Lecture8 正则表达式

1. 正则表达式介绍

- 正则表达式（或简称 regex 或 RE）是一个非常有用的工具，用于描述匹配文本的搜索模式
- Regex 只不过是定义搜索模式的一些字符序列
- Regex 用于解析、过滤、验证和从大型文本中提取有意义的信息，比如从其他程序生成的日志和输出

什么问题需要正则表达式

- 在搜索一些文本时，有时我们不知道文本的值，我们只知道文本的一些规则或模式
    - 例如，在日志消息中搜索 MAC 地址
    - 在 web 服务器访问日志中
    - 搜索 IP 地址搜索 10 位移动电话号码，该号码前可能有 0 或 +2 数字国家代码
- 我们试图提取的文本的长度是未知的，例如，在 csv 文件中搜索以 http:// 或 https:// 开头的 url
- 我们需要用变量类型和长度的分隔符分割给定的文本并生成标记
- 我们需要提取两个或多个搜索模式之间的文本
- 我们需要验证各种形式的用户输入，例如银行帐号、密码、用户名、信用卡信息、电话号码、出生日期，等等。
- 我们只想捕获一行中所有重复的单词，将输入文本转换为特定的预定义格式，例如每三个数字后插入一个逗号或只删除括号内的逗号
- 执行全局搜索替换，同时跳过所有转义字符

模式匹配 Pattern Matching

模式匹配问题：一个给定的字符串是一个给定的字符串集合的元素吗？

氨基酸问题

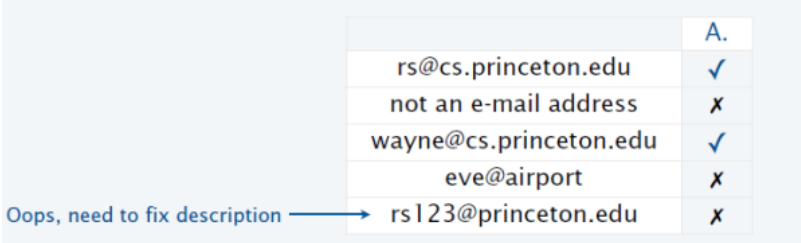
- 一个氨基酸可能由如下的字符组成 CAVLIMCRKHDENQSTYFWP
- 一个蛋白质是由一系列氨基酸组成的字符串

一种叫做 C_2H_2 zinc finger domain signature 类型的氨基酸的构成结构是

- C — 2/3/4 个氨基酸 —
- C — 3 个氨基酸 —
- L/I/V/M/F/Y/W/C/X — 8 个氨基酸 —
- H — 3/4/5 个氨基酸 — H

e-mail 地址问题

一个电子邮箱地址是

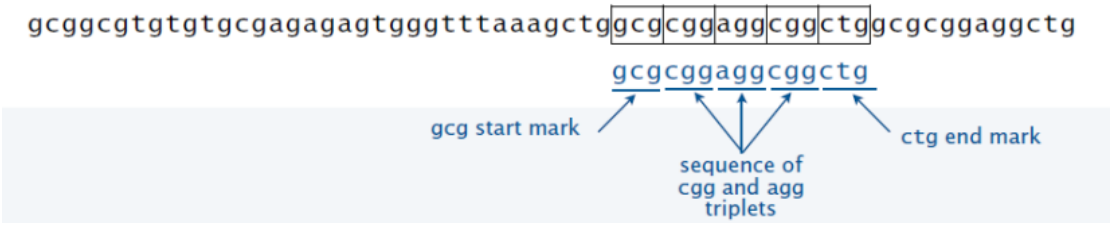


- 一系列字母 —
- `@` —
- 非空的小写字母序列 — `.` —
- `edu` / `com`（为了简洁，其他部分省略）

核酸问题

- 一个核酸可能由如下的字符组成 a, c, t, g
- 基因是一组核酸

一种叫做 Fragile X Syndrome 类型的基因的构成结构会存在下面一种结构



- gcg —
- 任何数量的 cgg/agg —
- ctg

正则表达式的定义

正则表达式 (RE) 是用于指定一组字符串（一种正式语言）的表示法

RE 可以是

- 一个空集合
- 一个空字符串
- 一个单一的字符或者一个通配符符号
- 一个用圆括号括起来的 RE
- 两个或多个正则表达式的 concatenation
- 两个或多个正则表达式的 union
- 两个或多个正则表达式的 closure

operation	*example RE*	*matches* (IN *the set*)	*does not match* (NOT *in the set*)
concatenation	aabaab	aabaab	*every other string*
wildcard	.u.u.u.	cumulus jugulum	succubus tumultuous
union	aa | baab	aa baab	*every other string*
closure	ab*a	aa abbba	ab ababa
parentheses	a(a|b)aab	aaaab abaab	*every other string*
	(ab)*a	a abababababa	aa abbba

2. 正则表达式的符号

标准正则表达式和元字符的构造

符号	意义	示例
`.`	匹配除了换行符以外的任何字符	匹配 #,@,A,f,5 等
`*`	匹配它前面的字符出现**零次或多次**	m* 匹配出现零次或多次 m
`+`	匹配它前面的字符出现**一次或多次**	m* 匹配出现一次或多次 m
`?`	匹配它前面的字符出现**零次或一次**	nm? 匹配 n 或者 nm
`|`	匹配由它分割的左右两个字符中的一个	m|n|p 匹配 m 或 n 或 p
`^`	是一个锚点，表示字符串的起始	^m 只匹配 字符串第一个字符是 m 的 不要在正则表达式中间使用 ^
`$`	是一个锚点，表示字符串的最后	m$ 表示 m 必须出现在字符串的最后
`(…)`	这是为了对可用于捕获特定子字符串或设置优先级的部分文本进行分组	m(ab)*t 匹配 m 后接零个或多个 ab，后接 t
`{min, max}`	一个量词范围，用于在最小和最大数字之间匹配前面的元素	mp{2, 4} 匹配 m 后接 2-4 个 p
`[...]`	这被称为字符类	[A-Z] 匹配任何大写英文字母
`\d`	匹配任何数字	\d 匹配任何 0-9 的数字
`\s`	匹配任何空格，包括制表符、空格或换行符	\s 匹配 [\t\n]
`\w`	匹配表示所有字母数字字符或下划线的任何 word characters	\w = [a-zA-z0-9_]
`\b`	匹配字母、数字和下划线被认为是一种 word characters \b 表示 word 的边界，也就是一个 word 前面和后面的界限	\bjava\b 匹配字符串 java 它不会匹配 javascript，因为后面的 \b 断言了 java 是一个单词

正则表达式示例

氨基酸问题

```
C.{2,4}C.{3}[LIVMFYWCX].{8}H.{3,5}H
```

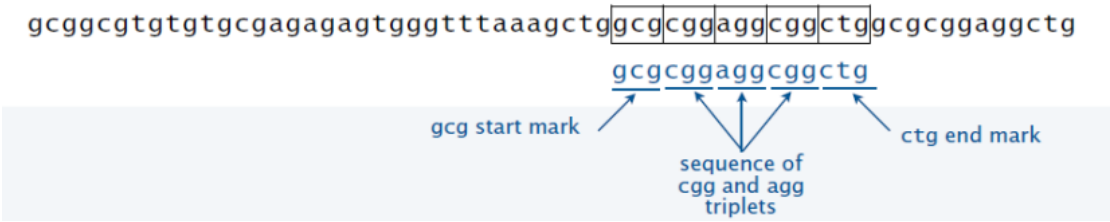
e-mail 地址问题

```
[a-z]+@([a-z]+\.)+(edu|com)
```

核酸问题

- 一个核酸可能由如下的字符组成 a, c, t, g
- 基因是一组核酸

一种叫做 Fragile X Syndrome 类型的基因的构成结构会存在下面一种结构



- gcg —
- 任何数量的 cgg/agg —
- ctg

`gcg(cgg|agg)*ctg`

3. Java 中的正则表达式

Java 的 `String` 类包含了一些与 RE 相关的方法

- RE 搜索和替换
- RE 分隔解析

public class String	
...	
String replaceAll(String re, String to)	*replace all occurrences of substrings matching RE with to*
String[] split(String re)	*split the string around matches of the given RE*
...	

Replace each sequence of at least one whitespace character with a single space.

```
String s = StdIn.readAll();
s = s.replaceAll("\\s+", " ");
```

Create an array of the words in StdIn (basis for StdIn.readAllStrings() method)

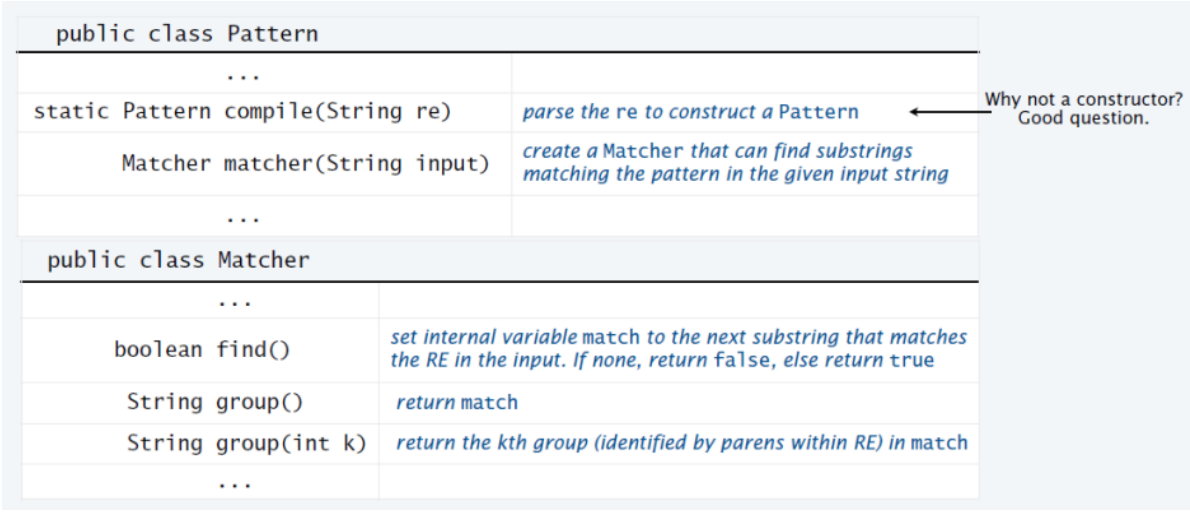
```
String s = StdIn.readAll();
String[] words = s.split("\\s+");
```

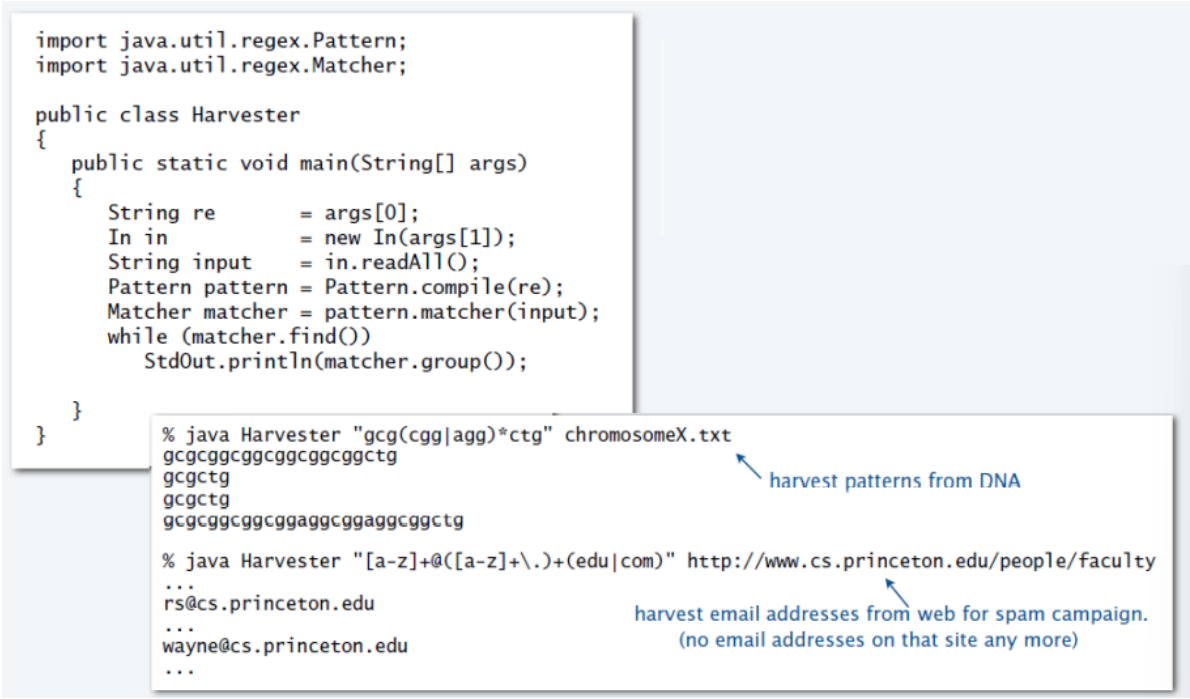
Java String API

Method Signature	Purpose
boolean matches(String regex)	Matches the given regular expression against the string that the method is invoked on and returns true/false, indicating whether the match is successful (true) or not (false).
String `replaceAll`(String regex, String replacement)	Replaces each substring of the subject string that matches the given regular expression with the replacement string and returns the new string with the replaced content.
String `replaceFirst`(String regex, String replacement)	This method does the same as the previous one with the exception that it replaces only the first substring of the subject string that matches the given regular expression with the replacement string and returns the new string with the replaced content.
`String[]` split(String regex)	Splits the subject string using the given regular expression into an array of substrings (example given ahead).
`String[]` split(String regex, int limit)	This overloaded method does the same as the previous one but there is an additional second parameter. The `limit` parameter controls the number of times regular expressions are applied for splitting.

Pattern 和 Matcher类

Java 的 `Pattern` 和 `Matcher` 类提供了很好的实现





`MatchResult` 是一个接口，表示所有 match 操作的结果，这个接口被 `Match` 类实现

Method Name	Description
int start()	Returns the start index of the match in the input
int start(int group)	Returns the start index of the specified capturing group
int end()	Returns the offset after the last character matched
int end(int group)	Returns the offset after the last character of the subsequence captured by the given group during this match
String group()	Returns the input substring matched by the previous match
String group(int group)	Returns the input subsequence captured by the given group during the previous match operation
int groupCount()	Returns the number of capturing groups in this match result's pattern

使用 `Pattern.compile().asPredicate()` 方法可以从一个编译后的正则表达式中获得一个 `Predicate` 类

这个对象可以被 stream 使用

```java
public static void main(String[] args){
    final String input = "value1||value2||value3";
    final Pattern p = Pattern.compile(Pattern.quote("||"));

    Array.stream(p.split(input)) // 或者使用 p.splitAsStream(in)
        .forEach(System.out::println)
}
```

Scanner 类

- Scanner 是一个实用程序类，用于解析输入文本并将输入分解为各种类型的标记，例如 boolean、int、float、double、long 等
- 它使用基于正则表达式的分隔符生成各种类型的标记，默认的分隔符是一个空格
- 使用 Scanner API，我们可以生成所有原始类型的标记，除了字符串标记
- String、Pattern 和 Matcher 类能够解析输入并只生成字符串类型的标记，但是 Scanner 类对于从输入源检查和生成不同类型的标记非常有用
- Scanner 实例可以使用 File、Inputstream、Path、Readable、ReadableBytechannel 和字符串参数来构造

Method Signature	**Purpose**
Scanner useDelimiter(String pattern)	Sets this scanner's delimiter regex pattern to a String regex argument.
Scanner useDelimiter(Pattern pattern)	This method is almost the same as the previous one but gets a `Pattern` as an argument instead of a `String`. This means that we can pass a regular expression already compiled. If we are forced to use the version with the `String` argument, the scanner would compile the string to a `Pattern` object even if we have already executed that compilation in other parts of the code. We will discuss the `Pattern` and `Matcher` class in the next chapter.
Pattern delimiter()	Returns the pattern being used by this scanner to match delimiters.
MatchResult match()	Returns the match result of the latest scan operation performed by this scanner.

Method Signature	Purpose
boolean hasNext(String pattern)	Returns `true` if the next token matches the pattern constructed from the specified string.
boolean hasNext(Pattern pattern)	This method is almost the same as the previous one but gets `Pattern` as an argument instead of `String`.
String next(String pattern)	Returns the next token if it matches the pattern constructed from the specified string.
String next(Pattern pattern)	This method is almost the same as the previous one but gets `Pattern` as an argument instead of `String`.
String findInLine(String pattern)	Attempts to find the next occurrence of a pattern constructed from the specified string, ignoring delimiters.

Method Signature	Purpose
String findInLine(Pattern pattern)	This method is almost the same as the previous one but gets `Pattern` as an argument instead of `String`.
Scanner skip(String pattern)	Skips the input that matches a pattern constructed from the specified string, ignoring delimiters.
Scanner skip(Pattern pattern)	This method is almost the same as the previous one but gets `Pattern` as an argument instead of `String`.
String findWithinHorizon(String pattern, int horizon)	Attempts to find the next occurrence of a pattern constructed from the specified string, ignoring delimiters.
String findWithinHorizon(Pattern pattern, int horizon)	This method is almost the same as the previous one but gets `Pattern` as an argument instead of `String`.