信息科学系
低年级讨论班

（1）

裘宗燕

北京大学数学学院信息科学系

2010年3月～6月

讨论班安排（设想）

- 本讨论班将以计算机程序和软件技术、方法和理论为学习讨论的主题，欢迎对这方面有兴趣的同学参加
- 课上讨论计划分为三个阶段：
  1. 我和其他同学讲一些文章（5周左右）
  2. 参加讨论班学习的同学讲一些文章（5－8周）
  3. 参加本讨论班的同学讲自己的想法和工作（3－5周）
- 要取得本讨论班的学分，需要：
  1. 到讨论班现场积极参加讨论
  2. 做至少一次读文章的报告
  3. 课下做一点相关研究/开发/整理工作，做一次报告
- 讨论班网页（有些说明，有中文样章）

  http://www.math.pku.edu.cn/teachers/qiuzy/seminar_A/

讨论班安排（设想）

- 讨论班材料：

  《代码之美》，原书：**“Beautiful Code”，O'Reilly**出版。中译本：机械工业出版社

  是一本很少见的讨论计算机软件技术的文集，三十多位作者描述了自己在软件各领域做的有趣工作。内容广泛，包括：

  - 良好程序设计的思想、技术和方法
  - 通用的系统设计和实现技术
  - 并发和分布式程序的有趣技术
  - 如何支持程序的变化和扩充（适应性设计）
  - 模块接口，用户界面，调试等等

- 不同文章在内容、风格、深度、广度、层次等各方面分差别也很大，反映了计算机软件工作的丰富多彩

正则表达式匹配

《代码之美》第1章

问题和概念

- 计算机处理的一类重要数据是文本，如文字（文章书籍），网页，一大批整数，DNA序列，程序，存储在计算机里二进制编码，等等
  - 也可以认为，计算机处理的基础数据都是文本
- 文本处理的基础是比较，最简单的比较是文本间的直接比较，如
  - 相同和不同
  - 按字典序比较大小
  - 是否出现（子串）及出现位置
  - 等等
- 可能根据比较结果做进一步操作，如：
  - 确定是否找到了所需数据
  - 文本代换，变形等
  - 统计，分析等

问题和概念

- 简单文本比较的抽象是“字符串比较”，有关概念
  - 字符：被考虑的基本符号（注意，不仅可以是字母数字）
  - 字符集：被考虑的所有字符的集合
  - 字符串：字符集中字符的有穷序列

  这方面知识是数据结构课程的一个内容
- 问题：有时希望找一类字符串，而不是一个字符串。如：
  - C程序里所有以 # 开头的代码行（预处理命令）
  - 名字里出现“昊”字的北大学生
  - C程序里所有字母或“_”开头的字母数字序列
  - 网页里所有“<A”开头“>”结尾的段（是链接）
  - DNA序列中具有某种形式的子序列（基因分析的基础）
  - 程序里具有某些特征的片段（扫描病毒）

问题和概念

- 新问题：要描述一类字符串，需要有一种描述方式
  - 有关字符串描述有丰富的理论成果，完全是数学理论，是计算机科学技术最重要的基础理论，“形式语言理论”，“自动机理论”
  - 下面讨论的是最简单的一类描述：正则表达式
- 正则表达式（**regular expression**）是数学家和逻辑学家 **Kleene** 在 **1950** 年代提出的一种描述字符串集合的记法形式，其定义是：

$$a \in \mathbb{A}$$ $\mathbb{A}$ 是我们考虑的字符集

$$\alpha \in \mathbb{A} \times \ldots \times \mathbb{A}$$ α 是字符的有限序列

$$\alpha ::= a$$ 一个字符是正则表达式

$$\alpha ::= \alpha\alpha$$ 两个正则表达式的连接

$$\alpha ::= \alpha \mid \alpha$$ 两个正则表达式的“或”

$$\alpha ::= (\alpha)$$ 加括号

$$\alpha ::= \alpha*$$ **Kleene** 星号

问题和概念

- 正则表达式匹配的字符串：

a	只与 a匹配
$\alpha_1\alpha_2$	字符串前一段与α_1匹配，后一段与α_2匹配
$\alpha_1 \mid \alpha_2$	字符串与α_1 或者α_2匹配
(α)	字符串与 α 匹配
$\alpha*$	字符串可以分为若干段，每段都与 α 匹配

- 刻画了一类字符串的描述称为一个字符串模式。正则表达式是一种描述字符串模式的“语言”
- 理论结果：正则表达式模式匹配问题可计算。从长 **m** 的正则表达式构造识别器（有穷自动机，用于做匹配的模型）的时间复杂性是 **$O(2^m)$**，构造好的识别器匹配长 **n** 的字符串的时间复杂性为 **O(n)**
- 本章讨论的问题是怎样实现一个简单高效的正则表达式匹配程序

正则表达式匹配器

- 正则表达式和匹配器在实际应用中有重要作用

  许多语言将正则表达式作为语言的基本功能，如 **Perl**，**Python**，**Ruby**等。主要语言都提供了正则表达式功能库

- 实际语言或库中的正则表达式都是 **Kleene** 正则表达式的某种变形

  选择（设计）一种正则表达式描述形式

  实现一个匹配器

- 本章介绍的是 **Brian Kernighan** 和 **Rob Pike** 的工作，其基本目标是阐释正则表达式的思想，实现一个简短紧凑的匹配器

  常规语言或库的正则表达式功能可能更为丰富，代码可能长达数百行甚至数千行

  这里介绍的匹配器只有几十行代码

正则表达式设计

- 首先设计正则表达式形式。这里的形式比较简单，但也很强大：

字符	解释和意义
c	除特殊字符外的任意字符，只与同一字符匹配
.	句号，与任意单个字符匹配（通配符）
^	匹配字符串开始
$	匹配字符串结束
*	匹配前一字符的0个或多个出现

- 显然：这一语言不能表达所有正则表达式能够表示的字符串集合
- 但其功能足以表达实际中需要的许多匹配模式。作者说，根据经验，可以表示实际中可能用到的 95% 以上的模式

设计分析

字符	解释和意义
c	除特殊字符外的任意字符，只与同一字符匹配
.	句号，与任意单个字符匹配
^	匹配字符串开始
$	匹配字符串结束
*	匹配前一字符的0次或多次出现

- 模式语言的特点：
  1. 基本元素只能匹配单个字符或所有字符
  2. 只能描述一个特定字符或任意字符的0次或多次出现
  3. 没有“或者”，也没有括号组合
- 后两条限制了它的模式描述能力，也保证了匹配的高效率（简单的问题往往存在更为高效的算法，一般性问题更难解决）
- 作者将匹配器实现为几个递归定义的函数

匹配器实现

- **match**是匹配器主函数，检查 **text** 里是否存在与 **regexp** 匹配的子串
- **matchhere** 检查从 **text** 当前位置开始的一段是否与 **regexp** 匹配
- **matchstar** 处理星号，在 **text** 里考虑去掉前面与 **c** 匹配的一段后能否与 **regexp** 匹配
- 有匹配时返回 **1**；否则返回 **0**
- 如模式首字符为 **^** 则只考虑从头开始的匹配，否则考虑任意位置匹配

```
/* match: search for regexp anywhere in text */
int match(char *regexp, char *text) {
    if (regexp[0] == '^')  return matchhere(regexp+1, text);
    do {    /* must look even if string is empty */
        if (matchhere(regexp, text)) return 1;
    } while (*text++ != '\0');
    return 0;
}

/* matchhere: search for regexp at beginning of text */
int matchhere(char *regexp, char *text) {
    if (regexp[0] == '\0') return 1;
    if (regexp[1] == '*')
        return matchstar(regexp[0], regexp+2, text);
    if (regexp[0] == '$' && regexp[1] == '\0') return *text == '\0';
    if (*text!='\0' && (regexp[0]=='.' || regexp[0]==*text))
        return matchhere(regexp+1, text+1);
    return 0;
}

/* matchstar: search for c*regexp at beginning of text */
int matchstar(int c, char *regexp, char *text) {
    do {    /* a * matches zero or more instances */
        if (matchhere(regexp, text)) return 1;
    } while (*text != '\0' && (*text++ == c || c == '.'));
    return 0;
}
```

设计和实现分析

- 如果 **regexp** 里包含 **$**（只能作为最后一个字符），就要求匹配的双方同时结束。程序里的条件是

  if (regexp[0] == '$' && regexp[1] == '\0') return *text == '\0';

  中文版这段翻译错了

- 对函数 **matchhere** 的可能改进：
  - 把特殊情况 **'$'** 的处理移到前面。一般原则：先处理简单的特殊的情况，后处理复杂的情况
  - 本算法识别的是 **text** 里的一个最左最短匹配，如需要，可以加些代码，让匹配器返回被匹配段的完整信息（例如用一个 **struct** 返回被匹配段在 **text** 里的起始和结束位置。请考虑如何修改）
  - 有时人们希望找与指定模式的最长匹配（如文本处理中查找被代换的文本，通常是想找能与正则表达式匹配的最长子串）
  - 下面考虑这个问题

设计和实现分析

- 实现最左最长匹配：

```
int matchstar(int c, char *regexp, char *text) {
    char *t;
    for (t = text; *t != '\0' && (*t == c || c == '.'); t++)
        ;
    do { /* * matches zero or more */
        if (matchhere(regexp, t)) return 1;
    } while (t-- > text);
    return 0;
}
```

- **text** 是文本当前位置。先用 **t** 找到与 **c** 匹配的最右位置之后一个位置
- 从 **t** 开始看文本内容能不能与 **regexp** 匹配
  - 能匹配则成功
  - 如果不能就后退一个位置（用 **t--** 实现）再试
  - 试到 **t** 等于 **text** 还不匹配，就确定匹配失败

算法扩充建议

- 增加元字符。例如 **+** 表示一个或多个，**?** 表示**0**个或一个。还可以考虑允许在正则表达式和被匹配文本里出现元字符。如用 **C** 语言换意字符的设计，令 **\$** 表示 **'$'**，**\.** 表示 **'.'** 等等
- 把正则表达式处理分为编译阶段和匹配阶段，提高匹配效率
  - 有些应用这样做意义不大，如在编辑文本中找匹配。原因：模式较简单，通常一个模式只用一次或很少几次
  - 对有些应用意义很大。如垃圾邮件过滤，**DNA** 序列匹配
  - 书上的例子：在 **4M** 文本里找模式 **a.*a.*a.*a.a** 的所有匹配，直接匹配用 **20**秒，编译后匹配用半秒
- 增加字符组的概念，如 **[abc]** 与 **a, b, c** 都匹配，**[0-9]** 与任何数字匹配。还可以考虑采用 **Perl** 语言的设计，用 **\d** 表示任意数字字符，用 **\D** 表示任意非数字字符
- 文中还提出了一些实现技术问题，以及将这个程序搬到各种语言里去的问题，请自己阅读参考

总结

- 基本功能集合的选择很重要

  体现在这里就是正则表达式的设计

  选择能满足（基本需要的）最简单的东西

- 成功地应用了递归的思想和递归程序设计技术

  请考虑把这一程序修改为一个只用循环不用递归的程序

- 利用 **C** 语言里的指针功能，扫描数组（字符串）内容，得到了非常紧凑的代码。包括指针运算 **++i** 等

参考材料

- 程序设计实践，**Brian W. Kernighan, Rob Pike**，中文版：机械工业出版社，**2004**，**http://www.china-pub.com/664**
- 精通正则表达式:第**3**版，**Jeffrey E.F.Friedl**，中文版：电子工业出版社 **2009**，链接：**http://www.china-pub.com/47529**
- 自动机理论、语言和计算导论(原书第**3**版)，**(美)John E. Hopcroft, Rajeev Motwani, Jeffrey D. Ullman**，中文版：机械工业出版社，链接：**http://www.china-pub.com/209080**
- 计算理论导引，**Michael Sipser**，中文版：机械工业出版社 **2007**，链接：**http://www.china-pub.com/31072**

研究课题

1. 设有一批（如 **100** 个或 **10000** 个）正则表达式表示的匹配模式，如何实现高效匹配器
   - 可能做怎样的预处理？
   - 如何把一组模式组合为一个模式？
   - 可以考虑具体场景，如垃圾邮件过滤

   注意：对规模很大的问题，标准技术不适用
2. 正则表达式的设计。考查多种语言和库中的正则表达式设计，考虑某种特殊需要下的正则表达式设计和匹配器实现

这两个问题都有重要应用背景，可能实际应用很多。如：

- 垃圾邮件过滤
- 计算机系统的日志（**log**）文件分析
- **DNA** 序列分析