

分子生物计算 (Perl 语言编程)

天津医科大学
生物医学工程与技术学院

2016-2017 学年上学期 (秋)
2014 级生信班

第八章 遗传密码

伊现富 (Yi Xianfu)

天津医科大学 (TJMU)
生物医学工程与技术学院

2015 年 12 月



教学提纲

- 1 引言
- 2 散列
- 3 数据结构和算法
- 4 遗传密码
 - 密码子翻译成氨基酸
 - 遗传密码的冗余性
 - 使用散列表表示遗传密码
- 5 DNA 翻译成蛋白质
- 6 读取 FASTA 格式的 DNA
 - 序列格式
 - 读取 FASTA 文件的思路
 - 读取 FASTA 文件的子程序
 - 输出格式化的序列数据

- 读取 FASTA 文件的主程序
- DNA 翻译成蛋白质的主程序
- 7 阅读框
 - 简介
 - 翻译阅读框
- 8 回顾和总结
 - 总结
 - 思考题
- 9 知识拓展
 - Perl 里面的复杂数据结构
 - 数据共享
 - 宽格式和长格式的数据
 - 数据处理实例

教学提纲

1

引言

2

散列

3

数据结构和算法

4

遗传密码

- 密码子翻译成氨基酸
- 遗传密码的冗余性
- 使用散列表表示遗传密码

5

DNA 翻译成蛋白质

6

读取 FASTA 格式的 DNA

- 序列格式
- 读取 FASTA 文件的思路
- 读取 FASTA 文件的子程序
- 输出格式化的序列数据

- 读取 FASTA 文件的主程序
- DNA 翻译成蛋白质的主程序

7

阅读框

- 简介
- 翻译阅读框

8

回顾和总结

- 总结
- 思考题

9

知识拓展

- Perl 里面的复杂数据结构
- 数据共享
- 宽格式和长格式的数据
- 数据处理实例



已经学习

- 查找基序
- 模拟 DNA 突变
- 生成随机序列
- DNA 转录成 RNA

即将学习

- Perl 中的散列
- DNA 翻译成蛋白质
- 处理 FASTA 文件



已经学习

- 查找基序
- 模拟 DNA 突变
- 生成随机序列
- DNA 转录成 RNA

即将学习

- Perl 中的散列
- DNA 翻译成蛋白质
- 处理 FASTA 文件



教学提纲

1

引言

2

散列

3

数据结构和算法

4

遗传密码

- 密码子翻译成氨基酸
- 遗传密码的冗余性
- 使用散列表表示遗传密码

5

DNA 翻译成蛋白质

6

读取 FASTA 格式的 DNA

- 序列格式
- 读取 FASTA 文件的思路
- 读取 FASTA 文件的子程序
- 输出格式化的序列数据

- 读取 FASTA 文件的主程序
- DNA 翻译成蛋白质的主程序

7

阅读框

- 简介
- 翻译阅读框

8

回顾和总结

- 总结
- 思考题

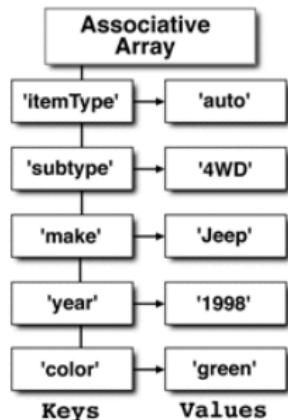
9

知识拓展

- Perl 里面的复杂数据结构
- 数据共享
- 宽格式和长格式的数据
- 数据处理实例



散列 | 简介



%Perez

Age	21
Name	"Jeff"
Eyes	"Br"
Temp	98.6

\$Perez{'Eyes'} = "Br";

%Kleiner

Age	21
Name	"Dave"
Eyes	"Blue"
Temp	99.0

\$Kleiner{'Temp'} = 99.0;



散列 | 初始化

```
1 %classification = (  
2     'dog',          'mammal',  
3     'robin',         'bird',  
4     'asp',           'reptile',  
5 );  
6  
7 # 更加清晰易懂  
8 %classification = (  
9     'dog'    => 'mammal',  
10    'robin'   => 'bird',  
11    'asp'     => 'reptile',  
12 );
```



- 胖箭头 (`=>`) : 对 Perl 而言, 只是逗号的另一种写法, 因此常被称为胖逗号
- 在任何需要逗号 (,) 的地方都可以用胖箭头 (`=>`) 代替, 对 Perl 而言没有区别
- 胖箭头左边的任何裸字 (一连串的字母、数字和下划线, 但不得以数字开头) 都会自动加上引号, 因此胖箭头左边的裸字不需要加引号
- 在作为散列键使用时, 如果花括号内只有裸字时, 两边的引号也可以省略
- 列表末尾有一个额外的逗号, 这种写法无伤大雅, 而且便于维护 (添加/删除)



散列 | 键值操作

```
1 # 给键赋值
2 $english_dictionary{'recreant'} = "One who
   calls out in surrender.";
3 # 散列 (hash) : %english_dictionary
4 # 键 (key) : 标量, 'recreant'
5 # 值 (value) : 标量, "One who calls out in
   surrender."
6
7 # 提取键对应的值
8 $english_dictionary{'recreant'};
9 $definition = $english_dictionary{'recreant'};
```



散列 | 常用函数 | keys & values

```
1 # 初始化散列
2 %hash = (a=>1, b=>2, c=>3);
3
4 # 获取键列表 (所有的键)
5 @keys = keys %hash;
6 # 获取值列表 (所有的值)
7 @values = values %hash;
8 # @keys和@values各自的顺序不定, 但两者总是一一对应
9
10 # 获取散列元素 (键值对) 的个数
11 $count = keys %hash;
12 $count = values %hash;
13 # 列表上下文
```



散列 | 常用函数 | each

```
1 # 逐项处理散列中的每一个元素
2 while ( ($key, $value) = each %hash ) {
3     print "$key => $value\n";
4 }
5 # each返回的键值对的顺序是乱的，但它与keys和values
6 # 返回的顺序相同，即散列的自然顺序
7
8 foreach my $key (sort keys %hash) {
9     my $value = $hash{$key};
10    print "$key => $value\n";
11 }
```



散列 | 常用函数 | 判断真假

```
1 # 真
2 if ($books{$someone}) {
3     print "$someone has at least one book
checked out.\n";
4 }
5
6 # 假
7 # 现在没有借阅图书
8 $books{"barney"} = 0
9 # 这是张新办的借书证，从未借阅过图书
10 $books{"pebbles"} = undef;
```



散列 | 常用函数 | delete

```
1 # 删除指定的键及其相对应的值  
2 delete $books{$person};  
3 # 撤消$person的借书证  
4 # delete之后, 键不会出现在散列之中  
5  
6 $books{$person} = undef;  
7 # 键依然存在于散列之中
```



散列 | 常用函数 | reverse

```
1 %hash = (a=>1, b=>2, c=>3);  
2  
3 # 只适用于散列值不重复的情况，否则会导致重复的键  
4 # “后发先至”：最后的键会覆盖之前的键  
5 %inverse_hash = reverse %hash;  
6 # %inverse_hash = (1=>a, 2=>b, 3=>c);
```



散列 | 常用函数 | 元素内插

```
1 # 可以将单一散列元素内插到双引号包裹起来的字符串中
2 foreach $person (sort keys %books) {
3     if ($books{$person}) {
4         print "$person has $books{$person} items
5             .\n";
6     }
7 }
8 # 注意：不支持内插整个散列 ("%books")
9 print "%books";
10 # 包含6个字符的字符串：%books
```



defined

- 判断某个字符串是 undef 而不是空字符串
- 如果是 undef, 返回假; 否则返回真

exists

- 检查散列中是否存在某个键, 和键对应的值无关
- 如果键存在, 返回真; 否则返回假



defined

- 判断某个字符串是 `undef` 而不是空字符串
- 如果是 `undef`, 返回假; 否则返回真

exists

- 检查散列中是否存在某个键, 和键对应的值无关
- 如果键存在, 返回真; 否则返回假



散列 | 常用函数 | defined vs. exists

```
1 #!/usr/bin/perl
2
3 use strict;
4 use warnings;
5 use utf8;
6
7 my %hash = (
8     a => 1,
9     b => 0,
10    c => "0",
11    d => "undef",
12    e => undef,
13 );
```



散列 | 常用函数 | defined vs. exists

```
1 if ( defined $hash{a} ) {
2     print "defined: $hash{a} (number) is TRUE!\n";
3 }
4 if ( defined $hash{b} ) {
5     print "defined: $hash{b} (number) is TRUE!\n";
6 }
7 if ( defined $hash{c} ) {
8     print "defined: $hash{c} (string) is TRUE!\n";
9 }
10 if ( defined $hash{d} ) {
11     print "defined: $hash{d} (string) is TRUE!\n";
12 }
13 unless ( defined $hash{e} ) {
14     print "defined: undef is FALSE!\n";
15 }
16 unless ( defined $hash{x} ) {
17     print "defined: key ?x? is FALSE!\n";
18 }
```



散列 | 常用函数 | defined vs. exists

```
1 if ( exists $hash{a} ) {  
2     print "exists: key a is TURE!\n";  
3 }  
4 if ( exists $hash{b} ) {  
5     print "exists: key b is TURE!\n";  
6 }  
7 if ( exists $hash{c} ) {  
8     print "exists: key c is TURE!\n";  
9 }  
10 if ( exists $hash{d} ) {  
11     print "exists: key d is TURE!\n";  
12 }  
13 if ( exists $hash{e} ) {  
14     print "exists: key e is TURE!\n";  
15 }  
16 unless ( exists $hash{x} ) {  
17     print "exists: key ?x? is FALSE!\n";  
18 }
```



散列 | 常用函数 | defined vs. exists

```
1 if ( $hash{a} ) {
2     print "value: $hash{a} (number) is TRUE!\n";
3 }
4 unless ( $hash{b} ) {
5     print "value: $hash{b} (number) is FALSE!\n";
6 }
7 unless ( $hash{c} ) {
8     print "value: $hash{c} (string) is FALSE!\n";
9 }
10 if ( $hash{d} ) {
11     print "value: $hash{d} (string) is TRUE!\n";
12 }
13 unless ( $hash{e} ) {
14     print "value: undef is FALSE!\n";
15 }
16 unless ( $hash{x} ) {
17     print "value: key ?x? is FALSE!\n";
18 }
```



三大类

- 标量变量 (scalar variable) : \$scalar
- 数组 (array) : @array
- 散列 (hash, associative array) : %hash

数组 vs. 散列

- 数组元素: \$array[0]
- 散列查找: \$hash{'key'}



三大类

- 标量变量 (scalar variable) : \$scalar
- 数组 (array) : @array
- 散列 (hash, associative array) : %hash

数组 vs. 散列

- 数组元素: \$array[0]
- 散列查找: \$hash{'key'}



教学提纲

1

引言

2

散列

3

数据结构和算法

4

遗传密码

- 密码子翻译成氨基酸
- 遗传密码的冗余性
- 使用散列表表示遗传密码

5

DNA 翻译成蛋白质

6

读取 FASTA 格式的 DNA

- 序列格式
- 读取 FASTA 文件的思路
- 读取 FASTA 文件的子程序
- 输出格式化的序列数据

- 读取 FASTA 文件的主程序
- DNA 翻译成蛋白质的主程序

7

阅读框

- 简介
- 翻译阅读框

8

回顾和总结

- 总结
- 思考题

9

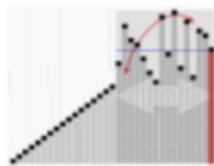
知识拓展

- Perl 里面的复杂数据结构
- 数据共享
- 宽格式和长格式的数据
- 数据处理实例



数据结构和算法 | 简介

$O(n^2)$



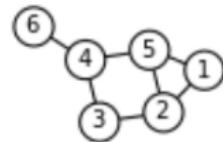
算法分析

12 → 99 → 37 → ⊗

→



计算几何



图论



算法

算法 (algorithm) 指定义良好的计算过程，它取一个或一组值作为输入，经过一系列定义好的计算过程，得到一个或一组输出。

在数学和计算机科学/算学之中，算法/演算法/算则法 (Algorithm) 为一个计算的具体步骤，常用于计算、数据处理和自动推理。精确而言，算法是一个表示为有限长列表的有效方法。算法应包含清晰定义的指令用于计算函数。

算法中的指令描述的是一个计算，当其运行时能从一个初始状态和初始输入（可能为空）开始，经过一系列有限而清晰定义的状态最终产生输出并停止于一个终态。

不同的算法通常需要不同的数据结构。



特征

高德纳在他的著作《计算机程序设计艺术》里对算法的特征归纳：

- 输入：一个算法必须有零个或以上输入量。
- 输出：一个算法应有一个或以上输出量，输出量是算法计算的结果。
- 明确性：算法的描述必须无歧义，以保证算法的实际执行结果是精确地匹配要求或期望，通常要求实际运行结果是确定的。
- 有限性：依据图灵的定义，一个算法是能够被任何图灵完备系统模拟的一串运算，而图灵机只有有限个状态、有限个输入符号和有限个转移函数（指令）。而一些定义更规定算法必须在有限个步骤内完成任务。
- 有效性：又称可行性。能够实现，算法中描述的操作都是可以通过已经实现的基本运算执行有限次来实现。



基本要素

算法的核心是**创建问题抽象的模型和明确求解目标**，之后可以根据具体的问题选择不同的模式和方法完成算法的设计。

形式化算法

算法是计算机处理信息的本质，因为**计算机程序本质上是一个算法来告诉计算机确切的步骤来执行一个指定的任务**。一般地，当算法在处理信息时，会从输入设备或数据的存储地址读取数据，把结果写入输出设备或某个存储地址供以后再调用。

常用实现方法

- 递归方法与迭代方法
- 顺序计算、并行计算和分布式计算
- 确定性算法和非确定性算法
- 精确求解和近似求解

基本要素

算法的核心是**创建问题抽象的模型和明确求解目标**，之后可以根据具体的问题选择不同的模式和方法完成算法的设计。

形式化算法

算法是计算机处理信息的本质，因为**计算机程序本质上是一个算法来告诉计算机确切的步骤来执行一个指定的任务**。一般地，当算法在处理信息时，会从输入设备或数据的存储地址读取数据，把结果写入输出设备或某个存储地址供以后再调用。

常用实现方法

- 递归方法与迭代方法
- 顺序计算、并行计算和分布式计算
- 确定性算法和非确定性算法
- 精确求解和近似求解

基本要素

算法的核心是**创建问题抽象的模型和明确求解目标**，之后可以根据具体的问题选择不同的模式和方法完成算法的设计。

形式化算法

算法是计算机处理信息的本质，因为**计算机程序本质上是一个算法来告诉计算机确切的步骤来执行一个指定的任务**。一般地，当算法在处理信息时，会从输入设备或数据的存储地址读取数据，把结果写入输出设备或某个存储地址供以后再调用。

常用实现方法

- 递归方法与迭代方法
- 顺序计算、并行计算和分布式计算
- 确定性算法和非确定性算法
- 精确求解和近似求解

常用设计模式

- 完全遍历法和不完全遍历法：把解空间的所有元素完全遍历一遍，逐个检测元素是否是我们的解。
- 分治法：把一个问题分区成互相独立的多个部分分别求解的思路，便于进行并行计算。
- 动态规划法：当问题的整体最优解就是由局部最优解组成的时候，经常采用的一种方法。
- 贪婪算法：常见的近似求解思路。当问题的整体最优解不是（或无法证明是）由局部最优解组成，且对解的最优化没有要求的时候，可以采用的一种方法。
- 线性规划法：线性规划是最优化问题中的一个重要领域。很多最优化问题算法都可以分解为线性规划子问题，然后逐一求解。
- 简并法：把一个问题通过逻辑或数学推理，简化成与之等价或者近似的、相对简单的模型，进而求解的方法。

数据结构

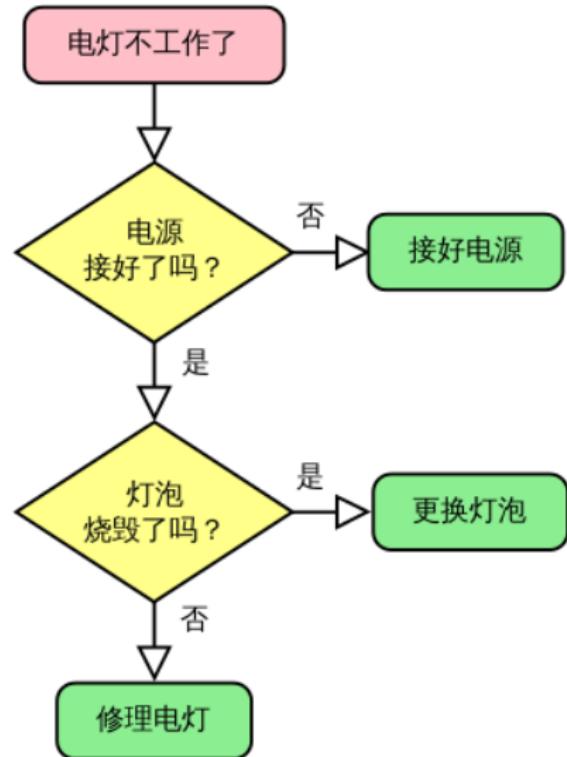
数据结构 (data structure) 是计算机中存储、组织数据的方式。

数据结构意味着接口或封装：一个数据结构可被视为两个函数之间的接口，或者是由数据类型联合组成的存储内容的访问方法封装。

数据结构可透过程序语言所提供的数据类型、引用及其他操作加以实现。一个设计良好的数据结构，应该在尽可能使用较少的时间与空间资源的前提下，支持各种程序运行。

不同种类的数据结构适合不同种类的应用，部分数据结构甚至是为了解决特定问题而设计出来的。**正确的数据结构选择可以提高算法的效率。**在计算机程序设计的过程里，选择适当的数据结构是一项重要工作。许多大型系统的编写经验显示，程序设计的困难程度与最终成果的质量与表现，取决于是否选择了最适合的数据结构。





常见数据结构

- 数组 (Array)
- 堆栈 (Stack)
- 队列 (Queue)
- 链表 (Linked List)
- 树 (Tree)
- 图 (Graph)
- 堆 (Heap)
- 散列表 (Hash)



生物学问题

特定环境条件下、特性类型细胞中，人类基因组中的每一个基因是否表达？

实验方法与数据处理

- 基因芯片、RNA 测序……
- 芯片：探针 \rightarrow 基因；亮度 \rightarrow 表达值
- 测序：reads \rightarrow 基因；reads 数 \rightarrow 表达值
- 其他：质量控制，标准化，基因注释，……

最终数据

- 所有基因（30000）：基因名
- 表达基因（8000）的表达水平：具体数值
- 未表达基因（22000）的表达值：0

生物学问题

特定环境条件下、特性类型细胞中，人类基因组中的每一个基因是否表达？

实验方法与数据处理

- 基因芯片、RNA 测序……
- 芯片：探针 \Rightarrow 基因；亮度 \Rightarrow 表达值
- 测序：reads \Rightarrow 基因；reads 数 \Rightarrow 表达值
- 其他：质量控制，标准化，基因注释，……

最终数据

- 所有基因（30000）：基因名
- 表达基因（8000）的表达水平：具体数值
- 未表达基因（22000）的表达值：0

生物学问题

特定环境条件下、特性类型细胞中，人类基因组中的每一个基因是否表达？

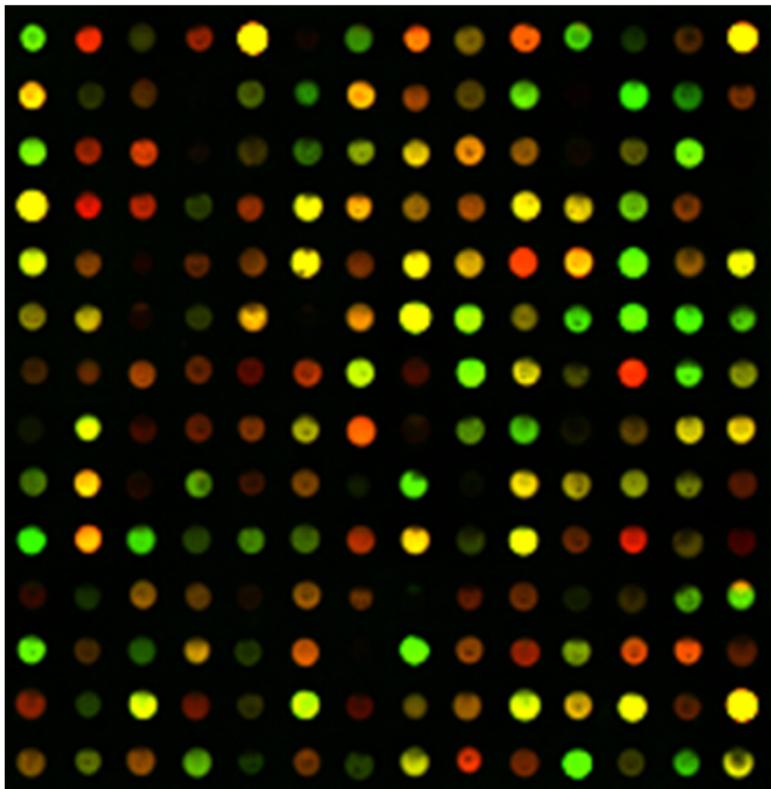
实验方法与数据处理

- 基因芯片、RNA 测序……
- 芯片：探针 \Rightarrow 基因；亮度 \Rightarrow 表达值
- 测序：reads \Rightarrow 基因；reads 数 \Rightarrow 表达值
- 其他：质量控制，标准化，基因注释，……

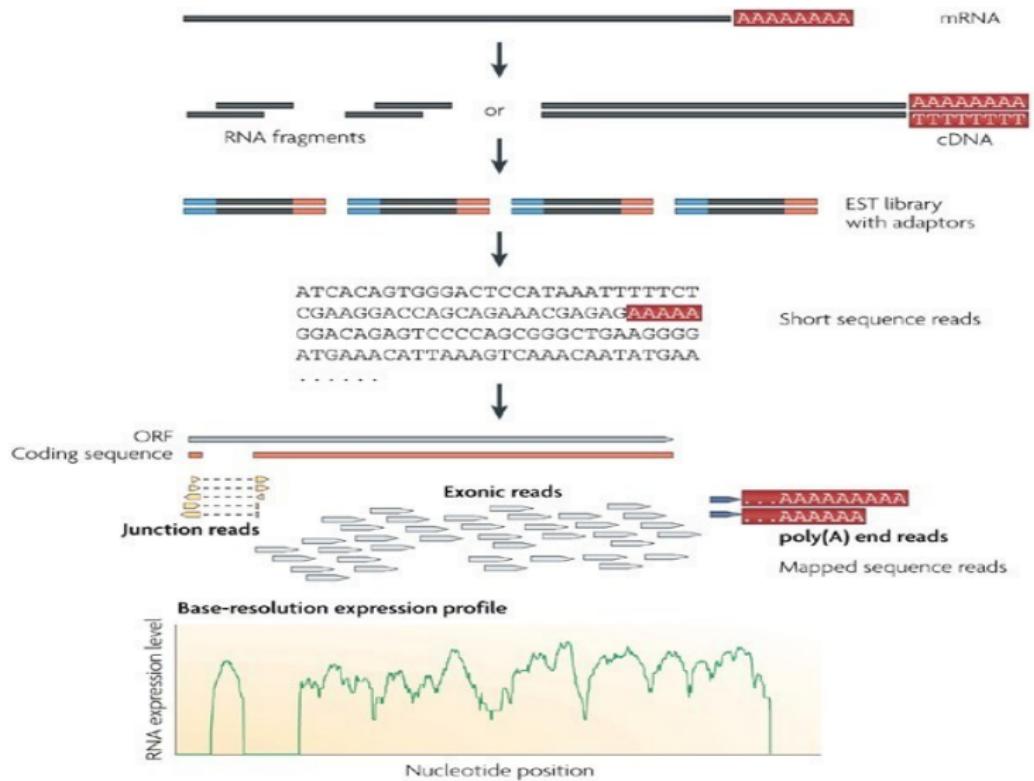
最终数据

- 所有基因（30000）：基因名
- 表达基因（8000）的表达水平：具体数值
- 未表达基因（22000）的表达值：0

数据结构和算法 | 基因表达 | 芯片



数据结构和算法 | 基因表达 | RNA-Seq



具体实现

只在数组中存储表达基因（8000）的基因名，丢弃未表达基因的基因名

可能的问题

- 查找表达基因：平均比较 4000 次
- 查找未表达基因：必须比较 8000 次
- 查找未研究的基因：未表达？未研究？



具体实现

只在数组中存储表达基因（8000）的基因名，丢弃未表达基因的基因名

可能的问题

- 查找表达基因：平均比较 4000 次
- 查找未表达基因：必须比较 8000 次
- 查找未研究的基因：未表达？未研究？



改进方法

- 把 30000 个基因全部存储到数组中
- 把表达值附加在基因名的后面

存在的问题

- 每次查询：平均比较 15000 次
- 分割基因名和表达值



改进方法

- 把 30000 个基因全部存储到数组中
- 把表达值附加在基因名的后面

存在的问题

- 每次查询：平均比较 15000 次
- 分割基因名和表达值



具体实现

- 把所有基因名按照字母顺序排序后存储到数组中
- 使用折半查找的算法进行查找

优缺点

- 优点：只需大约 15 次循环比较即可
- 缺点：需要首先对列表进行排序
- 缺点：添加新元素时稍显繁琐



具体实现

- 把所有基因名按照字母顺序排序后存储到数组中
- 使用折半查找的算法进行查找

优缺点

- 优点：只需大约 15 次循环比较即可
- 缺点：需要首先对列表进行排序
- 缺点：添加新元素时稍显繁琐



具体实现

- 基因名 \Rightarrow 散列的键
- 表达值 \Rightarrow 散列的值

优缺点

- 优点：查找速度通常比折半查找快
- 优点：可以明确知道要查找的基因是否在数据集中存在
- 优点：添加或删除元素时非常方便
- 缺点：其中的元素没有经过排序



具体实现

- 基因名 \Rightarrow 散列的键
- 表达值 \Rightarrow 散列的值

优缺点

- 优点：查找速度通常比折半查找快
- 优点：可以明确知道要查找的基因是否在数据集中存在
- 优点：添加或删除元素时非常方便
- 缺点：其中的元素没有经过排序



未排序的数组

- 需要遍历数组，速度很慢
- 必须要把基因名和表达值分割开来

排序数组和折半查找

- 查找速度大幅提升
- 必须对列表进行排序
- 不停的插入元素、重新排序会非常慢

散列

- 键：基因名；值：表达值
- 相比折半查找有优势：速度快，明确是否被定义，添加或删除元素不需重排序
- 缺点：其中元素没有被排序

未排序的数组

- 需要遍历数组，速度很慢
- 必须要把基因名和表达值分割开来

排序数组和折半查找

- 查找速度大幅提升
- 必须对列表进行排序
- 不停的插入元素、重新排序会非常慢

散列

- 键：基因名；值：表达值
- 相比折半查找有优势：速度快，明确是否被定义，添加或删除元素不需重排序
- 缺点：其中元素没有被排序

未排序的数组

- 需要遍历数组，速度很慢
- 必须要把基因名和表达值分割开来

排序数组和折半查找

- 查找速度大幅提升
- 必须对列表进行排序
- 不停的插入元素、重新排序会非常慢

散列

- 键：基因名；值：表达值
- 相比折半查找有优势：速度快，明确是否被定义，添加或删除元素不需重排序
- 缺点：其中元素没有被排序

需求

- 看看某个元素是否在数据集中，元素的顺序无所谓
- 数据集需要排序、相对快速的查找，不需要频繁添加或删减元素
- 不需要对元素排序，但需要快速找到最新添加的元素
- 不需要对元素排序，但需要添加元素、移除“最老”的元素

数据结构

- 散列
- 排序数组结合折半查找
- 数组结合 push 和 pop 函数
- 数组结合 push 和 shift 函数

需求

- 看看某个元素是否在数据集中，元素的顺序无所谓
- 数据集需要排序、相对快速的查找，不需要频繁添加或删减元素
- 不需要对元素排序，但需要快速找到最新添加的元素
- 不需要对元素排序，但需要添加元素、移除“最老”的元素

数据结构

- 散列
- 排序数组结合折半查找
- 数组结合 push 和 pop 函数
- 数组结合 push 和 shift 函数

需求

- 看看某个元素是否在数据集中，元素的顺序无所谓
- 数据集需要排序、相对快速的查找，不需要频繁添加或删减元素
- 不需要对元素排序，但需要快速找到最新添加的元素
- 不需要对元素排序，但需要添加元素、移除“最老”的元素

数据结构

- 散列
- 排序数组结合折半查找
- 数组结合 push 和 pop 函数
- 数组结合 push 和 shift 函数

需求

- 看看某个元素是否在数据集中，元素的顺序无所谓
- 数据集需要排序、相对快速的查找，不需要频繁添加或删减元素
- 不需要对元素排序，但需要快速找到最新添加的元素
- 不需要对元素排序，但需要添加元素、移除“最老”的元素

数据结构

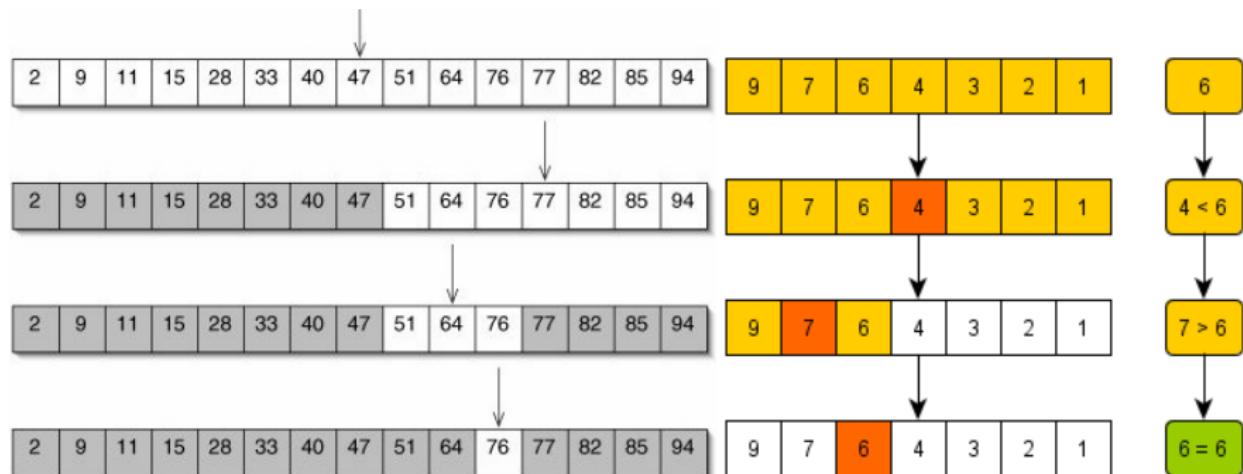
- 散列
- 排序数组结合折半查找
- 数组结合 push 和 pop 函数
- 数组结合 push 和 shift 函数

数据结构和算法 | 折半查找

```
1 # 折半查找 (binary search) 的伪代码
2 Given a sorted array, and an element:
3
4 Until you find the element or discover it's not
5 there,
6
7 Pick the midpoint of the array, $array[scalar(
8 @array)/2]
9
10 Compare your element with the element at the
11 midpoint
12
13 If that matches your element, you're done.
14 Else, ignore the half of the array that your
15 element is not in
16 }
```



数据结构和算法 | 折半查找



数据结构和算法 | 普通查找

```
1 #!/usr/bin/perl
2
3 use strict;
4 use warnings;
5
6 my @numbers = map { $_ * 3 } ( 0 .. 1000000 );
7
8 sub search {
9     my ( $numbers, $target ) = @_;
10    for my $i ( 0 .. $#numbers ) {
11        return $i if $numbers->[$i] == $target;
12    }
13    return;
14 }
15
16 print search( \@numbers, 699 ), "\n";
17 print search( \@numbers, 28 ), "\n";
```



数据结构和算法 | 折半查找

```
1 #!/usr/bin/perl
2 use strict; use warnings;
3
4 my @numbers = map { $_ * 3 } ( 0 .. 1000000 );
5 sub search {
6     my ( $numbers, $target ) = @_;
7     return binary_search( $numbers, $target, 0, $#$numbers );
8 }
9 sub binary_search {
10    my ( $numbers, $target, $low, $high ) = @_;
11    return if $high < $low;
12    my $middle = int( ( $low + $high ) / 2 );
13    if ( $numbers->[$middle] > $target ) {
14        return binary_search($numbers, $target, $low, $middle-1);
15    }
16    elsif ( $numbers->[$middle] < $target ) {
17        return binary_search($numbers, $target, $middle+1, $high);
18    }
19    return $middle;
20 }
21 print search( \@numbers, 699 ), "\n";
22 print search( \@numbers, 28 ), "\n";
```



数据结构和算法 | 比较与排序

```
1 # 两个字符串一样, 返回0
2 'ZZZ' cmp 'ZZZ'
3
4 # 两个字符串按字母顺序排列, 返回-1
5 'AAA' cmp 'ZZZ'
6
7 # 两个字符串按字母顺序的逆序排列, 返回1
8 'ZZZ' cmp 'AAA'
9
10 # 按照字母顺序/逆序对由字符串构成的数组进行排序
11 @array = sort @array;
12 @array = sort { $a cmp $b } @array;
13 @array = sort { $b cmp $a } @array;
14
15 # 以升序/降序对由数字构成的数组进行排序:
16 @array = sort { $a <=gt; $b } @array;
17 @array = sort { $b <=gt; $a } @array;
```



```
1 # 查看某个散列值是否被定义  
2 if ( defined $myhash{'mykey'} ) { ... }  
3  
4 # 对散列的键进行排序  
5 @sorted_keys = sort keys %my_hash;  
6  
7 # 对散列的值进行排序  
8 @sorted_values = sort values %my_hash;
```

散列

- Perl 的内置数据类型，易使用、易编程
- 对于程序员来说，懒惰的方法通常是最有效的方法（节省编程的时间通常要比节省程序运行的时间更加重要一些）

```
1 # 查看某个散列值是否被定义  
2 if ( defined $myhash{'mykey'} ) { ... }  
3  
4 # 对散列的键进行排序  
5 @sorted_keys = sort keys %my_hash;  
6  
7 # 对散列的值进行排序  
8 @sorted_values = sort values %my_hash;
```

散列

- Perl 的内置数据类型，易使用、易编程
- 对于程序员来说，懒惰的方法通常是最有效的方法（节省编程的时间通常要比节省程序运行的时间更加重要一些）

数据库

数据库，简单来说可视为电子化的文件柜——存储电子文件的处所，用户可以对文件中的数据运行新增、截取、更新、删除等操作。适用于存储和访问海量数据。

数据库指的是以一定方式储存在一起、能为多个用户共享、具有尽可能小的冗余度、与应用程序彼此独立的数据集合。

数据库管理系统

数据库管理系统（Database Management System, DBMS）是为管理数据库而设计的电脑软件系统，一般具有存储、截取、安全保障、备份等基础功能。

数据库管理系统可以依据它所支持的数据库模型来作分类（关系式、XML），或依据所支持的电脑类型来作分类（服务器群集、移动电话），或依据所用查询语言来作分类（SQL、XQuery），或依据性能冲量重点来作分类（最大规模、最高运行速度），亦或其他的分类方式。不论使用哪种分类方式，一些DBMS能够跨类，例如，同时支持多种查询语言。

数据库

数据库，简单来说可视为电子化的文件柜——存储电子文件的处所，用户可以对文件中的数据运行新增、截取、更新、删除等操作。适用于存储和访问海量数据。

数据库指的是以一定方式储存在一起、能为多个用户共享、具有尽可能小的冗余度、与应用程序彼此独立的数据集合。

数据库管理系统

数据库管理系统（Database Management System, DBMS）是为管理数据库而设计的电脑软件系统，一般具有存储、截取、安全保障、备份等基础功能。

数据库管理系统可以依据它所支持的数据库模型来作分类（关系式、XML），或依据所支持的电脑类型来作分类（服务器群集、移动电话），或依据所用查询语言来作分类（SQL、XQuery），或依据性能冲量重点来作分类（最大规模、最高运行速度），亦或其他的分类方式。不论使用哪种分类方式，一些DBMS能够跨类，例如，同时支持多种查询语言。

关系数据库 (relational database) , 是建立在关系模型基础上的数据库, 借助于集合代数等数学概念和方法来处理数据库中的数据。关系数据库是存储和提取海量数据最流行的方法。

关系数据库把数据组织成表格进行存储。表是以行和列的形式组织起来的数据的集合。一个数据库包括一个或多个表。例如, 可能有一个有关作者信息的名为 authors 的表。每列都包含特定类型的信息, 如作者的姓氏。每行都包含有关特定作者的所有信息: 姓、名、住址等等。在关系型数据库当中一个表就是一个关系, 一个关系数据库可以包含多个表。

数据通常通过一种查询语言进行输入和提取, 即结构化查询语言 (SQL, Structured Query Language) 语言。SQL 是 1974 年由 Boyce 和 Chamberlin 提出的一种介于关系代数与关系演算之间的结构化查询语言, 是一个通用的、功能极强的关系型数据库语言。



School Table

ID	Name
S001	University of Technology
S002	University of Applied Science

Student Table

School ID	ID	Name	DOB
S001	UT-1000	Tommy	05/06/1995
S001	UT-1000	Better	16/04/1995
S002	UAS-1000	Linda	02/09/1995
S002	UAS-1000	Jonathan	22/06/1995



The DBI(Perl's Database Interface) is the standard database interface module for Perl. It defines a set of methods, variables and conventions that provide a consistent database interface independent of the actual database being used.



DBM (database management) 是一种文件数据储存数据，由于采用散列结构进行链接，因此具有一些数据库的特点功能。与普通文本数据库相比，具有稳定、检索速度快和支持量大的优点。由于 DBM 是从 Unix 系统中移植来的，因此在 Unix/linux 系统中优点比较明显，而在 NT 系统中则不太理想，在 NT 中使用有时会令数据文件变得十分庞大。

DBM 数据库采用散列方式保存数据，并与散列结合使用。



DBM

Perl 中简单的、内置的一种方法，来存储散列数据

初始化

启动后，把一个散列“绑定”到计算机硬盘上的一个文件

使用

像使用散列一样使用它（查找、添加、删除……）

特色

- 简单且非常有用的数据库
- 适用于键-值数据集



DBM

Perl 中简单的、内置的一种方法，来存储散列数据

初始化

启动后，把一个散列“绑定”到计算机硬盘上的一个文件

使用

像使用散列一样使用它（查找、添加、删除……）

特色

- 简单且非常有用的数据库
- 适用于键-值数据集

DBM

Perl 中简单的、内置的一种方法，来存储散列数据

初始化

启动后，把一个散列“绑定”到计算机硬盘上的一个文件

使用

像使用散列一样使用它（查找、添加、删除……）

特色

- 简单且非常有用的数据库
- 适用于键-值数据集



DBM

Perl 中简单的、内置的一种方法，来存储散列数据

初始化

启动后，把一个散列“绑定”到计算机硬盘上的一个文件

使用

像使用散列一样使用它（查找、添加、删除……）

特色

- 简单且非常有用的数据库
- 适用于键-值数据集

Problem

You want to create, populate, inspect, or delete values in a DBM database.

Solution

Use **dbmopen** or **tie** to open the database and make it accessible through a hash. Then use the hash as you normally would. When you're done, call **dbmclose** or **untie**.



Problem

You want to create, populate, inspect, or delete values in a DBM database.

Solution

Use **dbmopen** or **tie** to open the database and make it accessible through a hash. Then use the hash as you normally would. When you're done, call **dbmclose** or **untie**.



Discussion

Accessing a database as a hash is powerful but easy, giving you a persistent hash that sticks around after the program using it has finished running. It's also much faster than loading in a new hash every time; even if the hash has a million entries, your program starts up virtually instantaneously.

The program treats the database as though it were a normal hash. You can even call **keys** or **each** on it. Likewise, **exists** and **defined** are implemented for tied DBM hashes. Unlike a normal hash, a DBM hash does not distinguish between those two functions.



```
1 use DB_File; # optional; overrides default
2 dbmopen %HASH, $FILENAME, 0666 # open
   database, accessed through %HASH
3 or die "Can't open $FILENAME:$!\\n";
4
5 $V = $HASH{$KEY}; # retrieve from database
6 $HASH{$KEY} = $VALUE; # put value into
   database
7 if (exists $HASH{$KEY}) { # check whether in
   database
8   # ...
9 }
10 delete $HASH{$KEY}; # remove from database
11 dbmclose %HASH; # close the database
```



数据结构和算法 | DBM | Perl | tie

```
1 use DB_File;    # load database module
2
3 tie %HASH, "DB_File", $FILENAME    # open
   database, to be accessed
4 or die "Can't open $FILENAME:$!\\n";  #
   through %HASH
5
6 $V = $HASH{$KEY};    # retrieve from database
7 $HASH{$KEY} = $VALUE;  # put value into
   database
8 if (exists $HASH{$KEY}) {    # check whether in
   database
9   # ...
10 }
11 delete $HASH{$KEY};  # delete from database
12 untie %HASH;    # close the database
```



数据结构和算法 | DBM | Perl | 实例

```
1 #!/usr/bin/perl
2
3 use warnings; use strict;
4 use DB_File;
5
6 my ( %h, $k, $v );
7 unlink "fruit";
8
9 tie %h, "DB_File", "fruit", O_RDWR | O_CREAT, 0666, $DB_HASH or
10 die "Cannot open file 'fruit': $!\n";
11
12 # Add a few key/value pairs to the file
13 $h{"apple"} = "red"; $h{"orange"} = "orange";
14 $h{"banana"} = "yellow"; $h{"tomato"} = "red";
15
16 # Check for existence of a key
17 print "Banana Exists\n\n" if $h{"banana"};
18
19 # Delete a key/value pair.
20 delete $h{"apple"};
21
22 # print the contents of the file
23 while ( ( $k, $v ) = each %h ) { print "$k -> $v\n" }
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
279
280
281
282
283
284
285
286
287
288
289
289
290
291
292
293
294
295
296
297
298
299
299
300
301
302
303
304
305
306
307
308
309
309
310
311
312
313
314
315
316
317
318
319
319
320
321
322
323
324
325
326
327
328
329
329
330
331
332
333
334
335
336
337
338
339
339
340
341
342
343
344
345
346
347
348
349
349
350
351
352
353
354
355
356
357
358
359
359
360
361
362
363
364
365
366
367
368
369
369
370
371
372
373
374
375
376
377
378
379
379
380
381
382
383
384
385
386
387
388
389
389
390
391
392
393
394
395
396
397
398
399
399
400
401
402
403
404
405
406
407
408
409
409
410
411
412
413
414
415
416
417
418
419
419
420
421
422
423
424
425
426
427
428
429
429
430
431
432
433
434
435
436
437
438
439
439
440
441
442
443
444
445
446
447
448
449
449
450
451
452
453
454
455
456
457
458
459
459
460
461
462
463
464
465
466
467
468
469
469
470
471
472
473
474
475
476
477
478
479
479
480
481
482
483
484
485
486
487
488
489
489
490
491
492
493
494
495
496
497
498
499
499
500
501
502
503
504
505
506
507
508
509
509
510
511
512
513
514
515
516
517
518
519
519
520
521
522
523
524
525
526
527
528
529
529
530
531
532
533
534
535
536
537
538
539
539
540
541
542
543
544
545
546
547
548
549
549
550
551
552
553
554
555
556
557
558
559
559
560
561
562
563
564
565
566
567
568
569
569
570
571
572
573
574
575
576
577
578
579
579
580
581
582
583
584
585
586
587
588
589
589
590
591
592
593
594
595
596
597
598
599
599
600
601
602
603
604
605
606
607
608
609
609
610
611
612
613
614
615
616
617
618
619
619
620
621
622
623
624
625
626
627
628
629
629
630
631
632
633
634
635
636
637
638
639
639
640
641
642
643
644
645
646
647
648
649
649
650
651
652
653
654
655
656
657
658
659
659
660
661
662
663
664
665
666
667
668
669
669
670
671
672
673
674
675
676
677
678
679
679
680
681
682
683
684
685
686
687
688
689
689
690
691
692
693
694
695
696
697
698
699
699
700
701
702
703
704
705
706
707
708
709
709
710
711
712
713
714
715
716
717
718
719
719
720
721
722
723
724
725
726
727
728
729
729
730
731
732
733
734
735
736
737
738
739
739
740
741
742
743
744
745
746
747
748
749
749
750
751
752
753
754
755
756
757
758
759
759
760
761
762
763
764
765
766
767
768
769
769
770
771
772
773
774
775
776
777
778
779
779
780
781
782
783
784
785
786
787
788
788
789
789
790
791
792
793
794
795
796
797
798
798
799
799
800
801
802
803
804
805
806
807
808
809
809
810
811
812
813
814
815
816
817
818
819
819
820
821
822
823
824
825
826
827
828
829
829
830
831
832
833
834
835
836
837
838
839
839
840
841
842
843
844
845
846
847
848
849
849
850
851
852
853
854
855
856
857
858
859
859
860
861
862
863
864
865
866
867
868
869
869
870
871
872
873
874
875
876
877
878
879
879
880
881
882
883
884
885
886
887
888
888
889
889
890
891
892
893
894
895
896
897
897
898
898
899
899
900
901
902
903
904
905
906
907
908
909
909
910
911
912
913
914
915
916
917
917
918
918
919
919
920
921
922
923
924
925
926
927
928
928
929
929
930
931
932
933
934
935
936
937
938
938
939
939
940
941
942
943
944
945
946
947
947
948
948
949
949
950
951
952
953
954
955
956
957
957
958
958
959
959
960
961
962
963
964
965
966
966
967
967
968
968
969
969
970
971
972
973
974
975
976
976
977
977
978
978
979
979
980
981
982
983
984
985
985
986
986
987
987
988
988
989
989
990
991
992
993
994
995
996
997
998
999
999
1000
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1038
1039
1039
1040
1041
1042
1043
1044
1045
1046
1047
1047
1048
1048
1049
1049
1050
1051
1052
1053
1054
1055
1056
1057
1057
1058
1058
1059
1059
1060
1061
1062
1063
1064
1065
1066
1066
1067
1067
1068
1068
1069
1069
1070
1071
1072
1073
1074
1075
1076
1076
1077
1077
1078
1078
1079
1079
1080
1081
1082
1083
1084
1085
1086
1086
1087
1087
1088
1088
1089
1089
1090
1091
1092
1093
1094
1095
1095
1096
1096
1097
1097
1098
1098
1099
1099
1100
1101
1102
1103
1104
1105
1105
1106
1106
1107
1107
1108
1108
1109
1109
1110
1111
1112
1113
1114
1115
1115
1116
1116
1117
1117
1118
1118
1119
1119
1120
1121
1122
1123
1124
1125
1125
1126
1126
1127
1127
1128
1128
1129
1129
1130
1131
1132
1133
1134
1135
1135
1136
1136
1137
1137
1138
1138
1139
1139
1140
1141
1142
1143
1144
1145
1145
1146
1146
1147
1147
1148
1148
1149
1149
1150
1151
1152
1153
1154
1155
1155
1156
1156
1157
1157
1158
1158
1159
1159
1160
1161
1162
1163
1164
1165
1165
1166
1166
1167
1167
1168
1168
1169
1169
1170
1171
1172
1173
1174
1175
1175
1176
1176
1177
1177
1178
1178
1179
1179
1180
1181
1182
1183
1184
1185
1185
1186
1186
1187
1187
1188
1188
1189
1189
1190
1191
1192
1193
1194
1194
1195
1195
1196
1196
1197
1197
1198
1198
1199
1199
1200
1201
1202
1203
1204
1204
1205
1205
1206
1206
1207
1207
1208
1208
1209
1209
1210
1211
1212
1213
1214
1214
1215
1215
1216
1216
1217
1217
1218
1218
1219
1219
1220
1221
1222
1223
1224
1225
1225
1226
1226
1227
1227
1228
1228
1229
1229
1230
1231
1232
1233
1234
1234
1235
1235
1236
1236
1237
1237
1238
1238
1239
1239
1240
1241
1242
1243
1244
1245
1245
1246
1246
1247
1247
1248
1248
1249
1249
1250
1251
1252
1253
1254
1255
1255
1256
1256
1257
1257
1258
1258
1259
1259
1260
1261
1262
1263
1264
1265
1265
1266
1266
1267
1267
1268
1268
1269
1269
1270
1271
1272
1273
1274
1275
1275
1276
1276
1277
1277
1278
1278
1279
1279
1280
1281
1282
1283
1284
1285
1285
1286
1286
1287
1287
1288
1288
1289
1289
1290
1291
1292
1293
1294
1294
1295
1295
1296
1296
1297
1297
1298
1298
1299
1299
1300
1301
1302
1303
1304
1304
1305
1305
1306
1306
1307
1307
1308
1308
1309
1309
1310
1311
1312
1313
1314
1314
1315
1315
1316
1316
1317
1317
1318
1318
1319
1319
1320
1321
1322
1323
1324
1324
1325
1325
1326
1326
1327
1327
1328
1328
1329
1329
1330
1331
1332
1333
1334
1334
1335
1335
1336
1336
1337
1337
1338
1338
1339
1339
1340
1341
1342
1343
1344
1344
1345
1345
1346
1346
1347
1347
1348
1348
1349
1349
1350
1351
1352
1353
1354
1354
1355
1355
1356
1356
1357
1357
1358
1358
1359
1359
1360
1361
1362
1363
1364
1364
1365
1365
1366
1366
1367
1367
1368
1368
1369
1369
1370
1371
1372
1373
1374
1374
1375
1375
1376
1376
1377
1377
1378
1378
1379
1379
1380
1381
1382
1383
1384
1384
1385
1385
1386
1386
1387
1387
1388
1388
1389
1389
1390
1391
1392
1393
1394
1394
1395
1395
1396
1396
1397
1397
1398
1398
1399
1399
1400
1401
1402
1403
1404
1404
1405
1405
1406
1406
1407
1407
1408
1408
1409
1409
1410
1411
1412
1413
1414
1414
1415
1415
1416
1416
1417
1417
1418
1418
1419
1419
1420
1421
1422
1423
1424
1424
1425
1425
1426
1426
1427
1427
1428
1428
1429
1429
1430
1431
1432
1433
1434
1434
1435
1435
1436
1436
1437
1437
1438
1438
1439
1439
1440
1441
1442
1443
1444
1444
1445
1445
1446
1446
1447
1447
1448
1448
1449
1449
1450
1451
1452
1453
1454
1454
1455
1455
1456
1456
1457
1457
1458
1458
1459
1459
1460
1461
1462
1463
1464
1464
1465
1465
1466
1466
1467
1467
1468
1468
1469
1469
1470
1471
1472
1473
1474
1474
1475
1475
1476
1476
1477
1477
1478
1478
1479
1479
1480
1481
1482
1483
1484
1484
1485
1485
1486
1486
1487
1487
1488
1488
1489
1489
1490
1491
1492
1493
1494
1494
1495
1495
1496
1496
1497
1497
1498
1498
1499
1499
1500
1501
1502
1503
1504
1504
1505
1505
1506
1506
1507
1507
1508
1508
1509
1509
1510
1511
1512
1513
1514
1514
1515
1515
1516
1516
1517
1517
1518
1518
1519
1519
1520
1521
1522
1523
1524
1524
1525
1525
1526
1526
1527
1527
1528
1528
1529
1529
1530
1531
1532
1533
1534
1534
1535
1535
1536
1536
1537
1537
1538
1538
1539
1539
1540
1541
1542
1543
1544
1544
1545
1545
1546
1546
1547
1547
1548
1548
1549
1549
1550
1551
1552
1553
1554
1554
1555
1555
1556
1556
1557
1557
1558
1558
1559
1559
1560
1561
1562
1563
1564
1564
1565
1565
1566
1566
1567
1567
1568
1568
1569
1569
1570
1571
1572
1573
1574
1574
1575
1575
1576
1576
1577
1577
1578
1578
1579
1579
1580
1581
1582
1583
1584
1584
1585
1585
1586
1586
1587
1587
1588
1588
1589
1589
1590
1591
1592
1593
1594
1594
1595
1595
1596
1596
1597
1597
1598
1598
1599
1599
1600
1601
1602
1603
1604
1604
1605
1605
1606
1606
1607
1607
1608
1608
1609
1609
1610
1611
1612
1613
1614
1614
1615
1615
1616
1616
1617
1617
1618
1618
1619
1619
1620
1621
1622
1623
1624
1624
1625
1625
1626
1626
1627
1627
1628
1628
1629
1629
1630
1631
1632
1633
1634
1634
1635
1635
1636
1636
1637
1637
1638
1638
1639
1639
1640
1641
1642
1643
1644
1644
1645
1645
1646
1646
1647
1647
1648
1648
1649
1649
1650
1651
1652
1653
1654
1654
1655
1655
1656
1656
1657
1657
1658
1658
1659
1659
1660
1661
1662
1663
1664
1664
1665
1665
1666
1666
1667
1667
1668
1668
1669
1669
1670
1671
1672
1673
1674
1674
1675
1675
1676
1676
1677
1677
1678
1678
1679
1679
1680
1681
1682
1683
1684
1684
1685
1685
1686
1686
1687
1687
1688
1688
1689
1689
1690
1691
1692
1693
1694
1694
1695
1695
1696
1696
1697
1697
1698
1698
1699
1699
1700
1701
1702
1703
1704
1704
1705
1705
1706
1706
1707
1707
1708
1708
1709
1709
1710
1711
1712
1713
1714
1714
1715
1715
1716
1716
1717
1717
1718
1718
1719
1719
1720
1721
1722
1723
1724
1724
1725
1725
1726
1726
1727
1727
1728
1728
1729
1729
1730
1731
1732
1733
1734
1734
1735
1735
1736
1736
1737
1737
1738
1738
1739
1739
1740
1741
1742
1743
1744
1744
1745
1745
1746
1746
1747
1747
1748
1748
1749
1749
1750
1751
1752
1753
1754
1754
1755
1755
1756
1756
1757
1757
1758
1758
1759
1759
1760
1761
1762
1763
1764
1764
1765
1765
1766
1766
1767
1767
1768
1768
1769
1769
1770
1771
1772
1773
1774
1774
1775
1775
1776
1776
1777
1777
1778
1778
1779
1779
1780
1781
1782
1783
1784
1784
1785
1785
1786
1786
1787
1787
1788
1788
1789
1789
1790
1791
1792
1793
1794
1794
1795
1795
1796
1796
1797
1797
1798
1798
1799
1799
1800
1801
1802
1803
1804
1804
1805
1805
1806
1806
1807
1807
1808
1808
1809
1809
1810
1811
1812
1813
1814
1814
1815
1815
1816
1816
1817
1817
1818
1818
1819
1819
1820
1821
1822
1823
1824
1824
1825
1825
1826
1826
1827
1827
1828
1828
1829
1829
1830
1831
1832
1833
1834
1834
1835
1835
1836
1836
1837
1837
1838
1838
1839
1839
1840
1841
1842
1843
1844
1844
1845
1845
1846
1846
1847
1847
1848
1848
1849
1849
1850
1851
1852
1853
1854
1854
1855
1855
1856
1856
1857
1857
1858
1858
1859
1859
1860
1861
1862
1863
1864
1864
1865
1865
1866
1866
1867
1867
1868
1868
1869
1869
1870
1871
1872
1873
1874
1874
1875
1875
1876
1876
1877
1877
1878
1878
1879
1879
1880
1881
1882
1883
1884
1884
1885
1885
1886
1886
1887
1887
1888
1888
1889
1889
1890
1891
1892
1893
1894
1894
1895
1895
1896
1896
1897
1897
1898
1898
1899
1899
1900
1901
1902
1903
1904
1904
1905
1905
1906
1906
1907
1907
1908
1908
1909
1909
1910
1911
1912
1913
1914
1914
1915
1915
1916
1916
1917
1917
1918
1918
1919
1919
1920
1921
1922
1923
1924
1924
1925
1925
1926
1926
1927
1927
1928
1928
1929
1929
1930
1931
1932
1933
1934
1934
1935
1935
1936
1936
1937
1937
1938
1938
1939
1939
1940
1941
1942
1943
1944
1944
1945
1945
1946
1946
1947
1947
1948
1948
1949
1949
1950
1951
1952
1953
1954
1954
1955
1955
1956
1956
1957
1957
1958
1958
1959
1959
1960
1961
1962
1963
1964
1964
1965
1965
1966
1966
1967
1967
1968
1968
1969
1969
1970
1971
1972
1973
1974
1974
1975
1975
1976
1976
1977
1977
1978
1978
1979
1979
1980
1981
1982
1983
1984
1984
1985
1985
1986
1986
1987
1987
1988
1988
1989
1989
1990
1991
1992
1993
1994
1994
1995
1995
1996
1996
1997
1997
1998
1998
1999
1999
2000
2001
2002
2003
2004
2004
2005
2005
2006
2006
2007
2007
2008
2008
2009
2009
2010
2011
20
```

数据结构和算法 | DBM | Perl | 实例

```
1 #!/usr/bin/perl
2 use warnings; use strict;
3
4 my %MYDB;
5 # 创建或打开DBM; 如果是新建DBM, 会自动生成mydb.dir和mydb
6 .pag两个文件
6 dbmopen(%MYDB, "mydb", 0644) or die "Can not
    access the database";
7
8 # 添加key/value对
9 $MYDB{name1} = "will1"; $MYDB{name2} = "will2";
10 # 读取内容
11 foreach my $key (keys %MYDB) {
12     print "$key\t$MYDB{$key}\n";
13 }
14
15 # 关闭DBM
16 dbmclose(%MYDB) or die "Can not close database";
```



教学提纲

1

引言

2

散列

3

数据结构和算法

4

遗传密码

- 密码子翻译成氨基酸
- 遗传密码的冗余性
- 使用散列表表示遗传密码

5

DNA 翻译成蛋白质

6

读取 FASTA 格式的 DNA

- 序列格式
- 读取 FASTA 文件的思路
- 读取 FASTA 文件的子程序
- 输出格式化的序列数据

- 读取 FASTA 文件的主程序
- DNA 翻译成蛋白质的主程序

7

阅读框

- 简介
- 翻译阅读框

8

回顾和总结

- 总结
- 思考题

9

知识拓展

- Perl 里面的复杂数据结构
- 数据共享
- 宽格式和长格式的数据
- 数据处理实例

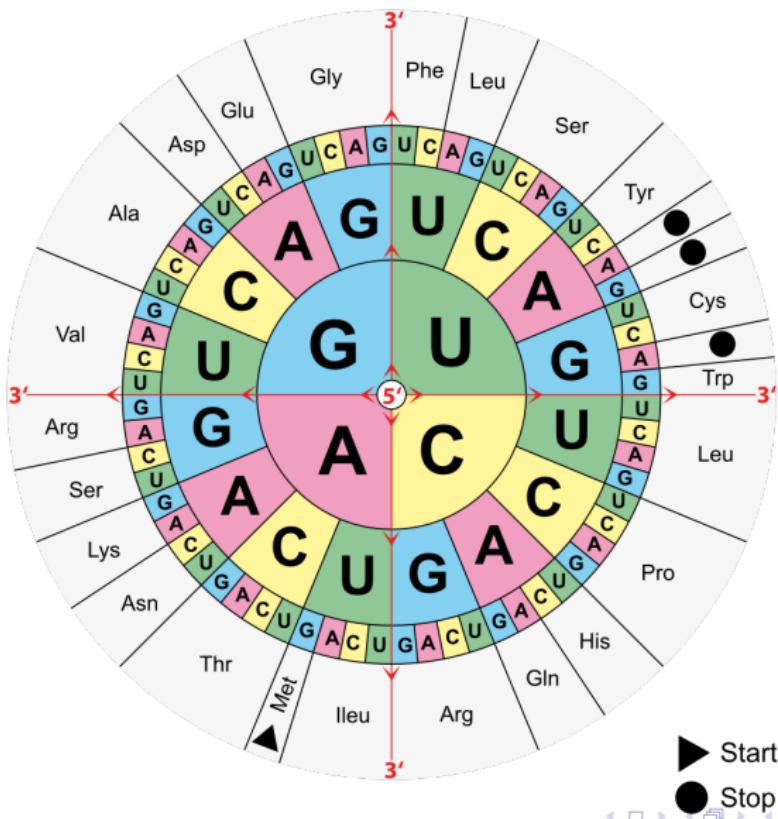


遗传密码 (genetic code) 是一组规则，将 DNA 或 mRNA 序列以三个核苷酸为一组的密码子 (codon) 翻译为蛋白质的氨基酸序列，用于蛋白质合成。

几乎所有的生物都使用同样的遗传密码，称为标准遗传密码；即使是非细胞结构的病毒，它们也是使用标准遗传密码。但是也有少数生物使用一些稍微不同的遗传密码。



遗传密码 | 简介



教学提纲

1

引言

2

散列

3

数据结构和算法

4

遗传密码

- 密码子翻译成氨基酸
- 遗传密码的冗余性
- 使用散列表表示遗传密码

5

DNA 翻译成蛋白质

6

读取 FASTA 格式的 DNA

- 序列格式
- 读取 FASTA 文件的思路
- 读取 FASTA 文件的子程序
- 输出格式化的序列数据

- 读取 FASTA 文件的主程序
- DNA 翻译成蛋白质的主程序

7

阅读框

- 简介
- 翻译阅读框

8

回顾和总结

- 总结
- 思考题

9

知识拓展

- Perl 里面的复杂数据结构
- 数据共享
- 宽格式和长格式的数据
- 数据处理实例



生物学目的

把三个核苷酸的密码子翻译成氨基酸。

程序实现

输入三字母的 DNA 密码子，返回一个单字母缩写表示的氨基酸。



生物学目的

把三个核苷酸的密码子翻译成氨基酸。

程序实现

输入三字母的 DNA 密码子，返回一个单字母缩写表示的氨基酸。



遗传密码 | 密码子翻译

```
1 # codon2aa
2 #
3 # A subroutine to translate a DNA 3-character codon to
4 # an amino acid
5 sub codon2aa {
6     my($codon) = @_;
7
8     if ( $codon =~ /TCA/i ) { return 'S' } # Serine
9     elsif ( $codon =~ /TCC/i ) { return 'S' } # Serine
10    elsif ( $codon =~ /TCG/i ) { return 'S' } # Serine
11    elsif ( $codon =~ /TCT/i ) { return 'S' } # Serine
12    elsif ( $codon =~ /TTC/i ) { return 'F' } #
13        Phenylalanine
14    elsif ( $codon =~ /TTT/i ) { return 'F' } #
15        Phenylalanine
16    elsif ( $codon =~ /TTA/i ) { return 'L' } # Leucine
17    elsif ( $codon =~ /TTG/i ) { return 'L' } # Leucine
```



遗传密码 | 密码子翻译

```
16  elsif ( $codon =~ /TAC/i ) { return 'Y' } # Tyrosine
17  elsif ( $codon =~ /TAT/i ) { return 'Y' } # Tyrosine
18  elsif ( $codon =~ /TAA/i ) { return '-' } # Stop
19  elsif ( $codon =~ /TAG/i ) { return '-' } # Stop
20  elsif ( $codon =~ /TGC/i ) { return 'C' } # Cysteine
21  elsif ( $codon =~ /TGT/i ) { return 'C' } # Cysteine
22  elsif ( $codon =~ /TGA/i ) { return '-' } # Stop
23  elsif ( $codon =~ /TGG/i ) { return 'W' } # Tryptophan
24  elsif ( $codon =~ /CTA/i ) { return 'L' } # Leucine
25  elsif ( $codon =~ /CTC/i ) { return 'L' } # Leucine
26  elsif ( $codon =~ /CTG/i ) { return 'L' } # Leucine
27  elsif ( $codon =~ /CTT/i ) { return 'L' } # Leucine
28  elsif ( $codon =~ /CCA/i ) { return 'P' } # Proline
29  elsif ( $codon =~ /CCC/i ) { return 'P' } # Proline
30  elsif ( $codon =~ /CCG/i ) { return 'P' } # Proline
31  elsif ( $codon =~ /CCT/i ) { return 'P' } # Proline
32  elsif ( $codon =~ /CAC/i ) { return 'H' } # Histidine
33  elsif ( $codon =~ /CAT/i ) { return 'H' } # Histidine
```



遗传密码 | 密码子翻译

```
34  elsif ( $codon =~ /CAA/i ) { return 'Q' } # Glutamine
35  elsif ( $codon =~ /CAG/i ) { return 'Q' } # Glutamine
36  elsif ( $codon =~ /CGA/i ) { return 'R' } # Arginine
37  elsif ( $codon =~ /CGC/i ) { return 'R' } # Arginine
38  elsif ( $codon =~ /CGG/i ) { return 'R' } # Arginine
39  elsif ( $codon =~ /CGT/i ) { return 'R' } # Arginine
40  elsif ( $codon =~ /ATA/i ) { return 'I' } # Isoleucine
41  elsif ( $codon =~ /ATC/i ) { return 'I' } # Isoleucine
42  elsif ( $codon =~ /ATT/i ) { return 'I' } # Isoleucine
43  elsif ( $codon =~ /ATG/i ) { return 'M' } # Methionine
44  elsif ( $codon =~ /ACA/i ) { return 'T' } # Threonine
45  elsif ( $codon =~ /ACC/i ) { return 'T' } # Threonine
46  elsif ( $codon =~ /ACG/i ) { return 'T' } # Threonine
47  elsif ( $codon =~ /ACT/i ) { return 'T' } # Threonine
48  elsif ( $codon =~ /AAC/i ) { return 'N' } # Asparagine
49  elsif ( $codon =~ /AAT/i ) { return 'N' } # Asparagine
50  elsif ( $codon =~ /AAA/i ) { return 'K' } # Lysine
51  elsif ( $codon =~ /AAG/i ) { return 'K' } # Lysine
```



遗传密码 | 密码子翻译

```
52  elsif ( $codon =~ /AGC/i ) { return 'S' } # Serine
53  elsif ( $codon =~ /AGT/i ) { return 'S' } # Serine
54  elsif ( $codon =~ /AGA/i ) { return 'R' } # Arginine
55  elsif ( $codon =~ /AGG/i ) { return 'R' } # Arginine
56  elsif ( $codon =~ /GTA/i ) { return 'V' } # Valine
57  elsif ( $codon =~ /GTC/i ) { return 'V' } # Valine
58  elsif ( $codon =~ /GTG/i ) { return 'V' } # Valine
59  elsif ( $codon =~ /GTT/i ) { return 'V' } # Valine
60  elsif ( $codon =~ /GCA/i ) { return 'A' } # Alanine
61  elsif ( $codon =~ /GCC/i ) { return 'A' } # Alanine
62  elsif ( $codon =~ /GCG/i ) { return 'A' } # Alanine
63  elsif ( $codon =~ /GCT/i ) { return 'A' } # Alanine
64  elsif ( $codon =~ /GAC/i ) { return 'D' } # Aspartic
     Acid
65  elsif ( $codon =~ /GAT/i ) { return 'D' } # Aspartic
     Acid
```

遗传密码 | 密码子翻译

```
66  elsif ( $codon =~ /GAA/i ) { return 'E' } # Glutamic  
Acid  
67  elsif ( $codon =~ /GAG/i ) { return 'E' } # Glutamic  
Acid  
68  elsif ( $codon =~ /GGA/i ) { return 'G' } # Glycine  
69  elsif ( $codon =~ /GGC/i ) { return 'G' } # Glycine  
70  elsif ( $codon =~ /GGG/i ) { return 'G' } # Glycine  
71  elsif ( $codon =~ /GGT/i ) { return 'G' } # Glycine  
72  else {  
73      print STDERR "Bad codon \"$codon\"!!\n";  
74      exit;  
75  }  
76 }
```



优点

- 代码清晰简单
- 排版布局美观
- 两者相得益彰，使整个翻译过程一目了然

缺点

- 大量的字符串比较
- 运行起来耗费时间



优点

- 代码清晰简单
- 排版布局美观
- 两者相得益彰，使整个翻译过程一目了然

缺点

- 大量的字符串比较
- 运行起来耗费时间



STDIN

- 标准输入（默认键盘）

STDOUT

- 标准输出（默认屏幕）
- print 默认使用 STDOUT
- print 可以使用一个文件句柄作为可选的参数

STDERR

- 标准错误输出（默认屏幕）



STDIN

- 标准输入（默认键盘）

STDOUT

- 标准输出（默认屏幕）
- print 默认使用 STDOUT
- print 可以使用一个文件句柄作为可选的参数

STDERR

- 标准错误输出（默认屏幕）



STDIN

- 标准输入（默认键盘）

STDOUT

- 标准输出（默认屏幕）
- print 默认使用 STDOUT
- print 可以使用一个文件句柄作为可选的参数

STDERR

- 标准错误输出（默认屏幕）



教学提纲

1

引言

2

散列

3

数据结构和算法

4

遗传密码

- 密码子翻译成氨基酸
- 遗传密码的冗余性
- 使用散列表表示遗传密码

5

DNA 翻译成蛋白质

6

读取 FASTA 格式的 DNA

- 序列格式
- 读取 FASTA 文件的思路
- 读取 FASTA 文件的子程序
- 输出格式化的序列数据

- 读取 FASTA 文件的主程序
- DNA 翻译成蛋白质的主程序

7

阅读框

- 简介
- 翻译阅读框

8

回顾和总结

- 总结
- 思考题

9

知识拓展

- Perl 里面的复杂数据结构
- 数据共享
- 宽格式和长格式的数据
- 数据处理实例



遗传密码 | 冗余性

碱基

$$4 = A + C + G + T/U$$

氨基酸

$$21 = 20(\text{氨基酸}) + 1(\text{起始, 同 M}) + 1(\text{终止})$$

密码子

- 每 3 个碱基组成一个密码子
- 一个密码子编码一种氨基酸
- $4 \times 4 \times 4 = 64 > 21$



遗传密码 | 冗余性

碱基

$$4 = A + C + G + T/U$$

氨基酸

$$21 = 20(\text{氨基酸}) + 1(\text{起始, 同 M}) + 1(\text{终止})$$

密码子

- 每 3 个碱基组成一个密码子
- 一个密码子编码一种氨基酸
- $4 \times 4 \times 4 = 64 > 21$



遗传密码 | 冗余性

碱基

$$4 = A + C + G + T/U$$

氨基酸

$$21 = 20(\text{氨基酸}) + 1(\text{起始, 同 M}) + 1(\text{终止})$$

密码子

- 每 3 个碱基组成一个密码子
- 一个密码子编码一种氨基酸
- $4 \times 4 \times 4 = 64 > 21$



遗传密码 | 冗余性

nonpolar polar basic acidic (stop codon)

Standard genetic code

1st base	2nd base								3rd base
	U		C		A		G		
U	UUU	(Phe/F) Phenylalanine		UCU	(Ser/S) Serine	UAU	(Tyr/Y) Tyrosine		U
	UUC			UCC		UAC	UGC	C	
	UUA			UCA		UAA	Stop (Ochre)		A
	UUG			UCG		UAG	Stop (Amber)		G
C	CUU	(Leu/L) Leucine	CCU	(Pro/P) Proline	CAU	(His/H) Histidine		CGU	U
	CUC		CCC		CAC			CGC	C
	CUA		CCA		CAA	(Gln/Q) Glutamine		CGA	A
	CUG		CCG		CAG			CGG	G
A	AUU	(Ile/I) Isoleucine	ACU	(Thr/T) Threonine	AAU	(Asn/N) Asparagine		AGU	U
	AUC		ACC		AAC			AGC	C
	AUA		ACA		AAA	(Lys/K) Lysine		AGA	A
	AUG ^[A]		(Met/M) Methionine		AAG			AGG	G
G	GUU	(Val/V) Valine	GCU	(Ala/A) Alanine	GAU	(Asp/D) Aspartic acid		GGU	U
	GUC		GCC		GAC			GGC	C
	GUА		GCA		GAA	(Glu/E) Glutamic acid		GGA	A
	GUG		GCG		GAG			GGG	G

遗传密码 | 冗余性

Inverse table (compressed using IUPAC notation)

Amino acid	Codons	Compressed	Amino acid	Codons	Compressed
Ala/A	GCU, GCC, GCA, GCG	GCN	Leu/L	UUA, UUG, CUU, CUC, CUA, CUG	YUR, CUN
Arg/R	CGU, CGC, CGA, CGG, AGA, AGG	CGN, MGR	Lys/K	AAA, AAG	AAR
Asn/N	AAU, AAC	AAY	Met/M	AUG	
Asp/D	GAU, GAC	GAY	Phe/F	UUU, UUC	UUY
Cys/C	UGU, UGC	UGY	Pro/P	CCU, CCC, CCA, CCG	CCN
Gln/Q	CAA, CAG	CAR	Ser/S	UCU, UCC, UCA, UCG, AGU, AGC	UCN, AGY
Glu/E	GAA, GAG	GAR	Thr/T	ACU, ACC, ACA, ACG	ACN
Gly/G	GGU, GGC, GGA, GGG	GGN	Trp/W	UGG	
His/H	CAU, CAC	CAY	Tyr/Y	UAU, UAC	UAY
Ile/I	AUU, AUC, AUA	AUH	Val/V	GUU, GUC, GUA, GUG	GUN
START	AUG		STOP	UAA, UGA, UAG	UAR, URA



遗传密码 | 冗余性

```
1 # codon2aa
2 #
3 # A subroutine to translate a DNA 3-character codon to an amino
4 # acid
5 #
6 sub codon2aa {
7     my($codon) = @_;
8
9     if ( $codon =~ /GC./i ) { return 'A' } # Alanine
10    elsif ( $codon =~ /TG[TC]/i ) { return 'C' } # Cysteine
11    elsif ( $codon =~ /GA[TC]/i ) { return 'D' } # Aspartic Acid
12    elsif ( $codon =~ /GA[AG]/i ) { return 'E' } # Glutamic Acid
13    elsif ( $codon =~ /TT[TC]/i ) { return 'F' } # Phenylalanine
14    elsif ( $codon =~ /GG./i ) { return 'G' } # Glycine
15    elsif ( $codon =~ /CA[TC]/i ) { return 'H' } # Histidine
16    elsif ( $codon =~ /AT[TCA]/i ) { return 'I' } # Isoleucine
17    elsif ( $codon =~ /AA[AG]/i ) { return 'K' } # Lysine
18    elsif ( $codon =~ /TT[AG]|CT./i ) { return 'L' } # Leucine
19    elsif ( $codon =~ /ATG/i ) { return 'M' } # Methionine
20    elsif ( $codon =~ /AA[TC]/i ) { return 'N' } # Asparagine
```



遗传密码 | 冗余性

```
21  elsif ( $codon =~ /CC./i ) { return 'P' } # Proline
22  elsif ( $codon =~ /CA[AG]/i ) { return 'Q' } # Glutamine
23  elsif ( $codon =~ /CG.|AG[AG]/i ) { return 'R' } # Arginine
24  elsif ( $codon =~ /TC.|AG[TC]/i ) { return 'S' } # Serine
25  elsif ( $codon =~ /AC./i ) { return 'T' } # Threonine
26  elsif ( $codon =~ /GT./i ) { return 'V' } # Valine
27  elsif ( $codon =~ /TGG/i ) { return 'W' } # Tryptophan
28  elsif ( $codon =~ /TA[TC]/i ) { return 'Y' } # Tyrosine
29  elsif ( $codon =~ /TA[AG]|TGA/i ) { return '_' } # Stop
30  else {
31      print STDERR "Bad codon \"$codon\"!!\n";
32      exit;
33  }
34 }
```



优势

- 使用正则表达式展示了遗传密码的冗余性
- 氨基酸的单字母代码是按照字母顺序排列的

正则表达式

- `/ [TC] /`: 匹配 T 或者 C
- `/ . /`: 匹配换行符以外的任意一个字符 (此处为 ACGT)
- `/T/i`: 不区分大小写, 匹配 T 或者 t
- `/TC.|AG[TC]/`: 匹配 `/TC./` 或者 `/AG[TC]/`
- `/TA[AG]|TGA/`: 等同于 `/T(A[AG]|GA)/`



优势

- 使用正则表达式展示了遗传密码的冗余性
- 氨基酸的单字母代码是按照字母顺序排列的

正则表达式

- `/ [TC] /`: 匹配 T 或者 C
- `/ . /`: 匹配换行符以外的任意一个字符 (此处为 ACGT)
- `/ T /i`: 不区分大小写, 匹配 T 或者 t
- `/ TC . | AG [TC] /`: 匹配 `/TC./` 或者 `/AG[TC]/`
- `/ TA [AG] | TGA /`: 等同于 `/T(A[AG] | GA)/`



教学提纲

1

引言

2

散列

3

数据结构和算法

4

遗传密码

- 密码子翻译成氨基酸
- 遗传密码的冗余性
- 使用散列表表示遗传密码

5

DNA 翻译成蛋白质

6

读取 FASTA 格式的 DNA

- 序列格式
- 读取 FASTA 文件的思路
- 读取 FASTA 文件的子程序
- 输出格式化的序列数据

- 读取 FASTA 文件的主程序
- DNA 翻译成蛋白质的主程序

7

阅读框

- 简介
- 翻译阅读框

8

回顾和总结

- 总结
- 思考题

9

知识拓展

- Perl 里面的复杂数据结构
- 数据共享
- 宽格式和长格式的数据
- 数据处理实例



遗传密码 | 散列

```
1 #
2 # codon2aa
3 #
4 # A subroutine to translate a DNA 3-character
5 # codon to an amino acid
6 #
7 sub codon2aa {
8     my($codon) = @_;
9
10    $codon = uc $codon;
```



遗传密码 | 散列

```
12 my(%genetic_code) = (
13
14     'TCA' => 'S',      # Serine
15     'TCC' => 'S',      # Serine
16     'TCG' => 'S',      # Serine
17     'TCT' => 'S',      # Serine
18     'TTC' => 'F',      # Phenylalanine
19     'TTT' => 'F',      # Phenylalanine
20     'TTA' => 'L',      # Leucine
21     'TTG' => 'L',      # Leucine
22     'TAC' => 'Y',      # Tyrosine
23     'TAT' => 'Y',      # Tyrosine
24     'TAA' => '—',      # Stop
25     'TAG' => '—',      # Stop
```



遗传密码 | 散列

```
26 'TGC' => 'C',      # Cysteine
27 'TGT' => 'C',      # Cysteine
28 'TGA' => ' ',       # Stop
29 'TGG' => 'W',       # Tryptophan
30 'CTA' => 'L',       # Leucine
31 'CTC' => 'L',       # Leucine
32 'CTG' => 'L',       # Leucine
33 'CTT' => 'L',       # Leucine
34 'CCA' => 'P',       # Proline
35 'CCC' => 'P',       # Proline
36 'CCG' => 'P',       # Proline
37 'CCT' => 'P',       # Proline
38 'CAC' => 'H',       # Histidine
39 'CAT' => 'H',       # Histidine
40 'CAA' => 'Q',       # Glutamine
41 'CAG' => 'Q',       # Glutamine
42 'CGA' => 'R',       # Arginine
43 'CGC' => 'R',       # Arginine
44 'CGG' => 'R',       # Arginine
45 'CGT' => 'R',       # Arginine
```

遗传密码 | 散列

```
46 'ATA' => 'I',      # Isoleucine
47 'ATC' => 'I',      # Isoleucine
48 'ATT' => 'I',      # Isoleucine
49 'ATG' => 'M',      # Methionine
50 'ACA' => 'T',      # Threonine
51 'ACC' => 'T',      # Threonine
52 'ACG' => 'T',      # Threonine
53 'ACT' => 'T',      # Threonine
54 'AAC' => 'N',      # Asparagine
55 'AAT' => 'N',      # Asparagine
56 'AAA' => 'K',      # Lysine
57 'AAG' => 'K',      # Lysine
58 'AGC' => 'S',      # Serine
59 'AGT' => 'S',      # Serine
60 'AGA' => 'R',      # Arginine
61 'AGG' => 'R',      # Arginine
62 'GTA' => 'V',      # Valine
63 'GTC' => 'V',      # Valine
64 'GTG' => 'V',      # Valine
65 'GTT' => 'V',      # Valine
```

遗传密码 | 散列

```
66  'GCA' => 'A',      # Alanine
67  'GCC' => 'A',      # Alanine
68  'GCG' => 'A',      # Alanine
69  'GCT' => 'A',      # Alanine
70  'GAC' => 'D',      # Aspartic Acid
71  'GAT' => 'D',      # Aspartic Acid
72  'GAA' => 'E',      # Glutamic Acid
73  'GAG' => 'E',      # Glutamic Acid
74  'GGA' => 'G',      # Glycine
75  'GGC' => 'G',      # Glycine
76  'GGG' => 'G',      # Glycine
77  'GGT' => 'G',      # Glycine
78 );
79
80 if(exists $genetic_code{$codon}) {
81     return $genetic_code{$codon};
82 }else{
83     print STDERR "Bad codon \"$codon\"!!\n";
84     exit;
85 }
```



代码解释

- uc \$codon: 把输入的参数转换为大写（子程序就可以同时处理大小写了）
- exists \$genetic_code{\$condon}: 散列中存在 \$codon 这个键就返回 true, 否则返回 false

补充说明

- 散列的键必须是简单的标量值，不能使用正则表达式，不能使用字符集
- 使用子程序（做成“黑盒子”）对程序进行模块化组织
- 把 codon2aa 这个子程序放到 BeginPerlBioinfo.pm 模块文件中备用

代码解释

- `uc $codon`: 把输入的参数转换为大写（子程序就可以同时处理大小写了）
- `exists $genetic_code{$condon}`: 散列中存在 \$codon 这个键就返回 true, 否则返回 false

补充说明

- 散列的键必须是简单的标量值，不能使用正则表达式，不能使用字符集
- 使用子程序（做成“黑盒子”）对程序进行模块化组织
- 把 `codon2aa` 这个子程序放到 `BeginPerlBioinfo.pm` 模块文件中备用

教学提纲

1

引言

2

散列

3

数据结构和算法

4

遗传密码

- 密码子翻译成氨基酸
- 遗传密码的冗余性
- 使用散列表表示遗传密码

5

DNA 翻译成蛋白质

6

读取 FASTA 格式的 DNA

- 序列格式
- 读取 FASTA 文件的思路
- 读取 FASTA 文件的子程序
- 输出格式化的序列数据

- 读取 FASTA 文件的主程序
- DNA 翻译成蛋白质的主程序

7

阅读框

- 简介
- 翻译阅读框

8

回顾和总结

- 总结
- 思考题

9

知识拓展

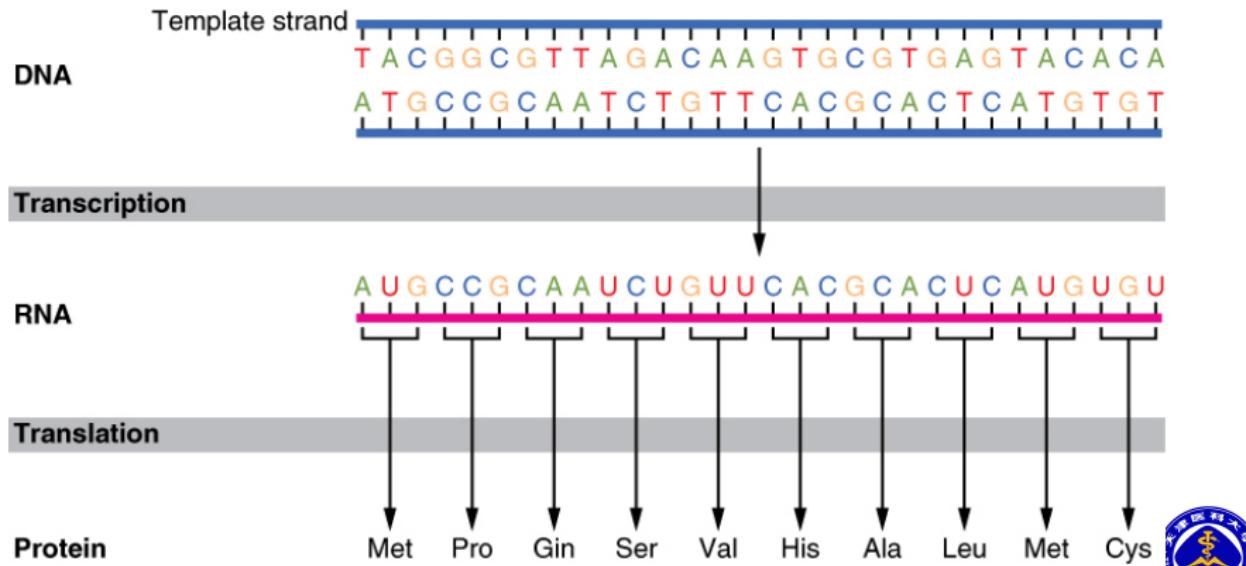
- Perl 里面的复杂数据结构
- 数据共享
- 宽格式和长格式的数据
- 数据处理实例



DNA 翻译成蛋白质 | 目的

目的

使用子程序 codon2aa 把整个 DNA 序列翻译成蛋白质。



DNA 翻译成蛋白质 | 程序 8.1.1

```
1 #!/usr/bin/perl -w
2 # Example 8-1      Translate DNA into protein
3
4 use strict;
5 use warnings;
6 use Bio::Perl::Bioinfo;          # see Chapter 6
7                         about this module
8
9 # Initialize variables
10 my $dna = '
11             CGACGTCTCGTACGGGACTAGCTCGTGTGGTCGC';
12 my $protein = '';
13 my $codon;
```



DNA 翻译成蛋白质 | 程序 8.1.2

```
13 # Translate each three-base codon into an
   amino acid, and append to a protein
14 for(my $i=0; $i < (length($dna) - 2) ; $i +=
   3) {
15     $codon = substr($dna,$i,3);
16     $protein .= codon2aa($codon);
17 }
18
19 print "I translated the DNA\n\n$dna\n\n into
   the protein\n\n$protein\n\n";
20
21 exit;
```



DNA 翻译成蛋白质 | 程序 8.1 | 输出

```
1 I translated the DNA
2
3 CGACGTCTCGTACGGGACTAGCTCGTGTGGTCGC
4
5 into the protein
6
7 RRLRTGLARVGR
```



DNA 翻译成蛋白质 | 程序 8.1 | 循环

```
1 for (my $i = 0; $i < (length($dna) - 2); $i  
    += 3) {
```

解释

- my \$i=0: 初始化计数器，限定作用域
- \$i<(length(\$dna)-2): 判断是否继续翻译，密码子长度为 3, DNA 序列长 length, 其中的碱基索引从 0 到 length-1
- \$i+=3: 计数器递增，一次处理 3 个碱基



DNA 翻译成蛋白质 | 程序 8.1 | 循环

```
1 for (my $i = 0; $i < (length($dna) - 2); $i  
    += 3) {
```

解释

- my \$i=0: 初始化计数器，限定作用域
- \$i<(length(\$dna)-2): 判断是否继续翻译，密码子长度为 3, DNA 序列长 length, 其中的碱基索引从 0 到 length-1
- \$i+=3: 计数器递增，一次处理 3 个碱基



DNA 翻译成蛋白质 | 程序 8.1 | 提取密码子

```
1 $codon = substr ($dna, $i, 3);
```

解释

- 从字符串 \$dna 的位置 \$i 开始向后提取 3 个字符
- 把提取的子字符串保存到变量 \$codon 中



```
1 $codon = substr ($dna, $i, 3);
```

解释

- 从字符串 \$dna 的位置 \$i 开始向后提取 3 个字符
- 把提取的子字符串保存到变量 \$codon 中



基本要求

- 输入：包含 DNA 的参数
- 返回：翻译后的肽链

问题简化

- 翻译整段 DNA (不需要提供指定起始和终止位点的参数)
- 移除不适合于子程序的 print 语句



基本要求

- 输入：包含 DNA 的参数
- 返回：翻译后的肽链

问题简化

- 翻译整段 DNA (不需要提供指定起始和终止位点的参数)
- 移除不适合于子程序的 print 语句



DNA 翻译成蛋白质 | 子程序

```
1 # dna2peptide
2 #
3 # A subroutine to translate DNA sequence into
4 # a peptide
5
6
7 sub dna2peptide {
8
9     my ($dna) = @_;
10
11    use strict;
12    use warnings;
13    use BeginPerlBioinfo;      # see Chapter 6
14    about this module
```



DNA 翻译成蛋白质 | 子程序

```
13 # Initialize variables
14 my $protein = '';
15
16 # Translate each three-base codon to an
17 # amino acid, and append to a protein
17 for(my $i=0; $i < (length($dna) - 2) ; $i
18 += 3) {
19     $protein .= codon2aa( substr($dna,$i,3));
20 }
21
22 return $protein;
22 }
```



易读 vs. 速度

- 使用 \$codon 中间变量：更加易读
- 不使用 \$codon 中间变量：避免字符串的复制，提高运行速度
- 当程序本身的易读性降低时，补充注释进行说明

use

- 可以在子程序中使用 use 载入模块
- 子程序的 use 可以和主程序的 use 存在冗余
- Perl 只会载入模块一次
- 子程序中载入模块有一定的优势（主程序中没有载入这些模块时）



易读 vs. 速度

- 使用 \$codon 中间变量：更加易读
- 不使用 \$codon 中间变量：避免字符串的复制，提高运行速度
- 当程序本身的易读性降低时，补充注释进行说明

use

- 可以在子程序中使用 use 载入模块
- 子程序的 use 可以和主程序的 use 存在冗余
- Perl 只会载入模块一次
- 子程序中载入模块有一定的优势（主程序中没有载入这些模块时）



教学提纲

1

引言

2

散列

3

数据结构和算法

4

遗传密码

- 密码子翻译成氨基酸
- 遗传密码的冗余性
- 使用散列表表示遗传密码

5

DNA 翻译成蛋白质

6

读取 FASTA 格式的 DNA

- 序列格式
- 读取 FASTA 文件的思路
- 读取 FASTA 文件的子程序
- 输出格式化的序列数据

- 读取 FASTA 文件的主程序
- DNA 翻译成蛋白质的主程序

7

阅读框

- 简介
- 翻译阅读框

8

回顾和总结

- 总结
- 思考题

9

知识拓展

- Perl 里面的复杂数据结构
- 数据共享
- 宽格式和长格式的数据
- 数据处理实例



教学提纲

1 引言

散列

数据结构和算法

遗传密码

- 密码子翻译成氨基酸
- 遗传密码的冗余性
- 使用散列表表示遗传密码

DNA 翻译成蛋白质

6 读取 FASTA 格式的 DNA

● 序列格式

- 读取 FASTA 文件的思路
- 读取 FASTA 文件的子程序
- 输出格式化的序列数据

- 读取 FASTA 文件的主程序
- DNA 翻译成蛋白质的主程序

7 阅读框

- 简介
- 翻译阅读框

8 回顾和总结

- 总结
- 思考题

9 知识拓展

- Perl 里面的复杂数据结构
- 数据共享
- 宽格式和长格式的数据
- 数据处理实例



读取 FASTA 格式的 DNA | 序列格式

- Plain sequence format
- FASTA format
- GenBank format
- EMBL format
- GCG format
- GCG-RSF (rich sequence format)
- IG format



读取 FASTA 格式的 DNA | 序列格式 | Plain

Plain sequence format

A sequence in plain format may contain only IUPAC characters and spaces (no numbers!).

Note: A file in plain sequence format may only contain **one** sequence, while most other formats accept several sequences in one file.

An example sequence in plain format is:

```
ACAAGATGCCATTGTCCCCGGCCTCTGCTGCTGCTCTCCGGGGCACGGCCACCGCTGCCCTGCC  
CCTGGAGGGTGGCCCCACCGGCCGAGACAGCGAGCATATGCAGGAAGCGGCAGGAATAAGGAAAAGCAGC  
CTCCTGACTTCTCGCTTGGTGGTTGAGTGGACCTCCCAGGCCAGTGCCGGCCCTCATAGGAGAGG  
AAGCTCGGGAGGTGCCAGGCAGGAAGGCGCACCCCCCAGCAATCCGCGGCCGGACAGAATGCC  
CTGCAGGAACCTCTTCTGAAAGACCTTCCTCCTGCAAATAAACCTCACCCATGAATGCTCACGCAAG  
TTTAATTACAGACCTGAA
```



读取 FASTA 格式的 DNA | 序列格式 | FASTA

FASTA format

A sequence file in FASTA format can contain several sequences.

Each sequence in FASTA format begins with a single-line description, followed by lines of sequence data. The description line must begin with a greater-than (">") symbol in the first column.

An example sequence in FASTA format is:

```
>AB000263 |acc=AB000263|descr=Homo sapiens mRNA for prepro cortistatin like peptide, complete cds.|len=368
ACAAGATGCCATTGTCCTCCGGCCTCTGCTGCTGCTCTCGGGGCCACGGCCACCGCTGCCCTGCC
CCTGGAGGGTGGCCCCACCGGCCAGACAGCGCATATGCAGGAAGCGGCAGGAATAAGGAAAGCAGC
CTCCTGACTTTCTCGTTGGTGGTTGAGTGGACCTCCCAGGCCAGTGCCGGGCCCTCATAGGAGAGG
AAGCTGGAGGTGGCCAGGCAGGAAGGCCACCCCCCAGCAATCCGCGCGCCGGGACAGAATGCC
CTGCAGGAACCTCTCTGGAAGACCTCTCCCTCTGCAAATAAACCTCACCATGAATGCTCACGCAAG
TTTAATTACAGACCTGAA
```



读取 FASTA 格式的 DNA | 序列格式 | GenBank

GenBank format

A sequence file in GenBank format can contain several sequences.

One sequence in GenBank format starts with a line containing the word LOCUS and a number of annotation lines. The start of the sequence is marked by a line containing "ORIGIN" and the end of the sequence is marked by two slashes ("//").

An example sequence in GenBank format is:

```
LOCUS      AB000263          368 bp      mRNA      linear      PRI 05-FEB-1999
DEFINITION Homo sapiens mRNA for prepro cortistatin like peptide, complete
cds.
ACCESSION  AB000263
ORIGIN
 1 acaagatgcc attgtccccc ggcctcctgc tgctgctgct ctccggggcc acggccaccc
 61 ctgcctgcc cctggagggt ggccccaccc gccgagacag cgagcatatg caggaagcg
121 caggaataag gaaaagcagc ctcctgactt tcctcgctt gttggtttag tggacctccc
181 aggccagtgc cggccccctc ataggagagg aagctcggga ggtggccagg cggcaggaag
241 ggcacccccc ccagcaatcc ggcgcggg acagaatgcc ctgcaggaac ttcttctgga
301 agaccttctc ctccctgcaaa taaaacctca cccatgaatg ctcacgcaag tttattaca
361 gacctgaa
//
```



读取 FASTA 格式的 DNA | 序列格式 | EMBL

EMBL format

A sequence file in EMBL format can contain several sequences.

One sequence entry starts with an identifier line ("ID"), followed by further annotation lines. The start of the sequence is marked by a line starting with "SQ" and the end of the sequence is marked by two slashes ("//").

An example sequence in EMBL format is:

```
ID  AB000263 standard; RNA; PRI; 368 BP.  
XX  
AC  AB000263;  
XX  
DE  Homo sapiens mRNA for prepro cortistatin like peptide, complete cds.  
XX  
SQ  Sequence 368 BP;  
     acaagatgcc attgtccccc ggccctcctgc tgctgctgct ctccggggcc acggccaccc 60  
     ctgccctgcc cctggagggt ggccccaccc gccgagacag cgagcatatg caggaagccg 120  
     caggaataag gaaaagcagc ctccctgactt tcctcgcttg gtggtttgag tggacctccc 180  
     aggccagtgc cggccccctc ataggagagg aagctcggga ggtggccagg cggcaggaag 240  
     gcgcacccccc ccagcaatcc gcgcgccggg acagaatgcc ctgcaggaac ttcttctgga 300  
     agaccttctc ctccctgcaaa taaaacctca cccatgaatg ctcacgcaag ttaattaca 360  
     gacctgaa 368  
//
```



读取 FASTA 格式的 DNA | 序列格式 | GCG

GCG format

A sequence file in GCG format contains exactly one sequence, begins with annotation lines and the start of the sequence is marked by a line ending with two dot ("..") characters. This line also contains the sequence identifier, the sequence length and a checksum. This format should only be used if the file was created with the GCG package.

An example sequence in GCG format is:

```
ID AB000263 standard; RNA; PRI; 368 BP.
XX
AC AB000263;
XX
DE Homo sapiens mRNA for prepro cortistatin like peptide, complete cds.
XX
SQ Sequence 368 BP;
AB000263 Length: 368 Check: 4514 ..
   1 acaagatgcc attgtccccc ggccctcctgc tgctgctgct ctccggggcc acggccaccg
    61 ctgccctgcc cctggagggt gggcccaccc gccgagacag cgagcatatg caggaagcgg
   121 caggaataag gaaaagcagc ctcctgactt tcctcgcttg gtggtttag tggacctccc
   181 aggccagtgc cggggccctc ataggagagg aagctcggga ggtggccagg cggcaggaag
   241 ggcacccccc ccagcaatcc gcgcgcggg acagaatgcc ctgcaggaac ttcttctgaa
   301 agaccttctc ctcctgaaaa taaaacctca cccatgaatg ctcacgcaga tttaattaca
   361 gacctqaa
```



读取 FASTA 格式的 DNA | 序列格式 | IG

IG format

A sequence file in IG format can contain several sequences, each consisting of a number of comment lines that must begin with a semicolon (";"), a line with the sequence name (it may not contain spaces!) and the sequence itself terminated with the termination character '1' for linear or '2' for circular sequences.

An example sequence in IG format is:

```
; comment
; comment
AB000263
ACAAGATGCCATTGTCCCCCGGCCCTGCTGCTGCTCTCCGGGGCACGGCCACCGCTGCCCTGCC
CCTGGAGGGTGGCCCCACCGGCCGAGACAGCGAGCATATGCAGGAAGCGGCAGGAATAAGGAAAAGCAGC
CTCCTGACTTCCTCGCTTGGTGGTTGAGTGGACCTCCCAGGCCAGTGCCGGCCCCCTCATAGGAGAGG
AAGCTCAGGGAGGTGGCCAGGCAGGAAGGCGCACCCCCCAGCAATCCGCGCGCCGGGACAGAATGCC
CTGCAGGAACCTCTTCTGAAAGACCTCTCCTGCAAATAAACCTCACCCATGAATGCTCACGCAAG
TTAACATTACAGACCTGAA1
```



教学提纲

1

引言

2

散列

3

数据结构和算法

4

遗传密码

- 密码子翻译成氨基酸
- 遗传密码的冗余性
- 使用散列表表示遗传密码

5

DNA 翻译成蛋白质

6

读取 FASTA 格式的 DNA

- 序列格式
- 读取 FASTA 文件的思路
- 读取 FASTA 文件的子程序
- 输出格式化的序列数据

- 读取 FASTA 文件的主程序
- DNA 翻译成蛋白质的主程序

7

阅读框

- 简介
- 翻译阅读框

8

回顾和总结

- 总结
- 思考题

9

知识拓展

- Perl 里面的复杂数据结构
- 数据共享
- 宽格式和长格式的数据
- 数据处理实例



读取文件的两种策略

- ① 从打开的文件中一次读入一行，边读入、边处理
 - 大文件
 - 寻找小的信息片段
- ② 一次性把整个文件都读入到数组中，然后对数组进行操作
 - 中、小文件
 - 容易遍历整个数据进行操作



要求

- 输入：FASTA 文件的文件名参数
- 返回：FASTA 文件中的序列数据

思路

- 一个子程序：打开、读取 FASTA 文件并提取序列数据
- 两个子程序
 - ① 第一个子程序：打开并读取文件
 - ② 第二个子程序：提取序列数据



要求

- 输入：FASTA 文件的文件名参数
- 返回：FASTA 文件中的序列数据

思路

- 一个子程序：打开、读取 FASTA 文件并提取序列数据
- 两个子程序
 - ① 第一个子程序：打开并读取文件
 - ② 第二个子程序：提取序列数据



```
1 subroutine get data from a file
2
3     argument = filename
4
5     open file
6         if can't open, print error message and
7             exit
8
9     read in data and
10
11    return @data
12 }
```



读取 FASTA 文件 | 两个子程序 | 伪代码 | 提取序列

```
1 Subroutine extract sequence data from fasta file
2
3     argument = array of file data in fasta format
4
5         Discard all header lines
6             (and blank and comment lines for good measure)
7             If first character of first line is >, discard
8                 it
9
10            Read in the rest of the file, join in a scalar,
11                edit out nonsequence data
12
13    return sequence
14 }
```



读取 FASTA 文件 | 读取失败

如果文件不能被读取，可能的处理方法：

- 退出程序：比较极端
- 向用户询问文件名：人性化，不利于自动化
- 使用默认的文件：人性化，自动化
- 返回 false（空数组）：容易和空文件相混淆
- 其他处理方法：返回 undefined



教学提纲

1

引言

2

散列

3

数据结构和算法

4

遗传密码

- 密码子翻译成氨基酸
- 遗传密码的冗余性
- 使用散列表表示遗传密码

5

DNA 翻译成蛋白质

6

读取 FASTA 格式的 DNA

- 序列格式
- 读取 FASTA 文件的思路
- 读取 FASTA 文件的子程序
- 输出格式化的序列数据

- 读取 FASTA 文件的主程序
- DNA 翻译成蛋白质的主程序

7

阅读框

- 简介
- 翻译阅读框

8

回顾和总结

- 总结
- 思考题

9

知识拓展

- Perl 里面的复杂数据结构
- 数据共享
- 宽格式和长格式的数据
- 数据处理实例



读取 FASTA 文件 | 子程序

```
1 # get_file_data
2 # A subroutine to get data from a file given its
3 # filename
4 sub get_file_data {
5     my ($filename) = @_;
6     use strict;
7     use warnings;
8     # Initialize variables
9     my @filedata = ();
10    unless( open(GET_FILE_DATA, $filename) ) {
11        print STDERR "Cannot open file \"$filename\"\n\n";
12        exit;
13    }
14    @filedata = <GET_FILE_DATA>;
15    close GET_FILE_DATA;
16    return @filedata;
17 }
```



读取 FASTA 文件 | 子程序

```
31 # extract_sequence_from_fasta_data: A subroutine to
32 sub extract_sequence_from_fasta_data {
33     my (@fasta_file_data) = @_;
34     use strict; use warnings;
35     my $sequence = ''; # Declare and initialize variables
36     foreach my $line (@fasta_file_data) {
37         # discard blank line
38         if ($line =~ /^$/) { next;
39         # discard comment line
40     } elsif($line =~ /^#/){ next;
41         # discard fasta header line
42     } elsif($line =~ />/) { next;
43         # keep line, add to sequence string
44     } else { $sequence .= $line; }
45 }
46 # remove non-sequence data (in this case, whitespace)
47 # from $sequence string
48 $sequence =~ s/\s//g; return $sequence;
```



读取 FASTA 文件 | 子程序 | 说明

```
1 # 使用my声明循环中使用的变量
2 foreach my $line (@fasta_file_data) {
3
4 # 匹配只有/没有空白的空行
5 if ($line =~ /^$|^$/ ) {
6
7 # 匹配注释行
8 } elsif($line =~ /^#/ ) {
9
10 # 匹配FASTA标题行
11 } elsif($line =~ /^>/ ) {
12
13 # 删除空白字符 (含换行符)
14 $sequence =~ s/\s//g;
```



教学提纲

1

引言

2

散列

3

数据结构和算法

4

遗传密码

- 密码子翻译成氨基酸
- 遗传密码的冗余性
- 使用散列表表示遗传密码

5

DNA 翻译成蛋白质

6

读取 FASTA 格式的 DNA

- 序列格式
- 读取 FASTA 文件的思路
- 读取 FASTA 文件的子程序
- 输出格式化的序列数据

- 读取 FASTA 文件的主程序
- DNA 翻译成蛋白质的主程序

7

阅读框

- 简介
- 翻译阅读框

8

回顾和总结

- 总结
- 思考题

9

知识拓展

- Perl 里面的复杂数据结构
- 数据共享
- 宽格式和长格式的数据
- 数据处理实例



格式化序列数据

- 目的：把序列数据格式化成适当长度
- 输入：序列和行的长度两个参数
- 输出：按指定要求格式化后的序列数据



格式化序列数据 | 子程序

```
1 # print_sequence
2 # A subroutine to format and print sequence
3 # data
4
5 sub print_sequence {
6     my($sequence, $length) = @_;
7     use strict;
8     use warnings;
9     # Print sequence in lines of $length
10    for ( my $pos = 0 ; $pos < length($sequence)
11        ) ; $pos += $length ) {
12        print substr($sequence, $pos, $length), "
13        \n";
14    }
15 }
```



教学提纲

- 1 引言
- 2 散列
- 3 数据结构和算法
- 4 遗传密码
 - 密码子翻译成氨基酸
 - 遗传密码的冗余性
 - 使用散列表表示遗传密码
- 5 DNA 翻译成蛋白质
- 6 读取 FASTA 格式的 DNA
 - 序列格式
 - 读取 FASTA 文件的思路
 - 读取 FASTA 文件的子程序
 - 输出格式化的序列数据

- 7 阅读框
 - 简介
 - 翻译阅读框
- 8 回顾和总结
 - 总结
 - 思考题
- 9 知识拓展
 - Perl 里面的复杂数据结构
 - 数据共享
 - 宽格式和长格式的数据
 - 数据处理实例



读取 FASTA 文件 | 程序 8.2.1

```
1 #!/usr/bin/perl -w
2 # Example 8-2 Read a fasta file and extract
   the sequence data
3
4 use strict;
5 use warnings;
6 use Bio::Perl::Bioinfo;          # see Chapter 6
   about this module
7
8 # Declare and initialize variables
9 my @file_data = ();
10 my $dna = '';
```



读取 FASTA 文件 | 程序 8.2.1

```
12 # Read in the contents of the file "sample.dna"
13 @file_data = get_file_data("sample.dna");
14
15 # Extract the sequence data from the contents
16 # of the file "sample.dna"
16 $dna = extract_sequence_from_fasta_data(
17     @file_data);
18
18 # Print the sequence in lines 25 characters
19 # long
19 print_sequence($dna, 25);
20
21 exit;
```



读取 FASTA 文件 | 程序 8.2 | 输出

```
1 agatggcgccgtgaggggtcttgg  
2 gggctctaggccggccacctactgg  
3 tttgcagcggagacgcacgcacatgggg  
4 cctgcgcataataggagtacgcgtgcct  
5 gggaggcgtgactagaagcggaagt  
6 ...  
7 ccaacagcagccacagccatcacag  
8 aagtttagggcgcatccgtgaagatg  
9 agggggcagtggcggtcatcaacagt  
10 caaggagcctcctgaggctacagcc  
11 acacctgagccactctcagatgagg  
12 accta
```



教学提纲

1

引言

2

散列

3

数据结构和算法

4

遗传密码

- 密码子翻译成氨基酸
- 遗传密码的冗余性
- 使用散列表表示遗传密码

5

DNA 翻译成蛋白质

6

读取 FASTA 格式的 DNA

- 序列格式
- 读取 FASTA 文件的思路
- 读取 FASTA 文件的子程序
- 输出格式化的序列数据

- 读取 FASTA 文件的主程序
- DNA 翻译成蛋白质的主程序

7

阅读框

- 简介
- 翻译阅读框

8

回顾和总结

- 总结
- 思考题

9

知识拓展

- Perl 里面的复杂数据结构
- 数据共享
- 宽格式和长格式的数据
- 数据处理实例



DNA 翻译成蛋白质 | 程序 8.3.1

```
1 #!/usr/bin/perl -w
2 # Example 8-3 Read a fasta file and extract
3 # the DNA sequence data
4 # Translate it to protein and print it out in
5 # 25-character-long lines
6
7 use strict;
8 use warnings;
9 use BeginPerlBioinfo;          # see Chapter 6
10 # about this module
11
12 # Initialize variables
13 my @file_data = ();
14 my $dna = '';
15 my $protein = '';
```



DNA 翻译成蛋白质 | 程序 8.3.1

```
13 # Read in the contents of the file "sample.dna"
14 @file_data = get_file_data("sample.dna");
15
16 # Extract the sequence data from the contents
17 # of the file "sample.dna"
17 $dna = extract_sequence_from_fasta_data(
18     @file_data);
19
19 # Translate the DNA to protein
20 $protein = dna2peptide($dna);
21
22 # Print the sequence in lines 25 characters
22 # long
23 print_sequence($protein, 25);
24
```



DNA 翻译成蛋白质 | 程序 8.3 | 输出

```
1 RWRR_GVLGALGRPPTGLQRRRRMG  
2 PAQ_EYAAWEA_LEAEVVVGAFATA  
3 WDAAEWSVQVRGSLAGVVRECAGSG  
4 DMEGDGSDPEPPDAGEDSKSENGEN  
5 APIYCICRKPDINCFMIGCDNCNEW  
6 FHGDCIRITEKMAKAIREWYCRECR  
7 EKDPKLEIRYRHKKSRERDGNERDS  
8 SEPRDEGGGRKRPVPDPDLQRRAKS  
9 GTGVGAMLARGSASP HKSSPQPLVA  
10 TPSQHHQQQQQQIKRSARMCGECEA  
11 CRRTEDCGHCDFCRDMKKFGGPNKI  
12 RQKCRLRQCQLRARESYKYFPSSLS  
13 PVT PSESLPRP RPLPTQQQPQPSQ  
14 KLGRIREDEGAVASSTVKEPPEATA  
15 TPEPLSDEDL
```



教学提纲

1

引言

2

散列

3

数据结构和算法

4

遗传密码

- 密码子翻译成氨基酸
- 遗传密码的冗余性
- 使用散列表表示遗传密码

5

DNA 翻译成蛋白质

6

读取 FASTA 格式的 DNA

- 序列格式
- 读取 FASTA 文件的思路
- 读取 FASTA 文件的子程序
- 输出格式化的序列数据

- 读取 FASTA 文件的主程序
- DNA 翻译成蛋白质的主程序

7

阅读框

- 简介
- 翻译阅读框

8

回顾和总结

- 总结
- 思考题

9

知识拓展

- Perl 里面的复杂数据结构
- 数据共享
- 宽格式和长格式的数据
- 数据处理实例



教学提纲

1

引言

2

散列

3

数据结构和算法

4

遗传密码

- 密码子翻译成氨基酸
- 遗传密码的冗余性
- 使用散列表表示遗传密码

5

DNA 翻译成蛋白质

6

读取 FASTA 格式的 DNA

- 序列格式
- 读取 FASTA 文件的思路
- 读取 FASTA 文件的子程序
- 输出格式化的序列数据

- 读取 FASTA 文件的主程序
- DNA 翻译成蛋白质的主程序

7

阅读框

- 简介
- 翻译阅读框

8

回顾和总结

- 总结
- 思考题

9

知识拓展

- Perl 里面的复杂数据结构
- 数据共享
- 宽格式和长格式的数据
- 数据处理实例



阅读框架 (reading frame) 是指 RNA 或 DNA 中，一组连续且不重复的 3 核苷酸密码子。每一条 mRNA 共有 3 种可能的阅读框架，而双股的 DNA 则每股各 3 种，共 6 种阅读框架。

同一序列具有多种阅读框架的现象使重叠基因 (overlapping genes) 可能存在，已知常见于某些细菌与病毒。至于人类细胞中则较少见。



阅读框 | 阅读框架 | 6 种

+1 ATGGTCGTACCCGAATTGAAATCA
start M V V P V L K S
F +2 ATGGTCGTACCCGAATTGAAATCA
W S Y P N stop
+3 ATGGTCGTACCCGAATTGAAATCA
G R T R I E I

Reading Orientation →

-1 TGATTTCAATTCTGGGTACGACCAT
stop
R -2 TGATTTCAATTCTGGGTACGACCAT
D F N S G T T
-3 TGATTTCAATTCTGGGTACGACCAT
I S I R V R P



开放阅读框（open reading frame, ORF）是指在给定的阅读框架中，不包含终止密码子的一串序列。这段序列是生物个体的基因组中，可能作为蛋白质编码序列的部分。基因中的 ORF 包含并位于起始编码与终止编码之间。

由于一段 DNA 或 RNA 序列有多种不同读取方式，因此可能同时存在许多不同的开放阅读框架。

开放阅读框是判断 DNA 序列中存在基因的一个重要依据；基因识别程序需要进行开放阅读框的分析。



基本思想

- 一切高大上的问题/任务都是纸老虎

解决思路

- 四处寻找已经写好的、可以完成该任务的（子）程序/工具
- 充分利用自己收集的子程序库
 - 有哪些相关的子程序可以直接使用
 - 有哪些相关的子程序需要进行修改/扩展
 - 还缺少哪些子程序需要编写添加



基本思想

- 一切高大上的问题/任务都是纸老虎

解决思路

- 四处寻找已经写好的、可以完成该任务的（子）程序/工具
- 充分利用自己收集的子程序库
 - 有哪些相关的子程序可以直接使用
 - 有哪些相关的子程序需要进行修改/扩展
 - 还缺少哪些子程序需要编写添加



已有的子程序

- get_file_data: 打开读取文件
- extract_sequence_from_fasta_data: 提取 FASTA 文件中的序列
- dna2peptide: 把 DNA 翻译成多肽
 - codon2aa: 把密码子翻译成氨基酸
- print_sequence: 输出格式化的序列

尚缺的子程序

- revcom: 计算反向互补序列
- translate_frame: 翻译 DNA 的指定区间



已有的子程序

- get_file_data: 打开读取文件
- extract_sequence_from_fasta_data: 提取 FASTA 文件中的序列
- dna2peptide: 把 DNA 翻译成多肽
 - codon2aa: 把密码子翻译成氨基酸
- print_sequence: 输出格式化的序列

尚缺的子程序

- revcom: 计算反向互补序列
- translate_frame: 翻译 DNA 的指定区间



教学提纲

1

引言

2

散列

3

数据结构和算法

4

遗传密码

- 密码子翻译成氨基酸
- 遗传密码的冗余性
- 使用散列表表示遗传密码

5

DNA 翻译成蛋白质

6

读取 FASTA 格式的 DNA

- 序列格式
- 读取 FASTA 文件的思路
- 读取 FASTA 文件的子程序
- 输出格式化的序列数据

- 读取 FASTA 文件的主程序
- DNA 翻译成蛋白质的主程序

7

阅读框

- 简介
- 翻译阅读框

8

回顾和总结

- 总结
- 思考题

9

知识拓展

- Perl 里面的复杂数据结构
- 数据共享
- 宽格式和长格式的数据
- 数据处理实例



阅读框 | 翻译 | 反向互补 | 子程序

```
1 # revcom
2 # A subroutine to compute the reverse
   complement of DNA sequence
3 sub revcom {
4     my($dna) = @_;
5
6     # First reverse the sequence
7     my($revcom) = reverse($dna);
8
9     # Next, complement the sequence, dealing
10    with upper and lower case
11    # A->T, T->A, C->G, G->C
12    $revcom =~ tr/ACGTacgt/TGCATgca/;
13
14    return $revcom;
}
```



```
1 Given DNA sequence
2
3 subroutine translate_frame ( DNA, start, end )
4
5     return dna2peptide( substr( DNA, start, end - start +
6         1 ) )
7 }
```

```
1 Given DNA sequence
2
3 subroutine translate_frame ( DNA, start, end )
4
5     # start and end are numbering the sequence from 1 to
6     # length
7     return dna2peptide( substr( DNA, start - 1, end -
8         start + 1 ) )
9 }
```



```
1 Given DNA sequence
2
3 subroutine translate_frame ( DNA, start, end )
4
5     return dna2peptide( substr( DNA, start, end - start +
6         1 ) )  
7 }
```

```
1 Given DNA sequence
2
3 subroutine translate_frame ( DNA, start, end )
4
5     # start and end are numbering the sequence from 1 to
6     # length
7     return dna2peptide( substr( DNA, start - 1, end -
8         start + 1 ) )  
9 }
```



两种索引

- Perl 的处理方式：从 0 开始
- 生物学家的处理方式：从 1 开始

推荐策略

- 面向程序：以 0 起始的索引方式（计算简便）
- 面向用户：以 1 起始的索引方式（人性化）



两种索引

- Perl 的处理方式：从 0 开始
- 生物学家的处理方式：从 1 开始

推荐策略

- 面向程序：以 0 起始的索引方式（计算简便）
- 面向用户：以 1 起始的索引方式（人性化）



阅读框 | 翻译 | 指定区间 | 子程序

```
1 # translate_frame
2 # A subroutine to translate a frame of DNA
3
4 sub translate_frame {
5     my ($seq, $start, $end) = @_;
6     #my $protein;
7
8     # To make the subroutine easier to use, you won't need
9     # to specify the end point--it will just go to the end
10    # of the sequence by default.
11    unless ($end) {
12        $end = length($seq);
13    }
14
15    # Finally, calculate and return the translation
16    return dna2peptide ( substr ( $seq, $start - 1, $end -
$start + 1 ) );
17}
```



阅读框 | 翻译 | 程序 8.4.1

```
1 #!/usr/bin/perl -w
2 # Example 8-4      Translate a DNA sequence in
3 #                  all six reading frames
4
5 use strict;
6 use warnings;
7 use Bio::Perl::Bioinfo;          # see Chapter 6
8 # Initialize variables
9 my @file_data = ();
10 my $dna = '';
11 my $revcom = '';
12 my $protein = '';
```



阅读框 | 翻译 | 程序 8.4.2

```
14 # Read in the contents of the file "sample.dna"
15 @file_data = get_file_data("sample.dna");
16
17 # Extract the sequence data from the contents of
18 # the file "sample.dna"
19 $dna = extract_sequence_from_fasta_data(@file_data
20 );
21
22 # Translate the DNA to protein in six reading
23 # frames
24 # and print the protein in lines 70 characters
25 # long
26 print "\n -----Reading Frame 1-----\n\n";
27 $protein = translate_frame($dna, 1);
28 print_sequence($protein, 70);
```



阅读框 | 翻译 | 程序 8.4.3

```
26 print "\n -----Reading Frame 2-----\n\n"
      ;
27 $protein = translate_frame($dna, 2);
28 print_sequence($protein, 70);
29
30 print "\n -----Reading Frame 3-----\n\n"
      ;
31 $protein = translate_frame($dna, 3);
32 print_sequence($protein, 70);
```



阅读框 | 翻译 | 程序 8.4.4

```
34 # Calculate reverse complement
35 $revcom = revcom($dna);
36
37 print "\n -----Reading Frame 4-----\n\n";
38 $protein = translate_frame($revcom, 1);
39 print_sequence($protein, 70);
40
41 print "\n -----Reading Frame 5-----\n\n";
42 $protein = translate_frame($revcom, 2);
43 print_sequence($protein, 70);
44
45 print "\n -----Reading Frame 6-----\n\n";
46 $protein = translate_frame($revcom, 3);
47 print_sequence($protein, 70);
48
49 exit;
```



阅读框 | 翻译 | 程序 8.4 | 输出

```
1 -----Reading Frame 1-----
2
3 RWRR_GVLGALGRPPTGLQRRRRMGPAA_EYAAWEA_LEAEVVVGAFATAWDAAEWSVQVRGSLAGVVRE
4 CAGSGDMEGDGSDEPPDAGEDSKSENGENAPIYCICRKPDINCFMIGCDNCNEWFHGDCIRITEKMAKA
5 IREWYCRECREKDPKLEIIRYRHKKSRERDGNERDSSEPRDEGGGRKRPVPDPDLQRRAAGSGTGVGAMLAR
6 GSASPHKSSPQPLVATPSQHHQQQQQQIKRSARMCECEACRRTEDCGHCDFCRDMKKFGGPNKIRQKCR
7 LRCQCLRARESYKFPSLSPVTPSESPLPRRPLPTQQQPQPSQKLGRIREDEGAVASSTVKEPPEATA
8 TPEPLSDEDL
9
10 ...
11
12 -----Reading Frame 6-----
13
14 GPHLRVAQVWL_PQEAP_LLMTPLPPHLHGCALTSVMAVAAGWAVAGGALAGTLRASLVRARKGSTCTI
15 PGPAAAGTGAAGTSAGSCWGPRTSSCPDRNHSDDHSPQCADMPTHHTCGLTV_SAAAAAGDAGWVWPPRAA
16 ERICGAKQSPEQAWPQPLSLTPGAAGLDQGQASCALHPHPGAHCPCAHCHPAPVTSCADSES LAWGLSL
17 CTPDSTTPGWPWPSSQ_SGCSPHGTTHCSCHTRS_SS_CPVCGRCSRWAHSRSHSRTCCPFRHLEALGLNH
18 LPPYLRSSRRTSPRPPPATREPAQTTRRRPRLQRRPQLLVTPPRQRTPIAQAPCVVSAANQ_VAGLE
19 PPRPLSAAI
```



教学提纲

1

引言

2

散列

3

数据结构和算法

4

遗传密码

- 密码子翻译成氨基酸
- 遗传密码的冗余性
- 使用散列表表示遗传密码

5

DNA 翻译成蛋白质

6

读取 FASTA 格式的 DNA

- 序列格式
- 读取 FASTA 文件的思路
- 读取 FASTA 文件的子程序
- 输出格式化的序列数据

- 读取 FASTA 文件的主程序
- DNA 翻译成蛋白质的主程序

7

阅读框

- 简介
- 翻译阅读框

8

回顾和总结

- 总结
- 思考题

9

知识拓展

- Perl 里面的复杂数据结构
- 数据共享
- 宽格式和长格式的数据
- 数据处理实例



教学提纲

1 引言

散列

数据结构和算法

遗传密码

- 密码子翻译成氨基酸
- 遗传密码的冗余性
- 使用散列表表示遗传密码

5 DNA 翻译成蛋白质

DNA 翻译成蛋白质

6 读取 FASTA 格式的 DNA

- 序列格式
- 读取 FASTA 文件的思路
- 读取 FASTA 文件的子程序
- 输出格式化的序列数据

- 读取 FASTA 文件的主程序
- DNA 翻译成蛋白质的主程序

7

阅读框

- 简介
- 翻译阅读框

8

回顾和总结

- 总结
- 思考题

9

知识拓展

- Perl 里面的复杂数据结构
- 数据共享
- 宽格式和长格式的数据
- 数据处理实例



知识点

- 散列：键，值，初始化，检查定义/存在，排序，……
- Perl 中的数据类型：标量变量，数组，散列
- 正则表达式：元字符，字符集，修饰符，择一匹配
- 序列格式：plain, FASTA, GenBank, EMBL, ……
- Perl 读取文件：两种策略，读取失败的处理方法
- 阅读框：6 种框架，开放阅读框
- 其他：数据结构，比较排序，文件句柄，循环，子字符串提取，索引，……

技能

- 熟练使用 Perl 中的散列
- 熟练使用 Perl 读取文件
- 能编写把 DNA 翻译成蛋白质的 Perl 程序

知识点

- 散列：键，值，初始化，检查定义/存在，排序，……
- Perl 中的数据类型：标量变量，数组，散列
- 正则表达式：元字符，字符集，修饰符，择一匹配
- 序列格式：plain, FASTA, GenBank, EMBL, ……
- Perl 读取文件：两种策略，读取失败的处理方法
- 阅读框：6 种框架，开放阅读框
- 其他：数据结构，比较排序，文件句柄，循环，子字符串提取，索引，……

技能

- 熟练使用 Perl 中的散列
- 熟练使用 Perl 读取文件
- 能编写把 DNA 翻译成蛋白质的 Perl 程序

教学提纲

1

引言

2

散列

3

数据结构和算法

4

遗传密码

- 密码子翻译成氨基酸
- 遗传密码的冗余性
- 使用散列表表示遗传密码

5

DNA 翻译成蛋白质

6

读取 FASTA 格式的 DNA

- 序列格式
- 读取 FASTA 文件的思路
- 读取 FASTA 文件的子程序
- 输出格式化的序列数据

- 读取 FASTA 文件的主程序
- DNA 翻译成蛋白质的主程序

7

阅读框

- 简介
- 翻译阅读框

8

回顾和总结

- 总结
- 思考题

9

知识拓展

- Perl 里面的复杂数据结构
- 数据共享
- 宽格式和长格式的数据
- 数据处理实例



- ① 比较 Perl 的三种常见数据类型。
- ② 如何获取散列中的所有键/值？
- ③ 如何对字符串/数字进行正序/逆序的排序？
- ④ 举例说明不同任务下数据结构的选择。
- ⑤ 如何使用正则表达式表征密码子？
- ⑥ 列举并举例说明常见的序列格式。
- ⑦ 比较读取文件的两种策略。
- ⑧ 举例说明 6 种阅读框。
- ⑨ 比较两种不同的索引方式。
- ⑩ 举例说明子程序的使用。



下节预告

计算机

回顾正则表达式的相关知识：元字符，字符集，量词，修饰符，……

生物学

回顾限制性核酸内切酶的相关知识：定义，分类，特点，……



教学提纲

1

引言

2

散列

3

数据结构和算法

4

遗传密码

- 密码子翻译成氨基酸
- 遗传密码的冗余性
- 使用散列表表示遗传密码

5

DNA 翻译成蛋白质

6

读取 FASTA 格式的 DNA

- 序列格式
- 读取 FASTA 文件的思路
- 读取 FASTA 文件的子程序
- 输出格式化的序列数据

- 读取 FASTA 文件的主程序
- DNA 翻译成蛋白质的主程序

7

阅读框

- 简介
- 翻译阅读框

8

回顾和总结

- 总结
- 思考题

9

知识拓展

- Perl 里面的复杂数据结构
- 数据共享
- 宽格式和长格式的数据
- 数据处理实例



教学提纲

1 引言

散列

数据结构和算法

遗传密码

- 密码子翻译成氨基酸
- 遗传密码的冗余性
- 使用散列表表示遗传密码

5 DNA 翻译成蛋白质

6 读取 FASTA 格式的 DNA

- 序列格式
- 读取 FASTA 文件的思路
- 读取 FASTA 文件的子程序
- 输出格式化的序列数据

- 读取 FASTA 文件的主程序
- DNA 翻译成蛋白质的主程序

7

阅读框

- 简介
- 翻译阅读框

8

回顾和总结

- 总结
- 思考题

9

知识拓展

- Perl 里面的复杂数据结构
- 数据共享
- 宽格式和长格式的数据
- 数据处理实例



- arrays of arrays, 数组的数组
- hashes of arrays, 散列的元素为数组
- arrays of hashes, 数组的元素为散列
- hashes of hashes, 散列的元素为散列
- more elaborate constructs, 更复杂的结构



```
1 @AoA = (
2     [ "fred", "barney" ],
3     [ "george", "jane", "elroy" ],
4     [ "homer", "marge", "bart" ],
5 );
```



知识拓展 | 复杂数据结构 | 数组的数组 | 生成

```
1 # reading from file
2 while ( <> ) {
3     push @AoA, [ split ];
4 }
5 # calling a function
6 for $i ( 1 .. 10 ) {
7     $AoA[$i] = [ somefunc($i) ];
8 }
9 # using temp vars
10 for $i ( 1 .. 10 ) {
11     @tmp = somefunc($i);
12     $AoA[$i] = [ @tmp ];
13 }
14 # add to an existing row
15 push @{ $AoA[0] }, "wilma", "betty";
```



```
1 # one element  
2 $AoA[0][0] = "Fred";  
3  
4 # another element  
5 $AoA[1][1] =~ s/(\w)/\u$1/;
```



知识拓展 | 复杂数据结构 | 数组的数组 | 访问

```
1 # print the whole thing with refs
2 for $aref ( @AoA ) {
3     print "\t [ @{$aref } ],\n";
4 }
5 # print the whole thing with indices
6 for $i ( 0 .. $#AoA ) {
7     print "\t [ @{$AoA[$i]} ],\n";
8 }
9 # print the whole thing one at a time
10 for $i ( 0 .. $#AoA ) {
11     for $j ( 0 .. ${#$AoA[$i]} ) {
12         print "element $i $j is $AoA[$i][$j]\n";
13     }
14 }
```



```
1 %HoA = (
2   flintstones => [ "fred", "barney" ],
3   jetsons       => [ "george", "jane", "elroy"
4   ],
5   simpsons      => [ "homer", "marge", "bart"
6   ],
7 );
```



知识拓展 | 复杂数据结构 | 散列的元素为数组 | 生成

```
1 # reading from file
2 # flintstones: fred barney wilma dino
3 while ( <> ) {
4     next unless s/^(.*):\s*/;;
5     $HoA{$1} = [ split ];
6 }
7
8 # reading from file; more temps
9 # flintstones: fred barney wilma dino
10 while ( $line = <> ) {
11     ($who, $rest) = split /:\s*/, $line, 2;
12     @fields = split ' ', $rest;
13     $HoA{$who} = [ @fields ];
14 }
```



知识拓展 | 复杂数据结构 | 散列的元素为数组 | 生成

```
1 # calling a function that returns a list
2 for $group ( "simpsons", "jetsons", "
3   flintstones" ) {
4   $HoA{$group} = [ get_family($group) ];
5 }
6 # likewise, but using temps
7 for $group ( "simpsons", "jetsons", "
8   flintstones" ) {
9   @members = get_family($group);
10  $HoA{$group} = [ @members ];
11 }
12 # append new members to an existing family
13 push @{$HoA{"flintstones"}}, "wilma", "
14   betty";
```



知识拓展 | 复杂数据结构 | 散列的元素为数组 | 访问

```
1 # one element
2 $HoA{flintstones}[0] = "Fred";
3 # another element
4 $HoA{simpsons}[1] =~ s/(\w)/\u$1/;
5 # print the whole thing
6 foreach $family ( keys %HoA ) {
7     print "$family: @{$HoA{$family}} \n";
8 }
9 # print the whole thing with indices
10 foreach $family ( keys %HoA ) {
11     print "family: ";
12     foreach $i ( 0 .. $#{$HoA{$family}} ) {
13         print " $i = $HoA{$family}[$i]";
14     }
15     print "\n";
16 }
```



```
1 # print the whole thing sorted by number of
  members
2 foreach $family ( sort { @{$HoA{$b}} <=> @{$
  HoA{$a}} } keys %HoA ) {
3     print "$family: @{$HoA{$family}}\n";
4 }
5
6 # print the whole thing sorted by number of
  members and name
7 foreach $family ( sort { @{$HoA{$b}} <=> @{$
  HoA{$a}} || $a cmp $b } keys %HoA ) {
8     print "$family: ", join( ", ", sort @{$
  HoA{$family}} ), "\n";
9 }
```



知识拓展 | 复杂数据结构 | 数组的元素为散列 | 声明

```
1 @AoH = (
2 {
3     Lead    => "fred",
4     Friend  => "barney",
5 },
6 {
7     Lead    => "george",
8     Wife    => "jane",
9     Son     => "elroy",
10 },
11 {
12     Lead    => "homer",
13     Wife    => "marge",
14     Son     => "bart",
15 },
16 );
```



知识拓展 | 复杂数据结构 | 数组的元素为散列 | 生成

```
1 # reading from file
2 # format: LEAD=fred FRIEND=barney
3 while ( <> ) {
4     $rec = {};
5     for $field ( split ) {
6         ($key, $value) = split /=/, $field;
7         $rec->{$key} = $value;
8     }
9     push @AoH, $rec;
10 }
11
12 # no temp
13 while ( <> ) {
14     push @AoH, { split /[\s+=]/ };
15 }
```



```
1 # calling a function that returns a key/
  value pair list, like
2 # "lead","fred","daughter","pebbles"
3 while ( %fields = getnextpairset() ) {
4     push @AoH, { %fields };
5 }
6
7 # likewise, but using no temp vars
8 while (<>) {
9     push @AoH, { parsepairs($_) };
10 }
11
12 # add key/value to an element
13 $AoH[0]{pet} = "dino";
14 $AoH[2]{pet} = "santa's little helper";
```



```
1 # one element
2 $AoH[0]{lead} = "fred";
3
4 # another element
5 $AoH[1]{lead} =~ s/(\w)/\u$1/;
```



知识拓展 | 复杂数据结构 | 数组的元素为散列 | 访问

```
1 # print the whole thing with refs
2 for $hhref ( @AoH ) {
3     print "{ ";
4     for $role ( keys %$hhref ) {
5         print "$role=$hhref->{$role} ";
6     }
7     print " } \n";
8 }
9 # print the whole thing with indices
10 for $i ( 0 .. $#AoH ) {
11     print "$i is { ";
12     for $role ( keys %{ $AoH[$i] } ) {
13         print "$role=$AoH[$i]{$role} ";
14     }
15     print " } \n";
16 }
```



```
1 # print the whole thing one at a time
2 for $i ( 0 .. $#AoH ) {
3     for $role ( keys %{ $AoH[$i] } ) {
4         print "element $i $role is $AoH[$i]{$role}
5             }\n";
6 }
```



知识拓展 | 复杂数据结构 | 散列的元素为散列 | 声明

```
1 %HoH = (
2   flintstones => {
3     lead      => "fred",
4     pal       => "barney",
5   },
6   jetsons     => {
7     lead      => "george",
8     wife      => "jane",
9     "his boy" => "elroy",
10  },
11  simpsons   => {
12    lead      => "homer",
13    wife      => "marge",
14    kid       => "bart",
15  },
16 );
```



知识拓展 | 复杂数据结构 | 散列的元素为散列 | 生成

```
1 # reading from file
2 # flintstones: lead=fred pal=barney wife=
  wilma pet=dino
3 while ( <> ) {
4   next unless s/^(.*):\s*/;;
5   $who = $1;
6   for $field ( split ) {
7     ($key, $value) = split /=/, $field;
8     $HoH{$who}{$key} = $value;
9   }
10 }
```



知识拓展 | 复杂数据结构 | 散列的元素为散列 | 生成

```
1 # reading from file; more temps
2 # flintstones: lead=fred pal=barney wife=
  wilma pet=dino
3 while ( <> ) {
4     next unless s/^.*?):\s*/;;
5     $who = $1;
6     $rec = {};
7     $HoH{$who} = $rec;
8     for $field ( split ) {
9         ($key, $value) = split /=/, $field;
10        $rec->{ $key } = $value;
11    }
12 }
```



```
1 # calling a function that returns a key,value  
  hash  
2 for $group ( "simpsons", "jetsons", "  
  flintstones" ) {  
3   $HoH{$group} = { get_family($group) };  
4 }  
5  
6 # likewise, but using temps  
7 for $group ( "simpsons", "jetsons", "  
  flintstones" ) {  
8   %members = get_family($group);  
9   $HoH{$group} = { %members };  
10 }
```



知识拓展 | 复杂数据结构 | 散列的元素为散列 | 生成

```
1 # append new members to an existing family
2 %new_folks = (
3     wife => "wilma",
4     pet   => "dino",
5 );
6
7 for $what (keys %new_folks) {
8     $HoH{flintstones}{$what} = $new_folks{$what}
9 }
```



```
1 # one element
2 $HoH{flintstones}{wife} = "wilma";
3
4 # another element
5 $HoH{simpsons}{lead} =~ s/(\w)/\u$1/;
```



```
1 # print the whole thing
2 foreach $family ( keys %HoH ) {
3     print "$family: { ";
4     for $role ( keys %{ $HoH{$family} } ) {
5         print "$role=$HoH{$family}{$role} ";
6     }
7     print " } \n";
8 }
```



```
1 # print the whole thing somewhat sorted
2 foreach $family ( sort keys %HoH ) {
3     print "$family: { ";
4     for $role ( sort keys %{ $HoH{$family} } )
5     ) {
6         print "$role=$HoH{$family}{$role} ";
7     }
8 }
```



```
1 # print the whole thing sorted by number of
2 # members
3
4 foreach $family ( sort { keys %{$HoH{$b}} } <=>
5   keys %{$HoH{$a}} } ) { HoH
6   print "$family: { ";
7   for $role ( sort keys %{ $HoH{$family} } )
8     {
9       print "$role=$HoH{$family}{$role} ";
10      }
11    print " } \n";
12 }
```



知识拓展 | 复杂数据结构 | 散列的元素为散列 | 访问

```
1 # establish a sort order (rank) for each role
2 $i = 0;
3 for ( qw(lead wife son daughter pal pet) ) {
4     $rank{$_} = ++$i }
5 # now print the whole thing sorted by number
6 # of members
7 foreach $family ( sort { keys %{$HoH{$b}} <=> keys %{$HoH{$a}} } keys %HoH ) {
8     print "$family: { ";
9     # and print these according to rank order
10    for $role ( sort { $rank{$a} <=gt; $rank{$b} } keys %{$HoH{$family}} ) {
11        print "$role=$HoH{$family}{$role} ";
12    }
13    print " } \n";
14 }
```



```
1 $rec = {  
2     TEXT      => $string,  
3     SEQUENCE   => [ @old_values ],  
4     LOOKUP     => { %some_table },  
5     THATCODE   => \&some_function,  
6     THISCODE   => sub { $_[0] ** $_[1] },  
7     HANDLE     => \*STDOUT,  
8 };
```



知识拓展 | 复杂数据结构 | 更复杂的结构 | 声明

```
1 print $rec->{TEXT};  
2 print $rec->{SEQUENCE}[0];  
3 $last = pop @ { $rec->{SEQUENCE} };  
4 print $rec->{LOOKUP}{"key"};  
5 ($first_k, $first_v) = each %{ $rec->{LOOKUP} };  
6 $answer = $rec->{THATCODE}->($arg);  
7 $answer = $rec->{THISCODE}->($arg1, $arg2);  
8 # careful of extra block braces on fh ref  
9 print { $rec->{HANDLE} } "a string\n";  
10 use FileHandle;  
11 $rec->{HANDLE}->autoflush(1);  
12 $rec->{HANDLE}->print(" a string\n");
```



知识拓展 | 复杂数据结构 | 更复杂的结构 | 定义

```
1 %TV = (
2   flintstones => {
3     series      => "flintstones",
4     nights      => [ qw(monday thursday friday) ],
5     members     => [
6       { name => "fred", role => "lead", age => 36, },
7       { name => "wilma", role => "wife", age => 31, },
8       { name => "pebbles", role => "kid", age => 4, },
9     ],
10   },
11   jetsons      => {
12     series      => "jetsons",
13     nights      => [ qw(wednesday saturday) ],
14     members     => [
15       { name => "george", role => "lead", age => 41, },
16       { name => "jane", role => "wife", age => 39, },
17       { name => "elroy", role => "kid", age => 9, },
18     ],
19   },
20   simpsons    => {
```



```
1 # reading from file
2 # here's a piece by piece build up
3 $rec = {};
4 $rec->{series} = "flintstones";
5 $rec->{nights} = [ find_days() ];
6 @members = ();
7 # assume this file in field=value syntax
8 while (<>) {
9     %fields = split /[\s=]+/;
10    push @members, { %fields };
11 }
12 $rec->{members} = [ @members ];
13 # now remember the whole thing
14 $TV{ $rec->{series} } = $rec;
```



知识拓展 | 复杂数据结构 | 更复杂的结构 | 生成

```
1 #
#####
# now, you might want to make interesting extra
# fields that
# include pointers back into the same data
# structure so if
# change one piece, it changes everywhere, like
# for example
# if you wanted a {kids} field that was a
# reference
# to an array of the kids' records without having
# duplicate
# records and thus update problems.
#
#####
```



```
1 foreach $family (keys %TV) {  
2     $rec = $TV{$family}; # temp pointer  
3     @kids = ();  
4     for $person ( @{ $rec->{members} } ) {  
5         if ($person->{role} =~ /kid|son|daughter  
/) {  
6             push @kids, $person;  
7         }  
8     }  
9     # REMEMBER: $rec and $TV{$family} point to  
same data!!  
10    $rec->{kids} = [ @kids ];  
11 }
```



```
1 # you copied the array, but the array itself  
2 # contains pointers  
3 # to uncopied objects. this means that if you  
4 # make bart get  
5 # older via  
6 $TV{$simpsons}{kids}[0]{age}++;  
7 # then this would also change in  
8 print $TV{$simpsons}{members}[2]{age};  
9 # because $TV{$simpsons}{kids}[0] and $TV{  
# simpsons}{members}[2]  
10 # both point to the same underlying anonymous  
# hash table
```



知识拓展 | 复杂数据结构 | 更复杂的结构 | 生成

```
1 # print the whole thing
2 foreach $family ( keys %TV ) {
3     print "the $family";
4     print " is on during @{ $TV{$family}{nights} } \n";
5     print "its members are:\n";
6     for $who ( @{ $TV{$family}{members} } ) {
7         print " $who->{name} ($who->{role}), age $who
8 ->{age}\n";
9     }
10    print "it turns out that $TV{$family}{lead} has
11    ";
12    print scalar ( @{ $TV{$family}{kids} } ), " kids
13    named ";
14    print join (", ", map { $_->{name} } @{ $TV{
15        $family}{kids} });
16    print "\n";
17 }
```



教学提纲

1

引言

2

散列

3

数据结构和算法

4

遗传密码

- 密码子翻译成氨基酸
- 遗传密码的冗余性
- 使用散列表表示遗传密码

5

DNA 翻译成蛋白质

6

读取 FASTA 格式的 DNA

- 序列格式
- 读取 FASTA 文件的思路
- 读取 FASTA 文件的子程序
- 输出格式化的序列数据

- 读取 FASTA 文件的主程序
- DNA 翻译成蛋白质的主程序

7

阅读框

- 简介
- 翻译阅读框

8

回顾和总结

- 总结
- 思考题

9

知识拓展

- Perl 里面的复杂数据结构
- 数据共享
- 宽格式和长格式的数据
- 数据处理实例



What you should deliver to the statistician

For maximum speed in the analysis this is the information you should pass to a statistician:

- ① The raw data.
- ② A tidy data set
- ③ A code book describing each variable and its values in the tidy data set.
- ④ An explicit and exact recipe you used to go from 1 → 2,3



It is critical that you include the rawest form of the data that you have access to. Here are some examples of the raw form of data:

- The strange binary file your measurement machine spits out
- The unformatted Excel file with 10 worksheets the company you contracted with sent you
- The complicated JSON data you got from scraping the Twitter API
- The hand-entered numbers you collected looking through a microscope



You know the raw data is in the right format if you:

- ① Ran no software on the data
- ② Did not manipulate any of the numbers in the data
- ③ You did not remove any data from the data set
- ④ You did not summarize the data in any way



If you did any manipulation of the data at all it is not the raw form of the data. Reporting manipulated data as raw data is a very common way to slow down the analysis process, since the analyst will often have to do a forensic study of your data to figure out why the raw data looks weird.



The four general principles you should pay attention to are:

- ① Each **variable** you measure should be in one **column**
- ② Each different **observation** of that variable should be in a different **row**
- ③ There should be one **table** for each “**kind**” of variable
- ④ If you have multiple tables, they should include a column in the table that allows them to be **linked**



While these are the hard and fast rules, there are a number of other things that will make your data set much easier to handle. First is to include **a row at the top** of each data table/spreadsheet that contains **full variable names**. So if you measured age at diagnosis for patients, you would head that column with the name `AgeAtDiagnosis` instead of something like `ADx` or another abbreviation that may be hard for another person to understand.

If you are sharing your data with the collaborator in Excel, the tidy data should be in one Excel file per table. They should not have multiple worksheets, no macros should be applied to the data, and no columns/cells should be highlighted. Alternatively share the data in a CSV or TAB-delimited text file.



Here is an example of how this would work from genomics. Suppose that for 20 people you have collected gene expression measurements with RNA-sequencing. You have also collected demographic and clinical information about the patients including their age, treatment, and diagnosis. You would have one table/spreadsheet that contains the clinical/demographic information. It would have four columns (patient id, age, treatment, diagnosis) and 21 rows (a row with variable names, then one row for every patient). You would also have one spreadsheet for the summarized genomic data. Usually this type of data is summarized at the level of the number of counts per exon. Suppose you have 100,000 exons, then you would have a table/spreadsheet that had 21 rows (a row for gene names, and one row for each patient) and 100,001 columns (one column for patient ids and one column for each data type).



- ① Each variable forms a column (Each variable in the data set is placed in its own column)
- ② Each observation forms a row (Each observation is placed in its own row)
- ③ Each value is placed in its own cell
- ④ Each type of observational unit forms a table

country	year	cases	population
Afghanistan	2000	15	1507071
Afghanistan	2000	666	2095360
Brazil	1999	30737	17206362
Brazil	2000	80488	17404898
China	1999	210258	127215272
China	2000	210766	128012583

variables

country	year	cases	population
Afghanistan	2000	15	1507071
Afghanistan	2000	666	2095360
Brazil	1999	30737	17206362
Brazil	2000	80488	17404898
China	1999	210258	127215272
China	2000	210766	128012583

observations

country	year	cases	population
Afghanistan	2000	15	1507071
Afghanistan	2000	666	2095360
Brazil	1999	30737	17206362
Brazil	2000	80488	17404898
China	1999	210258	127215272
China	2000	210766	128012583

values



	treatmenta	treatmentb
John Smith	—	2
Jane Doe	16	11
Mary Johnson	3	1

	John Smith	Jane Doe	Mary Johnson
treatmenta	—	16	3
treatmentb	2	11	1

name	trt	result
John Smith	a	—
Jane Doe	a	16
Mary Johnson	a	3
John Smith	b	2
Jane Doe	b	11
Mary Johnson	b	1



知识拓展 | datasharing | tidy data | raw2tidy

country	year	m014	m1524	m2534	m3544	m4554	m5564	m65	mu	f014
AD	2000	0	0	1	0	0	0	0	—	—
AE	2000	2	4	4	6	5	12	10	—	3
AF	2000	52	228	183	149	129	94	80	—	93
AG	2000	0	0	0	0	0	0	1	—	1
AL	2000	2	19	21	14	24	19	16	—	3
AM	2000	2	152	130	131	63	26	21	—	1
AN	2000	0	0	1	2	0	0	0	—	0
AO	2000	186	999	1003	912	482	312	194	—	247
AR	2000	97	278	594	402	419	368	330	—	121
AS	2000	—	—	—	—	1	1	—	—	—



知识拓展 | datasharing | tidy data | raw2tidy

country	year	column	cases
AD	2000	m014	0
AD	2000	m1524	0
AD	2000	m2534	1
AD	2000	m3544	0
AD	2000	m4554	0
AD	2000	m5564	0
AD	2000	m65	0
AE	2000	m014	2
AE	2000	m1524	4
AE	2000	m2534	4
AE	2000	m3544	6
AE	2000	m4554	5
AE	2000	m5564	12
AE	2000	m65	10
AE	2000	f014	3

(a) Molten data

country	year	sex	age	cases
AD	2000	m	0-14	0
AD	2000	m	15-24	0
AD	2000	m	25-34	1
AD	2000	m	35-44	0
AD	2000	m	45-54	0
AD	2000	m	55-64	0
AD	2000	m	65+	0
AE	2000	m	0-14	2
AE	2000	m	15-24	4
AE	2000	m	25-34	4
AE	2000	m	35-44	6
AE	2000	m	45-54	5
AE	2000	m	55-64	12
AE	2000	m	65+	10
AE	2000	f	0-14	3

(b) Tidy data



For almost any data set, the measurements you calculate will need to be described in more detail than you will sneak into the spreadsheet. The code book contains this information. At minimum it should contain:

- ① Information about the variables (including units!) in the data set not contained in the tidy data
- ② Information about the summary choices you made
- ③ Information about the experimental study design you used

A common format for this document is a Word (or Markdown) file. There should be a section called “Study design” that has a thorough description of how you collected the data. There is a section called “Code book” that describes each variable and its units.



In our genomics example, the analyst would want to know what the unit of measurement for each clinical/demographic variable is (age in years, treatment by name/dose, level of diagnosis and how heterogeneous). They would also want to know how you picked the exons you used for summarizing the genomic data (UCSC/Ensembl, etc.). They would also want to know any other information about how you did the data collection/study design. For example, are these the first 20 patients that walked into the clinic? Are they 20 highly selected patients by some characteristic like age? Are they randomized to treatments?



When you put variables into a spreadsheet there are several main categories you will run into depending on their data type:

- ① Continuous
- ② Ordinal
- ③ Categorical
- ④ Missing
- ⑤ Censored

Continuous variables are anything measured on a quantitative scale that could be any fractional number. An example would be something like weight measured in kg. Ordinal data are data that have a fixed, small (< 100) number of levels but are ordered. This could be for example survey responses where the choices are: poor, fair, good.

Categorical data are data where there are multiple categories, but they aren't ordered. One example would be sex: male or female. Missing data are data that are missing and you don't know the mechanism. You should code missing values as NA. Censored data/make up/ throw away missing observations.



In general, try to avoid coding categorical or ordinal variables as numbers. When you enter the value for sex in the tidy data, it should be "male" or "female". The ordinal values in the data set should be "poor", "fair", and "good" not 1, 2 ,3. This will avoid potential mixups about which direction effects go and will help identify coding errors.

Always encode every piece of information about your observations using **text**. For example, if you are storing data in Excel and use a form of colored text or cell background formatting to indicate information about an observation ("red variable entries were observed in experiment 1.") then this information will not be exported (and will be lost!) when the data is exported as raw text. Every piece of data should be encoded as actual text that can be exported.



You may have heard this before, but reproducibility is kind of a big deal in computational science. That means, when you submit your paper, the reviewers and the rest of the world should be able to exactly replicate the analyses from raw data all the way to final results. If you are trying to be efficient, you will likely perform some summarization/ data analysis steps before the data can be considered tidy.

The ideal thing for you to do when performing summarization is to create a computer script (in R, Python, or something else) that takes the raw data as input and produces the tidy data you are sharing as output. You can try running your script a couple of times and see if the code produces the same output.



In many cases, the person who collected the data has incentive to make it tidy for a statistician to speed the process of collaboration. They may not know how to code in a scripting language. In that case, what you should provide the statistician is something called pseudocode. It should look something like:

- ① Step 1 - take the raw file, run version 3.1.2 of summarize software with parameters a=1, b=2, c=3
- ② Step 2 - run the software separately for each sample
- ③ Step 3 - take column three of outputfile.out for each sample and that is the corresponding row in the output data set

You should also include information about which **system** (Mac/Windows/Linux) you used the software on and whether you tried it more than once to confirm it gave the same results. Ideally, you will run this by a fellow student/labmate to confirm that they can obtain the same output file you did.



When you turn over a properly tidied data set it dramatically decreases the workload on the statistician. So hopefully they will get back to you much sooner. But most careful statisticians will check your recipe, ask questions about steps you performed, and try to confirm that they can obtain the same tidy data that you did with, at minimum, spot checks.



You should then expect from the statistician:

- ① An analysis script that performs each of the analyses (not just instructions)
- ② The exact computer code they used to run the analysis
- ③ All output files/figures they generated.

This is the information you will use in the supplement to establish reproducibility and precision of your results. Each of the steps in the analysis should be clearly explained and you should ask questions when you don't understand what the analyst did. It is the responsibility of both the statistician and the scientist to understand the statistical analysis. You may not be able to perform the exact analyses without the statistician's code, but you should be able to explain why the statistician performed each step to a labmate/your principal investigator.



教学提纲

1

引言

2

散列

3

数据结构和算法

4

遗传密码

- 密码子翻译成氨基酸
- 遗传密码的冗余性
- 使用散列表表示遗传密码

5

DNA 翻译成蛋白质

6

读取 FASTA 格式的 DNA

- 序列格式
- 读取 FASTA 文件的思路
- 读取 FASTA 文件的子程序
- 输出格式化的序列数据

- 读取 FASTA 文件的主程序
- DNA 翻译成蛋白质的主程序

7

阅读框

- 简介
- 翻译阅读框

8

回顾和总结

- 总结
- 思考题

9

知识拓展

- Perl 里面的复杂数据结构
- 数据共享
- 宽格式和长格式的数据
- 数据处理实例



wide-format

Wide data has a column for each variable.

long-format

Long-format data has a column for possible variable types and a column for the values of those variables.

wide vs. long

- It turns out that you need wide-format data for some types of data analysis and long-format data for others.
- In reality, you need long-format data much more commonly than wide-format data.
- But people often find it easier to record their data in wide format.

wide-format

Wide data has a column for each variable.

long-format

Long-format data has a column for possible variable types and a column for the values of those variables.

wide vs. long

- It turns out that you need wide-format data for some types of data analysis and long-format data for others.
- In reality, you need long-format data much more commonly than wide-format data.
- But people often find it easier to record their data in wide format.

wide-format

Wide data has a column for each variable.

long-format

Long-format data has a column for possible variable types and a column for the values of those variables.

wide vs. long

- It turns out that you need wide-format data for some types of data analysis and long-format data for others.
- In reality, you need long-format data much more commonly than wide-format data.
- But people often find it easier to record their data in wide format.

知识拓展 | data | wide vs. long

subject	sex	condition		measurement
		control	cond1	
1	M	control	7.9	12.3
	M	cond1	10.7	10.6
	M	cond2	11.1	9.5
	F	control	6.3	13.1
2	F	cond1	10.6	13.8
	F	cond2	11.1	11.5
	F	control	13.8	13.4
	M	cond1	12.9	11.5
subject	sex	control	cond1	cond2
1	M	7.9	12.3	10.7
2	F	6.3	10.6	11.1
3	F	9.5	13.1	13.8
4	M	11.5	13.4	12.9



知识拓展 | data | wide vs. long

gene	A1	A2	B1	B2
GENE_1	35.41	36.60	29.96	29.73
GENE_2	36.60	23.45	24.39	24.74
GENE_3	29.96	23.30	32.17	25.94
GENE_4	29.73	22.84	31.66	26.22
GENE_5	34.46	22.79	31.39	24.75
GENE_6	35.66	21.37	31.34	24.72
GENE_7	33.28	21.74	31.10	25.39
GENE_8	33.03	22.96	30.90	25.65
GENE_9	26.58	22.87	31.14	30.57
GENE_10	26.05	25.18	31.03	29.99
CONTROL	26.60	25.60	26.03	25.79

gene	sample_id	ct_value
GENE_1	A1	35.41
GENE_2	A1	36.60
GENE_3	A1	29.96
GENE_4	A1	29.73
GENE_5	A1	34.46
GENE_6	A1	35.66
GENE_7	A1	33.28
GENE_8	A1	33.03
GENE_9	A1	26.58
GENE_10	A1	26.05
CONTROL	A1	26.60
GENE_1	A2	36.60
GENE_2	A2	23.45
GENE_3	A2	23.30

知识拓展 | data | wide vs. long

```
dcast formula dcast(aql, month + day ~ variable, value.var = "value")
```

	ID variables (left side of formula)	Variable to swing into column names (right side of formula)	Values (value.var)
Long-format data	month	day	variable
	5	1	ozone
	5	2	ozone
	5	3	ozone
	5	4	ozone
	5	5	ozone
	5	6	ozone
Wide-format data	month	day	ozone solar.r wind temp
	5	1	41 190 7.4 67
	5	2	36 118 8.0 72
	5	3	12 149 12.6 74
	5	4	18 313 11.5 62
	5	5	NA NA 14.3 56
	5	6	28 NA 14.9 66

知识拓展 | data | wide2long

country	year	cases	country	1999	2000
Afghanistan	1999	745	Afghanistan	745	2666
Afghanistan	2000	2666	Brazil	37737	80488
Brazil	1999	37737	China	212258	213766
Brazil	2000	80488			
China	1999	212258			
China	2000	213766			

table4



知识拓展 | data | long2wide

country	year	key	value	country	year	cases	population
Afghanistan	1999	cases	745	Afghanistan	1999	745	19987071
Afghanistan	1999	population	19987071	Afghanistan	2000	2666	20595360
Afghanistan	2000	cases	2666	Brazil	1999	37737	172006362
Afghanistan	2000	population	20595360	Brazil	2000	80488	174504898
Brazil	1999	cases	37737	China	1999	212258	1272915272
Brazil	1999	population	172006362	China	2000	213766	1280428583
Brazil	2000	cases	80488				
Brazil	2000	population	174504898				
China	1999	cases	212258				
China	1999	population	1272915272				
China	2000	cases	213766				
China	2000	population	1280428583				

教学提纲

1

引言

2

散列

3

数据结构和算法

4

遗传密码

- 密码子翻译成氨基酸
- 遗传密码的冗余性
- 使用散列表表示遗传密码

5

DNA 翻译成蛋白质

6

读取 FASTA 格式的 DNA

- 序列格式
- 读取 FASTA 文件的思路
- 读取 FASTA 文件的子程序
- 输出格式化的序列数据

- 读取 FASTA 文件的主程序
- DNA 翻译成蛋白质的主程序

7

阅读框

- 简介
- 翻译阅读框

8

回顾和总结

- 总结
- 思考题

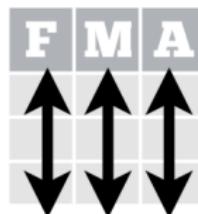
9

知识拓展

- Perl 里面的复杂数据结构
- 数据共享
- 宽格式和长格式的数据
- 数据处理实例

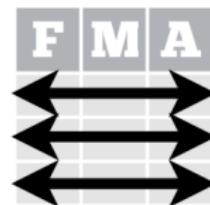


In a tidy
data set:



Each **variable** is saved
in its own **column**

&



Each **observation** is
saved in its own **row**



知识拓展 | example | wide vs. long

	Id	name	e6	e7	e8	e9	final
1	2013052101	连明	86	82	83	85	88
2	2013052102	李秉宣	88	85	83	86	77
3	2013052103	何雨泽	84	85	84	85	79
4	2013052104	陈泽川	80	89	84	85	60
5	2013052105	张韬	85	85	82	84	90
6	2013052106	李发金	89	85	88	89	94
7	2013052107	杨屹顶	83	85	80	81	75
8	2013052109	高奕	84	87	81	82	70
9	2013052110	孙少省	91	89	85	86	69
10	2013052112	王晨	78	78	75	75	52
11	2013052113	高杨	79	77	75	75	49
12	2013052114	李嘉良	76	76	75	75	48
13	2013052115	刘赜宇	81	82	80	80	85
14	2013052116	赵天晨	89	88	86	85	91
15	2013052118	彭永林	90	89	89	88	84
16	2013052119	江婉婷	91	94	88	89	95
17	2013052120	朱依青	93	95	92	90	79
18	2013052121	韦秋霞	88	83	86	87	91
19	2013052122	陆玉妹	91	92	89	95	86
20	2013052123	吴培伟	95	94	97	95	71

	id	name	class	score
1	2013052101	连明	e6	86
2	2013052102	李秉宣	e6	88
3	2013052103	何雨泽	e6	84
4	2013052104	陈泽川	e6	80
5	2013052105	张韬	e6	85
6	2013052106	李发金	e6	89
7	2013052107	杨屹顶	e6	83
8	2013052109	高奕	e6	84
9	2013052110	孙少省	e6	91
10	2013052112	王晨	e6	78
11	2013052113	高杨	e6	79
12	2013052114	李嘉良	e6	76
13	2013052115	刘晓宇	e6	81
14	2013052116	赵天晨	e6	89
15	2013052118	彭永林	e6	90
16	2013052119	江姝婷	e6	91
17	2013052120	朱依青	e6	93
18	2013052121	韦秋霞	e6	88
19	2013052122	陆玉妹	e6	91
20	2013052123	任玲玲	e6	96

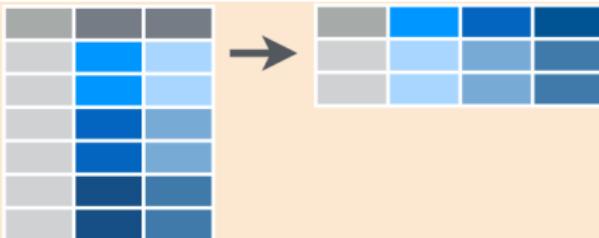


知识拓展 | example | Reshaping data



`tidyverse::gather(cases, "year", "n", 2:4)`

Gather columns into rows.



`tidyverse::spread(pollution, size, amount)`

Spread rows into columns.

```
1 library(readr)
2 library(tidyr)
3 library(dplyr)
4 library(stringr)
5 library(ggplot2)
6 library(ComplexHeatmap)
7 library(circlize)
8
9 score_wide <- read_csv("score_2013.csv")
10 score_long <- gather(score_wide, "class", "
    score", 3:7)
11
12 score_long <- score_long %>% mutate(id2=as.
    factor(str_replace(id, "20130521", "")))
```

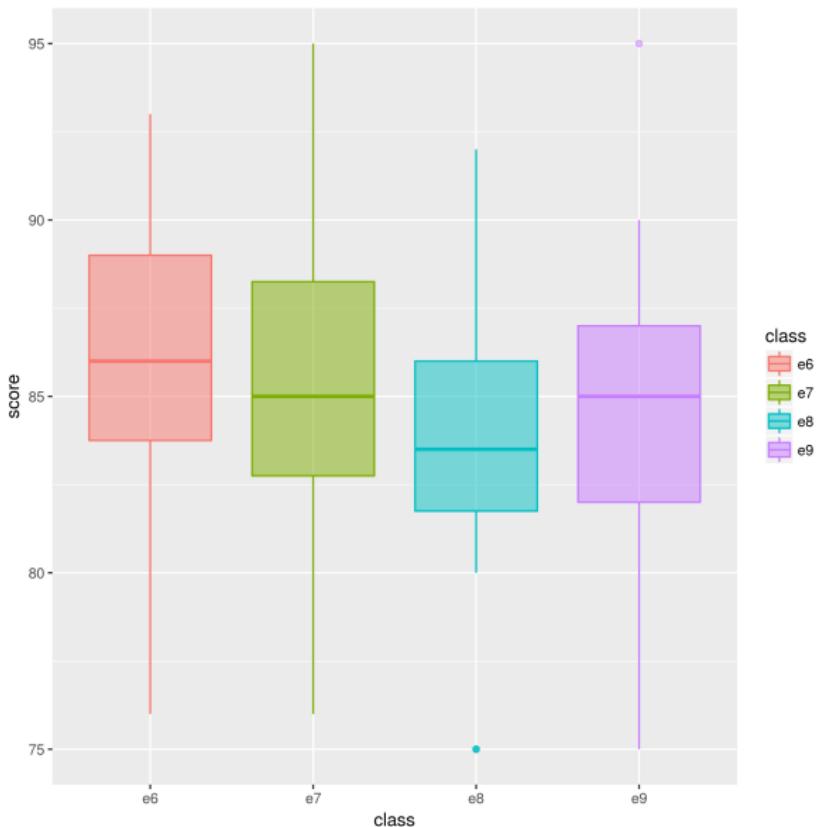


知识拓展 | example | R | boxplot

```
1 score_long2 <- score_long %>% filter(str_
  detect(class, "e"))
2
3 gb <- ggplot(score_long2, aes(class, score))
4 gb <- gb + geom_boxplot(aes(fill=class, color
  =class), alpha=0.5)
5
6 ggsave("exp_boxplot.png", gb)
```



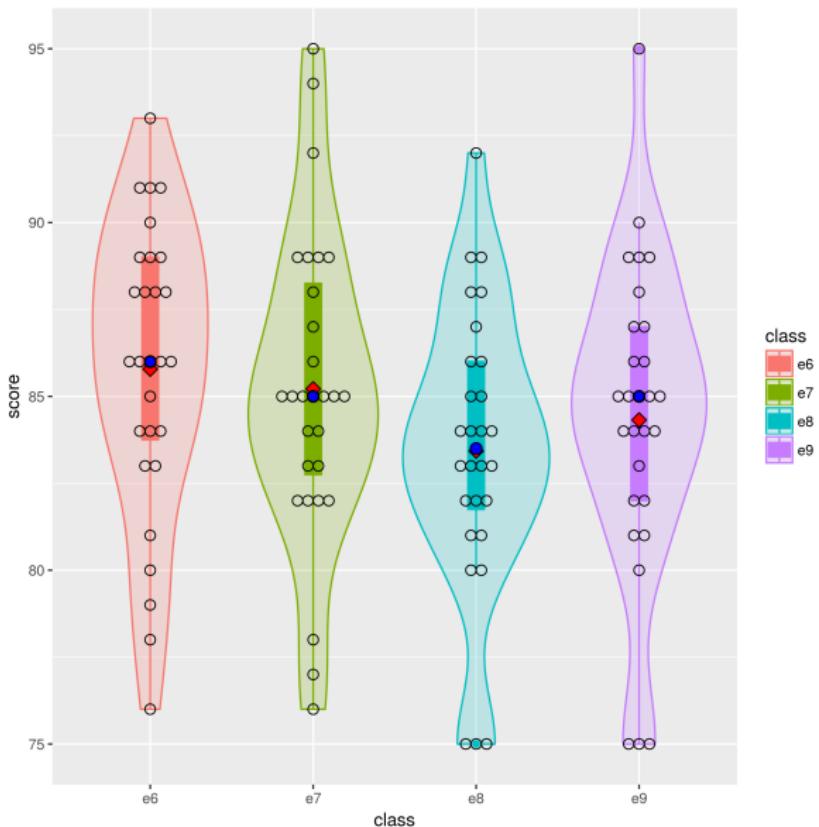
知识拓展 | example | R | plot | boxplot



```
1 gv <- ggplot(score_long2, aes(class, score))
2 gv <- gv + geom_violin(aes(fill=class, color=
  class), alpha=0.2)
3 gv <- gv + geom_boxplot(aes(fill=class, color
  =class), width=0.1)
4 gv <- gv + stat_summary(fun.y="mean", geom="
  point", shape=23, size=3, fill="red")
5 gv <- gv + stat_summary(fun.y="median", geom=
  "point", shape=21, size=3, fill="blue")
6 gv <- gv + geom_dotplot(binaxis="y", binwidth
  =0.3, stackdir="center", fill=NA)
7
8 ggsave("exp_violin.png", gv)
```



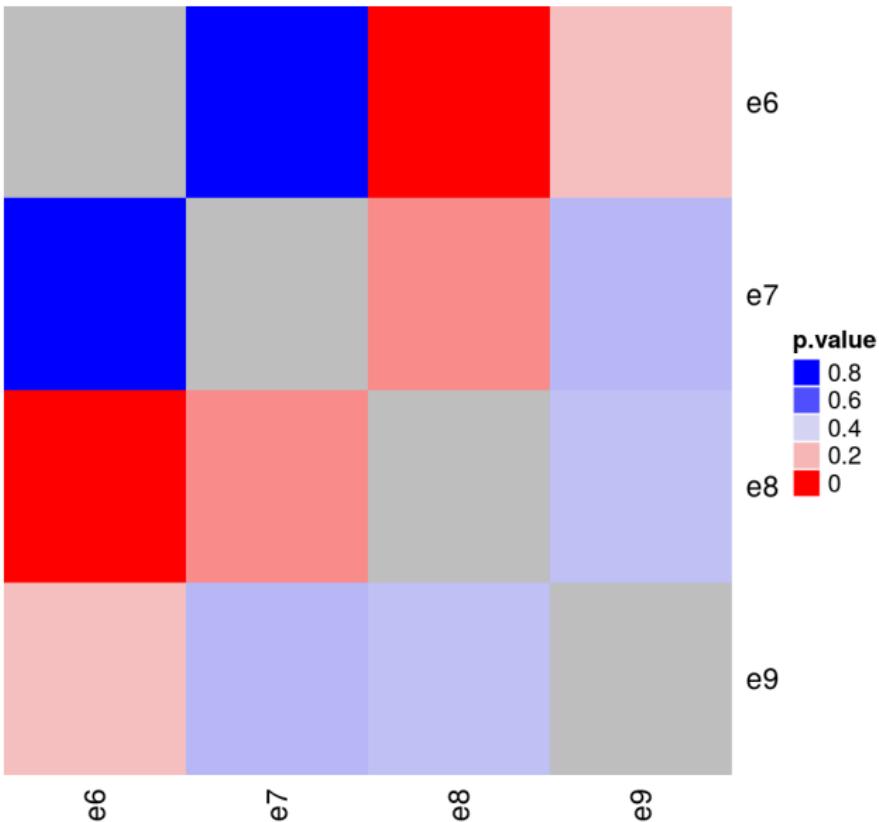
知识拓展 | example | R | plot | violin



知识拓展 | example | R | heatmap

```
1 tri2mat <- function(tp, pf) {  
2   tp <- cbind(tp, rep(NA, nrow(tp)))  
3   tp <- rbind(rep(NA, ncol(tp)), tp)  
4   colnames(tp) <- pf; rownames(tp) <- pf  
5   tp[upper.tri(tp)] <- t(tp)[upper.tri(tp)]  
6   tp  
7 }  
8  
9 t_test <- pairwise.t.test(score_long2$score, score_long2  
  $class, p.adjust.method="none", pool.sd=FALSE)  
10 tp <- t_test$p.value  
11 pf <- levels(as.factor(score_long2$class))  
12 tm <- tri2mat(tp, pf)  
13 col = colorRamp2(seq(min(tm, na.rm=TRUE), max(tm, na.rm=  
  TRUE), length=3), c("red", "#EEEEEE", "blue"), space="  
  RGB")  
14 png("exp_heatmap.png", width=1024, height=960, res=200)  
15 Heatmap(tm, col=col, heatmap_legend_param=list(title="p.  
  value"), cluster_rows=FALSE, cluster_columns=FALSE)  
16 dev.off()
```



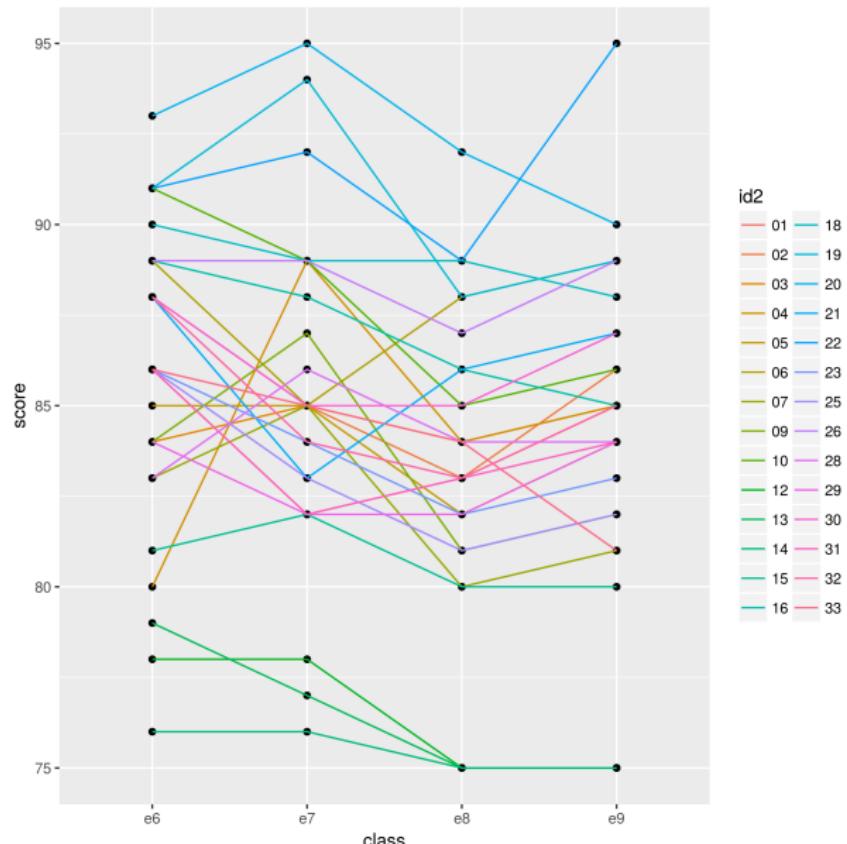


知识拓展 | example | R | line

```
1 gl <- ggplot(score_long2, aes(class, score,  
  group=id2))  
2 gl <- gl + geom_point()  
3 gl <- gl + geom_line(aes(color=id2))  
4  
5 ggsave("id_lineplot.png", gl)
```



知识拓展 | example | R | plot | line



知识拓展 | example | R | correlation(1/3)

```
1 lm_eqn <- function(df) {  
2   m <- lm(final ~ exp, df)  
3   eq <- substitute(italic(y) == a + b %.%  
4                     italic(x) ^ " * " ~ italic(r) ^ " = " ~ r1 ^ " * " ^ ~  
5                     italic(R) ^ 2 ^ " = " ~ r2,  
6                     list(a = format(coef(m)  
7 [1], digits=2),  
8                      b = format(coef(m)  
9 [2], digits=2),  
10                     r1 = format(sqrt(  
11                     summary(m)$r.squared), digits=3),  
12                     r2 = format(summary(m)  
13                     )$r.squared, digits=3)))  
14   as.character(as.expression(eq))  
15 }
```

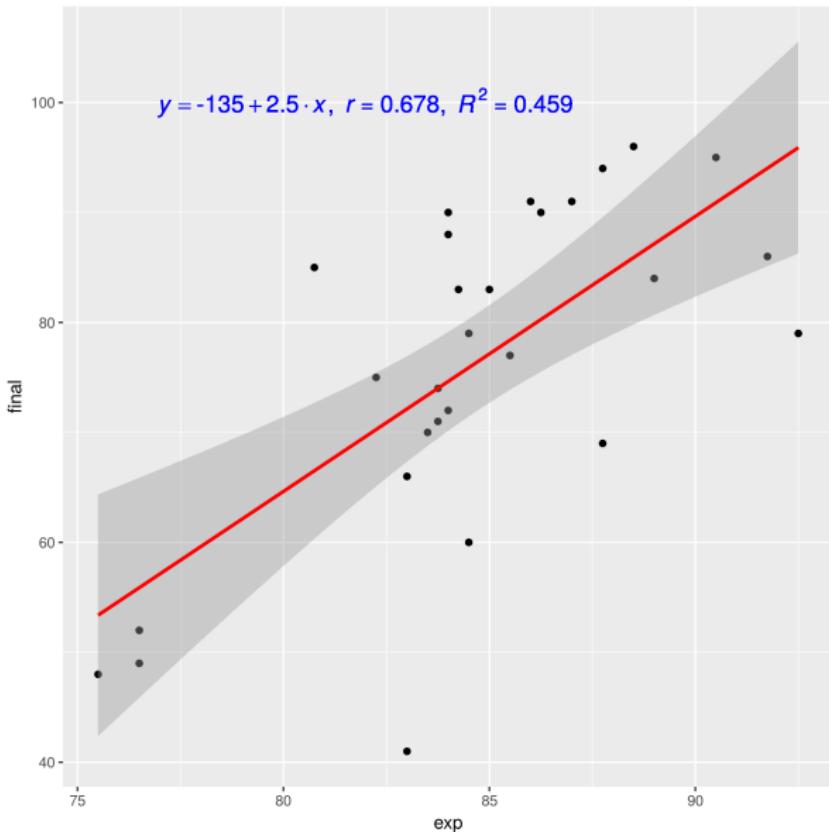


知识拓展 | example | R | correlation(2/3)

```
1 plot_cor <- function (df) {  
2   gr <- ggplot(df, aes(exp, final))  
3   gr <- gr + geom_point()  
4   gr <- gr + geom_smooth(method="lm", color="red", formula=y~x)  
5   gr <- gr + geom_text(x=82, y=100, label=lm_eqn(df), parse=TRUE, size=5, color="blue")  
6 }  
7  
8 score_wide2 <- score_wide %>% group_by(id,  
9       name, final) %>% summarise(exp=mean(c(e6,e7,  
10      e8,e9)))  
11 gr <- plot_cor(score_wide2)  
12 ggsave("correlation_plot.png", gr)
```



知识拓展 | example | R | plot | correlation(1/2)

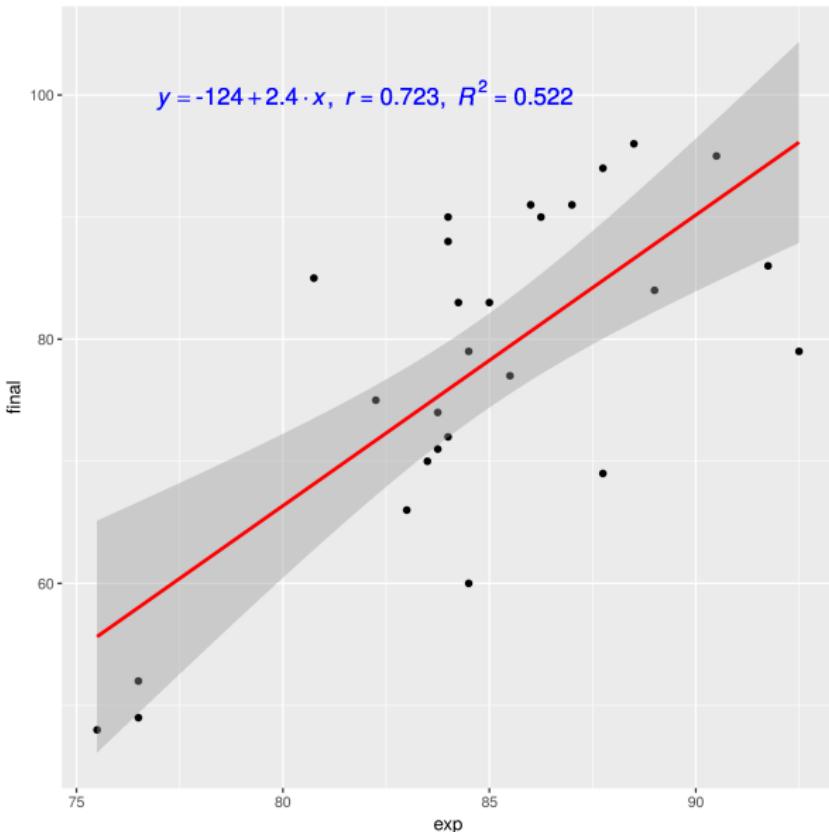


知识拓展 | example | R | correlation(3/3)

```
1 id1 <- score_wide2[which.min(score_wide2$  
  final),]$id  
2 score_wide3 <- score_wide2 %>% filter(id!=id1  
  )  
3  
4 gr2 <- plot_cor(score_wide3)  
5  
6 ggsave("correlation_plot_correct.png", gr2)
```



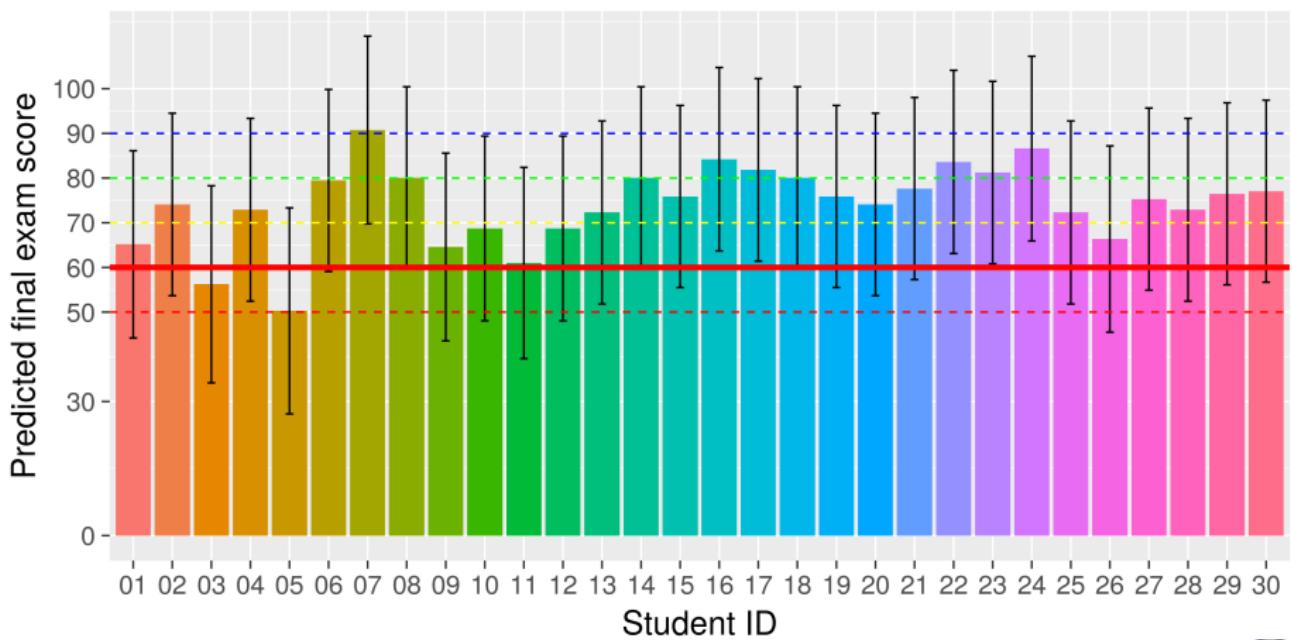
知识拓展 | example | R | plot | correlation(2/2)



知识拓展 | example | R | bar

```
1 m <- lm(final ~ exp, score_wide3)
2 score2014 <- read_csv("score_2014.csv")
3 mp <- predict(m, score2014, interval="prediction")
4 s4 <- bind_cols(score2014, as.data.frame(mp))
5 colnames(s4) <- c("ID", "name", "exp", "final", "fl", "fu")
6 s4 <- s4 %>% mutate(id=as.factor(str_replace(ID, "20140521", "")))
7
8 g <- ggplot(s4, aes(x=id, y=final, fill=id))
9 g <- g + theme_gray(base_size=18)
10 g <- g + geom_bar(stat="identity")
11 g <- g + geom_errorbar(aes(ymax=fu, ymin=fl), width=0.2)
12 g <- g + geom_hline(yintercept=60, color="red", size=1.5)
13 g <- g + geom_hline(yintercept=c(50, 70, 80, 90), color=c("red",
   "yellow", "green", "blue"), size=0.5, linetype=2)
14 g <- g + scale_y_continuous(breaks=c(0, 30, seq(50,100,10)))
15 g <- g + labs(x="Student ID", y="Predicted final exam score")
16 g <- g + guides(fill=FALSE)
17 ggsave("prediction_2014.png", g, width=10, height=5)
```





Powered by



T_EX L^AT_EX X_ET_EX Beamer