `char`

**Go – UTF-8**

```
s := "Aé世😀"
len(s)                         // 10(字节!)
utf8.RuneCountInString(s)      // 4
s[2]                           // 0xA9 — 一个字节!
[]rune(s)[2]                   // '世'

for i, r := range s {          // 逐个 rune
    fmt.Printf("%d %c\n", i, r)
}
```

- `s[i]` 得到一个**字节**，`range` 得到一个 **rune**
- 需要逐字符处理时：`[]rune(s)`

# 字符运算

英文字母和数字的编码是**连续**的：

| | 编码 | 推论 |
|---|---|---|
| `'0'` … `'9'` | 48 … 57 | 数字的值：`c - '0'` |
| `'A'` … `'Z'` | 65 … 90 | 字母的序号：`c - 'A'` |
| `'a'` … `'z'` | 97 … 122 | 小写 → 大写：`c - 32` |

```
int digit = '7' - '0';                    // 7
char upper = (char) ('g' - ('a' - 'A')); // 'G'
int[] freq = new int[26];
freq[c - 'a']++;                          // 字母频率
```

用 26 个元素的数组统计字母频率很简单，但只适用于英文字母。一般情况下我们使用**哈希表**（第 11 周）。

# 字符串操作

# 不可变的字符串

在 Java 和 Go 中，字符串的内容**创建之后就不能改变**。

```
String s = "hello";
s.toUpperCase();           // 返回一个新字符串,s 不变!
s = s.toUpperCase();       // 应该这样写
```

**优点：**

- 可以安全地共享（在变量之间、在线程之间）
- 可以用作哈希表的键
- Java：**字符串常量池**中相同的字面量只存一份

**缺点：**每次“修改”都会创建新字符串 → **复制**

# 在循环中拼接：O(n²)！

```java
// 慢 — O(n²)
String s = "";
for (int i = 0; i < n; i++)
    s += "x";   // 新字符串 + 复制
```

```java
// 快 — O(n)
StringBuilder sb = new StringBuilder();
for (int i = 0; i < n; i++)
    sb.append("x");
String s = sb.toString();
```

Go: `strings.Builder` · `sb.WriteString("x")`

| n | += | Builder |
|---|---|---|
| 10 000 | 10 ms | 0.05 ms |
| 40 000 | 46 ms | 0.08 ms |
| 160 000 | 624 ms | 0.4 ms |

（Go，实测值）

`StringBuilder` 内部是一个**动态数组**（第 3 周） — 追加操作均摊 O(1)。

# 比较

**字典序：**从左往右逐个字符比较，第一个不同的字符决定大小；如果一个是另一个的前缀，较短的那个较小。

```
Function Compare(S, T)                  // -1、0 或 1
  i ← 1
  While i ≤ length(S) and i ≤ length(T) and S[i] = T[i]
    i ← i + 1
  End while
  If i ≤ length(S) and i ≤ length(T) then
    If S[i] < T[i] then Return -1 else Return 1
  else if length(S) = length(T) then Return 0
  else if length(S) < length(T) then Return -1
  else Return 1
End function
```

`"apple" < "apply"` · `"data" < "database"` · `"Zebra" < "apple"`（大写字母的编码更小！）

# Java 中的常见错误：`==` 与 `equals`

```
String a = "data";
String b = "data";                  // 同一个对象(常量池)
String c = new String("data");      // 一个新对象

a == b          // true  — 纯属巧合!
a == c          // false — 不同的对象
a.equals(c)     // true  — 内容相等
```

在 Java 中比较字符串**一律使用 `equals`**。`==` 比较的是**引用**（第 1 周）。

在 Go 中 `==` 比较的是**内容**：`a == c` → `true`

# 操作：Java 和 Go

| 操作 | Java | Go | 步数 |
|---|---|---|---|
| 长度 | `s.length()` | `len(s)`（字节） | O(1) |
| 第 i 个字符 | `s.charAt(i)` | `s[i]`（字节） | O(1) |
| 子串 | `s.substring(a, b)` | `s[a:b]` | O(b−a) / O(1)* |
| 拼接 | `s + t`、`s.concat(t)` | `s + t` | O(n + m) |
| 相等 | `s.equals(t)` | `s == t` | O(n) |
| 比较 | `s.compareTo(t)` | `s < t`、`strings.Compare` | O(n) |
| 查找 | `s.indexOf(t)` | `strings.Index(s, t)` | O(n · m)** |
| 分割 | `s.split(",")` | `strings.Split(s, ",")` | O(n) |
| 替换 | `s.replace(a, b)` | `strings.ReplaceAll(s, a, b)` | O(n) |
| 大小写转换 | `toLowerCase()`、`toUpperCase()` | `strings.ToLower`、`ToUpper` | O(n) |

* Go 中取子串不复制，而是引用共享的字节。 ** 类库中也使用更聪明的算法。

# 经典问题：回文

**回文**：正着读和倒着读都一样 – `"radar"`、`"上海自来水来自海上"`

伪代码

```
Function Palindrome(S, n)
  i ← 1 ; j ← n
  While i < j and S[i] = S[j]
    i ← i + 1
    j ← j − 1
  End while
  Return i ≥ j
End function
```

两个下标相向移动 – **O(n)**,O(1) 空间。

Go – 支持 Unicode

```go
func isPalindrome(s string) bool {
    r := []rune(s)
    for i, j := 0, len(r)-1; i < j;
            i, j = i+1, j-1 {
        if r[i] != r[j] {
            return false
        }
    }
    return true
}
```

如果在 Go 中逐字节检查，这段中文就**不是**回文了！

# 模式匹配

# 模式匹配问题

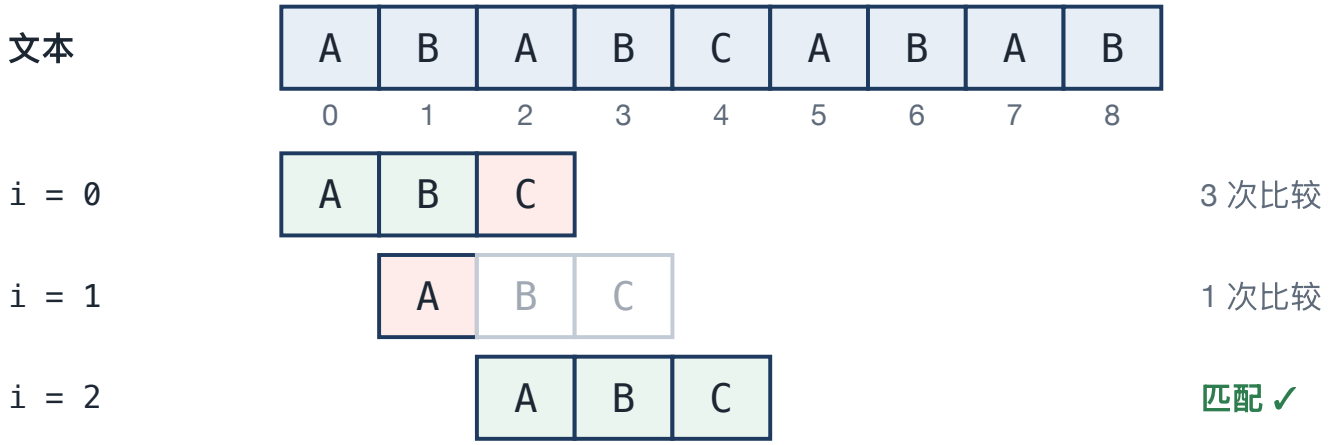
**已知：**长度为 **n** 的**文本**（S）和长度为 **m** 的**模式**（P），$m \le n$。
**任务：**P 在 S 中第一次出现在哪里？

应用：文本编辑器的“查找”功能、`grep`、病毒扫描、**DNA 序列**分析、抄袭检测

**朴素（暴力）算法：**

- 尝试模式的每一个可能的起始位置：$i = 1, 2, \ldots, n - m + 1$
- 在每个位置逐个字符地比较模式和文本
- 一旦不匹配，就移到下一个位置

# 朴素模式匹配 – 示例




总共 **7** 次字符比较

# 朴素模式匹配 – 算法

伪代码

```
Function NaiveSearch(S, n, P, m)
  For i ← 1 to n − m + 1
    j ← 1
    While j ≤ m and S[i + j − 1] = P[j]
      j ← j + 1
    End while
    If j > m then Return i
  End for
  Return 0
End function
```

Java

```java
static int naiveSearch(String s, String p) {
    int n = s.length(), m = p.length();
    for (int i = 0; i <= n − m; i++) {
        int j = 0;
        while (j < m &&
               s.charAt(i + j) == p.charAt(j))
            j++;
        if (j == m) return i;
    }
    return −1;
}
```

# 朴素算法的效率

- **最好情况：**模式在开头 → O(m)
- **最坏情况：**在每个位置都几乎匹配到最后
  S = `aaaa…aab` (n = 10 001),P = `aa…ab` (m = 101)
  → **1 000 001** 次比较 ≈ n · m → **O(n · m)**
- 在自然语言文本中通常在前 1–2 个字符就发现不匹配 → 实际上很快

## 更高效的算法（仅作介绍）

- **KMP(Knuth–Morris–Pratt):**O(n + m) – 利用模式本身的结构，文本指针从不回退
- **Boyer–Moore：**从右往左比较，可以大步跳跃 – 实际中非常快
- **Rabin–Karp：**比较哈希值（第 11 周）

# 小结

- 字符串是**字符序列**，顺序存储，直接访问
- 编码：**Unicode** 码点 → **UTF-8**（Go、Web）或 **UTF-16**(Java)
- “长度”可以指字节数、UTF-16 单元数或码点数 – Go 中 `len(s)` 给出的是字节数！
- Java 和 Go 的字符串是**不可变**的 → 在循环中使用 `StringBuilder` / `strings.Builder`
- Java 中要用 `equals`，不要用 `==`！
- **朴素模式匹配：**最坏情况 O(n · m)；更好的算法：KMP、Boyer–Moore

**下周：**排序 I – 冒泡排序、选择排序和插入排序。

# 复习题

1. Go 中 `len("中国")` 的值是多少？Java 中 `"中国".length()` 呢？
2. 判断对错：在 Java 中调用 `s.toUpperCase()` 之后，`s` 就变成大写了。
3. 为什么下面的代码是 $O(n^2)$ 的？`for (...) s += "x";`
4. `"data"` 和 `"Data"` 哪个的字典序更小？为什么？
5. 对 `S = "AAAAA"` 和 `P = "AAB"`，朴素算法进行多少次字符比较？
6. Java 输出什么：`new String("go") == "go"`？

# 实践课

# ✎ 示例程序

| 程序 | Java | Go | 演示内容 |
| --- | --- | --- | --- |
| 基础 | `StringBasics.java` | `w04/stringbasics` | 各种操作、`==` 与 `equals`、比较 |
| 编码 | `Encoding.java` | `w04/encoding` | 字节、`char`、码点、UTF-8 字节 |
| Builder | `Builder.java` | `w04/builder` | `+=` 与 `StringBuilder` 的计时 |
| 模式匹配 | `PatternSearch.java` | `w04/patternsearch` | 朴素查找、所有出现位置、最坏情况 |

# ✎ 练习

1. **回文：**判断一个句子是否是回文（忽略大小写和标点）！（`"A man, a plan, a canal: Panama"` ✓）
2. **单词计数：**如果单词之间用一个或多个空格分隔，文本中有多少个单词?
3. **字母频率：**用 26 个元素的数组统计英文字母出现的次数!
4. **凯撒密码：**把字母移动 **k** 位（`"Hello" → "Khoor"`，k = 3），再解密!
5. **变位词：**两个单词是否由相同的字母组成?（`listen` – `silent`）
6. **压缩：**`"aaabbcdddd" → "a3b2c1d4"`
7. **单词逆序：**`"I love data structures" → "structures data love I"`

参考答案：`Exercises04.java`、`w04/exercises`

# ✎ 课后作业

1. **不使用库函数**实现字典序比较（返回 `-1`、`0`、`1`）！
2. 编写函数，返回模式**所有**出现位置的起始下标 – 包括重叠的出现！（在 `"aaaa"` 中找 `"aa"`:0、1、2）
3. 编写程序，输出文本中**出现次数最多**的单词！（提示：Java `HashMap<String,Integer>`,Go `map[string]int`。）
4. 用 Go 编写函数，把字符串逆序，并且让汉字和表情符号也能正确显示！