

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

第七章 突变和随机化

伊现富 (Yi Xianfu)

天津医科大学 (TJMU)
生物医学工程与技术学院

2016 年 12 月



教学提纲

1

引言

2

随机数生成器

3

随机化程序

4

模拟 DNA 突变

- 伪代码设计
- 改进设计
- 组合子程序
- bug?

5

生成随机 DNA

- 设计理念
- 伪代码
- 程序

6

分析 DNA

7

知识拓展

8

回顾和总结

- 总结
- 思考题

1 引言

2 随机数生成器

3 随机化程序

4 模拟 DNA 突变

- 伪代码设计
- 改进设计
- 组合子程序
- bug?

5 生成随机 DNA

- 设计理念
- 伪代码
- 程序

6 分析 DNA

- ## 7 知识拓展
- ## 8 回顾和总结
- 总结
 - 思考题



突变和随机化 | 突变

突变

突变 (mutation, 即基因突变)，指细胞中的遗传基因（通常指存在于细胞核中的脱氧核糖核酸）发生的改变。它包括单个碱基改变所引起的点突变，或多个碱基的缺失、重复和插入。

原因

细胞分裂时遗传基因的复制发生错误、或受化学物质、毒性、辐射或病毒的影响等。

影响

- 突变通常会导致细胞运作不正常或死亡，甚至可以在较高等生物中引发癌症。
- 同时，突变也被视为物种进化的“推动力”：不理想的突变会经自然选择过程被淘汰，而对物种有利的突变则会被累积下去。中性突变 (neutral mutation) 对物种没有影响而逐渐累积，会导致间断平衡。

突变和随机化 | 突变

突变

突变 (mutation, 即基因突变)，指细胞中的遗传基因（通常指存在于细胞核中的脱氧核糖核酸）发生的改变。它包括单个碱基改变所引起的点突变，或多个碱基的缺失、重复和插入。

原因

细胞分裂时遗传基因的复制发生错误、或受化学物质、毒性、辐射或病毒的影响等。

影响

- 突变通常会导致细胞运作不正常或死亡，甚至可以在较高等生物中引发癌症。
- 同时，突变也被视为物种进化的“推动力”：不理想的突变会经自然选择过程被淘汰，而对物种有利的突变则会被累积下去。中性突变 (neutral mutation) 对物种没有影响而逐渐累积，会导致间断平衡。

突变和随机化 | 突变

突变

突变 (mutation, 即基因突变)，指细胞中的遗传基因（通常指存在于细胞核中的脱氧核糖核酸）发生的改变。它包括单个碱基改变所引起的点突变，或多个碱基的缺失、重复和插入。

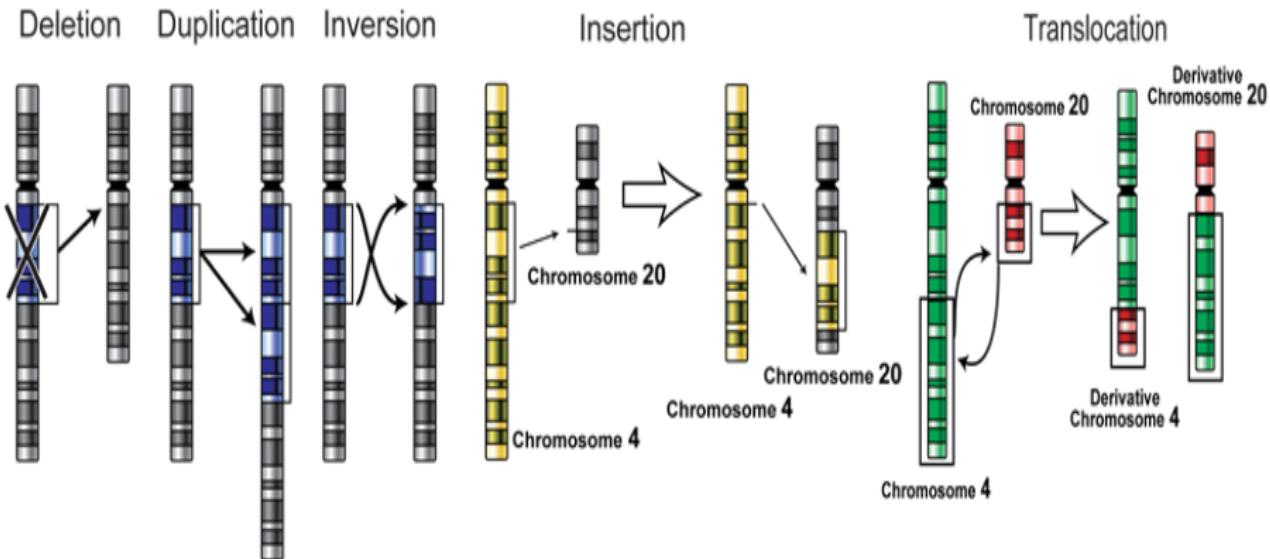
原因

细胞分裂时遗传基因的复制发生错误、或受化学物质、毒性、辐射或病毒的影响等。

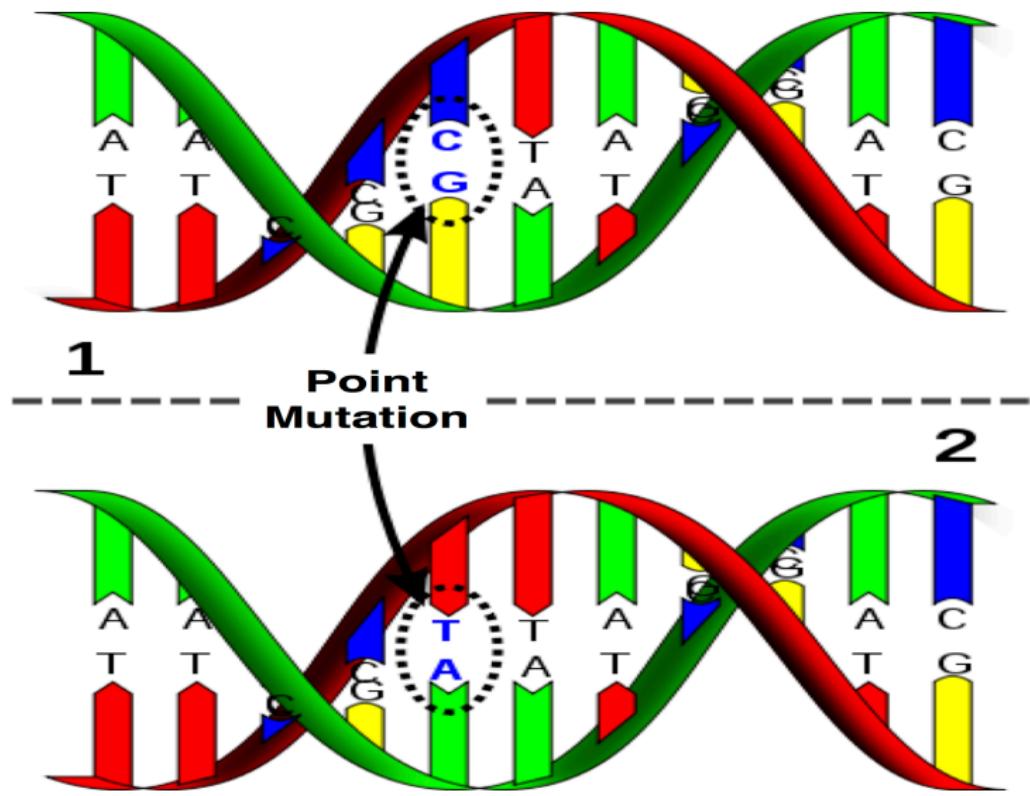
影响

- 突变通常会导致细胞运作不正常或死亡，甚至可以在较高等生物中引发癌症。
- 同时，突变也被视为物种进化的“推动力”：不理想的突变会经自然选择过程被淘汰，而对物种有利的突变则会被累积下去。中性突变 (neutral mutation) 对物种没有影响而逐渐累积，会导致间断平衡。

突变和随机化 | 突变 | 类型



突变和随机化 | 突变 | 点突变



点突变

点突变 (point mutation) 是突变的一种类型，在遗传材料 DNA 或 RNA 中，会使单一碱基核苷酸替换成另一种核苷酸。通常这个术语也包括只作用于单一碱基对的插入或删除。

分类

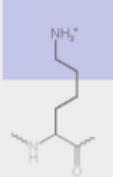
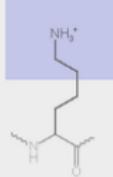
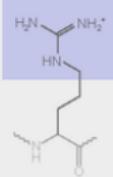
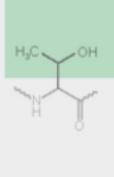
- 转换/颠换：转换 (Alpha) 和颠换 (Beta) 在突变速率有系统的差异，转换突变大于颠换突变约 10 倍以上。
 - 转换：嘌呤替换另一个嘌呤，或嘧啶替换另一个嘧啶
 - 颠换：嘌呤替换嘧啶，嘧啶替换嘌呤
- 功能
 - 无义突变 (nonsense mutation)：使原本可制造蛋白质的密码变成终止密码
 - 错义突变 (missense mutation)：使密码所对应的氨基酸改变
 - 无表型突变 (silent mutation)：密码改变，但对应的氨基酸不变

点突变

点突变 (point mutation) 是突变的一种类型，在遗传材料 DNA 或 RNA 中，会使单一碱基核苷酸替换成另一种核苷酸。通常这个术语也包括只作用于单一碱基对的插入或删除。

分类

- 转换/颠换：转换 (Alpha) 和颠换 (Beta) 在突变速率有系统的差异，转换突变大于颠换突变约 10 倍以上。
 - 转换：嘌呤替换另一个嘌呤，或嘧啶替换另一个嘧啶
 - 颠换：嘌呤替换嘧啶，嘧啶替换嘌呤
- 功能
 - 无义突变 (nonsense mutation)：使原本可制造蛋白质的密码变成终止密码
 - 错义突变 (missense mutation)：使密码所对应的氨基酸改变
 - 无表型突变 (silent mutation)：密码改变，但对应的氨基酸不变

No mutation	Point mutations			conservative	non-conservative
	Silent	Nonsense	Missense		
DNA level	TTC	TTT	ATC	TCC	TGC
mRNA level	AAG	AAA	UAG	AGG	ACG
protein level	Lys	Lys	STOP	Arg	Thr
					
					basic polar



随机性

随机性 (randomness) 这个词是用来表达目的、动机、规则或一些非科学用法的可预测性的缺失。一个随机的过程是一个不定因子不断产生的重复过程，但它可能遵循某个概率分布。

随机经常用于统计学中，表示一些定义清晰的、彻底的统计学属性，例如缺失偏差或者相关。随机与任意不同，因为“一个变量是随机的”表示这个变量遵循概率分布。而任意在另一方面又暗示了变量没有遵循可限定概率分布。

科学与随机

在自然与工程学里一些现象会通过随机性模型来模拟：

- 混沌理论，密码学，博弈论，信息论，量子力学，模式识别
- 统计学，概率论，统计力学

随机性

随机性 (randomness) 这个词是用来表达目的、动机、规则或一些非科学用法的可预测性的缺失。一个随机的过程是一个不定因子不断产生的重复过程，但它可能遵循某个概率分布。

随机经常用于统计学中，表示一些定义清晰的、彻底的统计学属性，例如缺失偏差或者相关。随机与任意不同，因为“一个变量是随机的”表示这个变量遵循概率分布。而任意在另一方面又暗示了变量没有遵循可限定概率分布。

科学与随机

在自然与工程学里一些现象会通过随机性模型来模拟：

- 混沌理论，密码学，博弈论，信息论，量子力学，模式识别
- 统计学，概率论，统计力学

进化论将观察到的分集性归因于随机突变。由于一些突变的基因带给了拥有它们的个体更高的存活与繁衍的机会，随机突变保留在了基因库中。

生物体的特征在某种程度上是确定性地发生的（例如：在基因和环境的影响下），在某种程度上是随机发生的。例如，基因与曝光量仅仅支配着人体皮肤上出现的色斑密度；而单个色斑的精确位置看来是随机决定的。



模拟

模拟 (simulation) , 泛指基于实验或训练为目的, 将原本的事务或流程, 予以系统化与公式化, 以便进行可重现预期结果的模拟。

随机化模拟突变

- 研究 DNA 突变的机制
- 研究突变对相关蛋白质生物活性的影响
- 有助于进化、疾病以及 DNA 分裂和修复机制等基本细胞过程的研究



模拟

模拟 (simulation) , 泛指基于实验或训练为目的, 将原本的事务或流程, 予以系统化与公式化, 以便进行可重现预期结果的模拟。

随机化模拟突变

- 研究 DNA 突变的机制
- 研究突变对相关蛋白质生物活性的影响
- 有助于进化、疾病以及 DNA 分裂和修复机制等基本细胞过程的研究



- 在数组中随机选取索引，在字符串中随机选取位置
- 随机选取 DNA 中的一个核苷酸突变成其他（随机）的核苷酸
- 使用随机数来生成 DNA 序列数据集（可以用作对照数据集）
- 重复突变 DNA 来研究在进化过程中突变随时间累积的影响



教学提纲

1 引言

2 随机数生成器

3 随机化程序

4 模拟 DNA 突变

- 伪代码设计
- 改进设计
- 组合子程序
- bug?

5 生成随机 DNA

- 设计理念
- 伪代码
- 程序

6 分析 DNA

- 7 知识拓展
- 8 回顾和总结
- 总结
 - 思考题



随机数生成器

随机数生成器 (random number generator) 是通过一些算法、物理信号、环境噪音等来产生看起来似乎没有关联性的数列的方法或装置。丢硬币、掷骰子、洗牌就是生活上常见的随机数产生方式。

大部分计算机上的伪随机数，并不是真正的随机数，只是重复的周期比较大的数列，是按一定的算法和种子值生成的。

伪随机数

伪随机性 (pseudorandomness) 是指一个过程似乎是随机的，但实际上并不是。例如伪随机数（或称伪乱数），是使用一个确定性的算法计算出来的似乎是随机的数序，因此伪随机数实际上并不随机。在计算伪随机数时假如使用的开始值不变的话，那么伪随机数的数序也不变。伪随机数的随机性可以用它的统计特性来衡量，其主要特征是每个数出现的可能性和它出现时与数序中其它数的关系。伪随机数的优点是它的计算比较简单，而且只使用少数数值很难推算出计算它的算法。一般人们使用一个假的随机数，比如电脑上的时间作为计算伪随机数的开始值。

随机数生成器

随机数生成器 (random number generator) 是通过一些算法、物理信号、环境噪音等来产生看起来似乎没有关联性的数列的方法或装置。丢硬币、掷骰子、洗牌就是生活上常见的随机数产生方式。

大部分计算机上的伪随机数，并不是真正的随机数，只是重复的周期比较大的数列，是按一定的算法和种子值生成的。

伪随机数

伪随机性 (pseudorandomness) 是指一个过程似乎是随机的，但实际上并不是。例如伪随机数（或称伪乱数），是使用一个确定性的算法计算出来的似乎是随机的数序，因此伪随机数实际上并不随机。在计算伪随机数时假如使用的开始值不变的话，那么伪随机数的数序也不变。伪随机数的随机性可以用它的统计特性来衡量，其主要特征是每个数出现的可能性和它出现时与数序中其它数的关系。伪随机数的优点是它的计算比较简单，而且只使用少数数值很难推算出计算它的算法。一般人们使用一个假的随机数，比如电脑上的时间作为计算伪随机数的开始值。

- 随机数生成器输出的数字是伪随机数，并不是真正随机的
- 一个随机数生成器作为一种算法，是可以被预测的
- 随机数生成器需要一个种子（seed）作为输入，种子改变，伪随机数随之改变
- 初始化使用的种子本身应该是随机选择的



教学提纲

1 引言

2 随机数生成器

3 随机化程序

4 模拟 DNA 突变

- 伪代码设计
- 改进设计
- 组合子程序
- bug?

5 生成随机 DNA

- 设计理念
- 伪代码
- 程序

6 分析 DNA

7 知识拓展

8 回顾和总结

- 总结
- 思考题



突变和随机化 | 随机化程序 | 程序 7.1.1

```
1 #!/usr/bin/perl -w
2 # Example 7-1 Children's game, demonstrating
3 # primitive artificial intelligence,
4 # using a random number generator to randomly
5 # select parts of sentences.
6
7
8 # Declare the variables
9 my $count;
10 my $input;
11 #my $number;
12 my $sentence;
13 my $story;
```



突变和随机化 | 随机化程序 | 程序 7.1.2

```
15 # Here are the arrays of parts of sentences:  
16 my @nouns = (  
17     'Dad',      'TV',       'Mom',          'Groucho',  
18     'Rebecca',   'Harpo',    'Robin Hood',   'Joe and Moe  
19     ',  
20 );  
21 my @verbs = (  
22     'ran to',   'giggled with', 'put hot sauce into  
the orange juice of',  
23     'exploded', 'dissolved',   'sang stupid songs  
with',  
24     'jumped with',  
25 );
```



突变和随机化 | 随机化程序 | 程序 7.1.3

```
27 my @prepositions = (
28     'at the store',
29     'over the rainbow',
30     'just for the fun of it',
31     'at the beach',
32     'before dinner',
33     'in New York City',
34     'in a dream',
35     'around the world',
36 );
37
38 # Seed the random number generator.
39 # time|$$ combines the current time with the
40 # current process id
41 # in a somewhat weak attempt to come up with a
42 # random seed.
43 srand( time | $$ );
```



突变和随机化 | 随机化程序 | 程序 7.1.4

```
43 # This do-until loop composes six-sentence "stories".  
44 # until the user types "quit".  
45 do {  
46  
47     # (Re)set $story to the empty string each time through the loop  
48     $story = '';  
49  
50     # Make 6 sentences per story.  
51     for ( $count = 0 ; $count < 6 ; $count++ ) {
```



突变和随机化 | 随机化程序 | 程序 7.1.5

```
53      # Notes on the following statements:
54      # 1) scalar @array gives the number of elements
55      # in the array.
56      # 2) rand returns a random number greater than
57      # 0 and
58      # less than scalar(@array).
59      # 3) int removes the fractional part of a
60      # number.
61      # 4) . joins two strings together.
62      $sentence =
63          $nouns[ int( rand( scalar @nouns ) ) ] . " "
64          . $verbs[ int( rand( scalar @verbs ) ) ] . " "
65          . $nouns[ int( rand( scalar @nouns ) ) ] . " "
66          . $prepositions[ int( rand( scalar
@prepositions ) ) ] . '. ';
```

```
67
68      $story .= $sentence;
69  }
```



突变和随机化 | 随机化程序 | 程序 7.1.6

```
68 # Print the story.  
69 print "\n", $story, "\n";  
70  
71     # Get user input.  
72 print "\nType \"quit\" to quit, or press  
Enter to continue: ";  
73  
74 $input = <STDIN>;  
75  
76     # Exit loop at user's request  
77 } until ( $input =~ /^$q/i );  
78  
79 exit;
```



突变和随机化 | 随机化程序 | 程序 7.1 | 输出

- 1 Joe and Moe jumped with Rebecca in New York City.
Rebecca exploded Groucho in a dream. Mom ran to Harpo over the rainbow. TV giggled with Joe and Moe over the rainbow. Harpo exploded Joe and Moe at the beach. Robin Hood giggled with Harpo at the beach.
- 2
- 3 Type "quit" to quit, or press Enter to [continue](#):
- 4
- 5 Harpo put hot sauce into the orange juice of TV before dinner. Dad ran to Groucho in a dream. Joe and Moe put hot sauce into the orange juice of TV in New York City.
Joe and Moe giggled with Joe and Moe over the rainbow.
TV put hot sauce into the orange juice of Mom just [for](#) the fun of it. Robin Hood ran to Robin Hood at the beach.
- 6
- 7 Type "quit" to quit, or press Enter to [continue](#): quit



```
1 srand( time | $$ );  
2 srand(100);  
3 srand;
```

说明

- `srand;` 会自动设置种子
- `rand` 会自动调用 `srand;` 设置种子
- `time` 返回代表时间的数
- `$$` 返回运行的 Perl 程序的 PID (每次运行都会改变)
- `|` 表示位元的或运算 (bitwise OR) , 把两个数的位 (bit) 组合起来
- 逻辑或: $0 \text{ or } 0 = 0; 0 \text{ or } 1 = 1; 1 \text{ or } 0 = 1; 1 \text{ or } 1 = 1$

do-until

- 用途：在每次循环中采取任何行动（比如询问用户是否继续）之前就做一些事情
- 流程：首先执行代码块中的语句，然后进行测试，决定是否应该重复执行代码块中的语句
- 注意：流程和其他类型的循环（先测试后执行代码块）相反
- 重置：在特定的地方把某种形式递增的变量进行清空



- 使用一个名词、一个动词、一个名词和一个介词短语造句
- 用以空格分隔的句子片段构造一个字符串（一个句子）
- 用一个句点和空格将其结束（句子后面的句号和空格）
- 使用字符串连接操作符（点号）构建字符串



突变和随机化 | 随机化程序 | 随机选取数组元素

```
1 $verbs[ int( rand( scalar @verbs ) ) ]  
2 # 由内向外逐步进行阅读/计算/理解/分析  
3  
4 # STEP1  
5 scalar @verbs # scalar返回数组元素的个数, 7  
6 $verbs[ int( rand( 7 ) ) ]  
7  
8 # STEP2  
9 rand(7) # 返回>=0、<7的（伪）随机数（小数），3.47429  
10 $verbs[ int( 3.47429 ) ]  
11  
12 # STEP3  
13 int(3.47429) # 丢掉小数的小数部分、留下整数部分，3  
14  
15 # STEP4  
16 $verbs[3] # 给出数组@verbs的第4个元素
```



突变和随机化 | 随机化程序 | 格式化

```
1 $verbs[ int( rand( scalar @verbs ) ) ]  
2  
3 # 等价的语句 (更加易懂/繁琐)  
4 $verb_array_size = scalar @verbs;  
5 $random_floating_point = rand (  
    $verb_array_size );  
6 $random_integer = int $random_floating_point;  
7 $verb = $verbs[$random_integer];  
8  
9 $sentence = "$subject $verb $object  
$prepositional_phrase. ";
```



```
1 # 原程序中的写法
2 $verbs[ int( rand( scalar @verbs ) ) ]
3
4 # 没有小括号的写法
5 $verbs[ int rand scalar @verbs ]
6
7 # 更简单的写法
8 $verbs[rand @verbs]
9 # rand期望一个标量值，所以它会把@verbs放在标量上下文中（返回数组大小）
10 # 数组的下标总是整数值，所以当它需要下标时，会自动提取小数的整数部分（不需要int了）
```



教学提纲

- 1 引言
- 2 随机数生成器
- 3 随机化程序
- 4 模拟 DNA 突变
 - 伪代码设计
 - 改进设计
 - 组合子程序
 - bug?

- 5 生成随机 DNA
 - 设计理念
 - 伪代码
 - 程序
- 6 分析 DNA
- 7 知识拓展
- 8 回顾和总结
 - 总结
 - 思考题



教学提纲

- 1 引言
- 2 随机数生成器
- 3 随机化程序
- 4 模拟 DNA 突变
 - 伪代码设计
 - 改进设计
 - 组合子程序
 - bug?

- 5 生成随机 DNA
 - 设计理念
 - 伪代码
 - 程序
- 6 分析 DNA
- 7 知识拓展
- 8 回顾和总结
 - 总结
 - 思考题



把 DNA 序列中的一个随机位置上的核苷酸突变成一个随机核苷酸

- ① 选取 DNA 序列字符串中的一个随机位置 (x)
- ② 选择一个随机的核苷酸 (N)
- ③ 把 DNA 随机位置 (x) 上的核苷酸替换成选取的随机核苷酸 (N)



- 采取随机选取数组元素的策略
- length 返回字符串的长度
- 字符串中的位置是从 0 到 length-1 进行编号的（类似于数组元素的索引）



```
1 # randomposition
2 # A subroutine to randomly select a position
   in a string.
3 # WARNING: make sure you call srand to seed
   the random number generator before you call
   this function.
4
5 sub randomposition {
6     my($string) = @_;
7     # This expression returns a random number
   between 0 and length-1,
8     # which is how the positions in a string
   are numbered in Perl.
9     return int(rand(length($string)));
10 }
```



```
1 #!/usr/bin/perl -w
2 # Test the randomposition subroutine
3
4 my $dna = 'AACCGTTAATGGGCATCGATGCTATGCGAGCT';
5 srand(time|$$);
6 for (my $i=0 ; $i < 20 ; ++$i ) {
7     print randomposition($dna), " ";
8 }
9 print "\n";
10 exit;
11
12 sub randomposition {
13     my($string) = @_;
14     return int rand length $string;
15 }
```



```
1 # randomnucleotide
2 # A subroutine to randomly select a
   nucleotide
3 # WARNING: make sure you call srand to seed
   the random number generator before you call
   this function.
4
5 sub randomnucleotide {
6     my(@nucs) = @_;
7     # scalar returns the size of an array.
8     # The elements of the array are numbered 0
   to size-1
9     return $nucs[rand @nucs];
10 }
```



```
1 #!/usr/bin/perl -w
2 # Test the randomnucleotide subroutine
3
4 my @nucleotides = ('A', 'C', 'G', 'T');
5 srand(time|$$);
6 for (my $i=0 ; $i < 20 ; ++$i ) {
7     print randomnucleotide(@nucleotides), " ";
8 }
9 print "\n";
10 exit;
11
12 sub randomnucleotide {
13     my (@nucs) = @_;
14     return $nucs[rand @nucs];
15 }
```



突变和随机化 | 模拟 DNA 突变 | 设计 | 突变

```
1 # mutate
2 # A subroutine to perform a mutation in a string of DNA
3
4 sub mutate {
5     my($dna) = @_;
6     my(@nucleotides) = ('A', 'C', 'G', 'T');
7     # Pick a random position in the DNA
8     my($position) = randomposition($dna);
9     # Pick a random nucleotide
10    my($newbase) = randomnucleotide(@nucleotides);
11    # Insert the random nucleotide into the random
12    # position in the DNA.
13    # The substr arguments mean the following:
14    # In the string $dna at position $position change 1
15    # character to
16    # the string in $newbase
17    substr($dna,$position,1,$newbase);
18    return $dna;
19 }
```



- 在 for 循环中使用 my 对计数器变量进行声明
- 把所有变量放在程序的顶部统一进行声明：对阅读代码有帮助
- 在第一次使用变量的时候进行声明：更自然的方式



教学提纲

- 1 引言
- 2 随机数生成器
- 3 随机化程序
- 4 模拟 DNA 突变
 - 伪代码设计
 - 改进设计
 - 组合子程序
 - bug?

- 5 生成随机 DNA
 - 设计理念
 - 伪代码
 - 程序
- 6 分析 DNA
- 7 知识拓展
- 8 回顾和总结
 - 总结
 - 思考题



思考

- @nucleotides 只在子程序 randomnucleotide 中使用



突变和随机化 | 模拟 DNA 突变 | 改进设计

```
1 # randomnucleotide
2 # A subroutine to randomly select a
   nucleotide
3 # WARNING: make sure you call srand to seed
   the random number generator before you call
   this function.
4
5 sub randomnucleotide {
6     my(@nucs) = ('A', 'C', 'G', 'T');
7     # scalar returns the size of an array.
8     # The elements of the array are numbered 0
   to size-1
9     return $nucs[rand @nucs];
10 }
```



进一步思考

- 编写通用的从任意数组中随机选取一个元素的子程序



突变和随机化 | 模拟 DNA 突变 | 改进设计

```
1 # randomnucleotide
2 # A subroutine to randomly select a
   nucleotide
3 sub randomnucleotide {
4     my(@nucleotides) = ('A', 'C', 'G', 'T');
5     return randomelement(@nucleotides);
6 }
7
8 # randomelement
9 # A subroutine to randomly select an element
   from an array
10 sub randomelement {
11     my(@array) = @_;
12     return $array[rand @array];
13 }
```



总结

- 不更改 mutate 子程序，只改变 randomnucleotide 子程序的内部结构（而非行为）



教学提纲

- 1 引言
- 2 随机数生成器
- 3 随机化程序
- 4 模拟 DNA 突变
 - 伪代码设计
 - 改进设计
 - 组合子程序
 - bug?

- 5 生成随机 DNA
 - 设计理念
 - 伪代码
 - 程序
- 6 分析 DNA
- 7 知识拓展
- 8 回顾和总结
 - 总结
 - 思考题



突变和随机化 | 模拟 DNA 突变 | 组合 | 程序 7.2.1

```
1 #!/usr/bin/perl -w
2 # Example 7-2 Mutate DNA
3 # using a random number generator to
4 # randomly select bases to mutate
5
6 use strict;
7 use warnings;
8
9
10 # Declare the variables
11
12 # The DNA is chosen to make it easy to see
13 # mutations:
14 my $DNA = 'AAAAAAAAAAAAAAAAAAAAAAA';
```



突变和随机化 | 模拟 DNA 突变 | 组合 | 程序 7.2.2

```
13 # $i is a common name for a counter variable,  
#     short for "integer"  
14 my $i;  
15  
16 my $mutant;  
17  
18 # Seed the random number generator.  
19 # time|$$ combines the current time with the  
#   current process id  
20 srand( time | $$ );  
21  
22 # Let's test it, shall we?  
23 $mutant = mutate($DNA);  
24  
25 print "\nMutate DNA\n\n";
```



突变和随机化 | 模拟 DNA 突变 | 组合 | 程序 7.2.3

```
27 print "\nHere is the original DNA:\n\n";
28 print "$DNA\n";
29
30 print "\nHere is the mutant DNA:\n\n";
31 print "$mutant\n";
32
33 # Let's put it in a loop and watch that bad
   boy accumulate mutations:
34 print "\nHere are 10 more successive
   mutations:\n\n";
35
36 for ( $i = 0 ; $i < 10 ; ++$i ) {
37     $mutant = mutate($mutant);
38     print "$mutant\n";
39 }
40
```



突变和随机化 | 模拟 DNA 突变 | 组合 | 程序 7.2.4

```
54 sub mutate {  
55  
56     my ($dna) = @_;  
57  
58     my ($position) = randomposition($dna);  
59  
60     #my (@nucleotides) = ( 'A', 'C', 'G', 'T'  
61     );  
62     #my ($newbase) = randomnucleotide(  
63     @nucleotides);  
64     my ($newbase) = randomnucleotide();  
65  
66     substr( $dna, $position, 1, $newbase );  
67  
}
```



突变和随机化 | 模拟 DNA 突变 | 组合 | 程序 7.2.5

```
80 sub randomelement {  
81  
82     my (@array) = @_;  
83  
84     return $array[ rand @array ];  
85 }
```



```
94 sub randomnucleotide {  
95  
96     my (@nucleotides) = ( 'A', 'C', 'G', 'T'  
97 );  
98  
99     # scalar returns the size of an array.  
100    # The elements of the array are numbered  
101    0 to size-1  
100    return randomelement(@nucleotides);  
101 }
```



突变和随机化 | 模拟 DNA 突变 | 组合 | 程序 7.2.7

```
110 sub randomposition {
111     my ($string) = @_;
112     # $string is the argument to length
113     # length($string) is the argument to rand
114     # rand(length($string)) is the argument to int
115     # int(rand(length($string))) is the argument to
116     # return
117
118     # rand returns a decimal number between 0 and its
119     # argument.
120     # int returns the integer portion of a decimal
121     # number.
122
123     # The whole expression returns a random number
124     # between 0 and length-1,
125     # which is how the positions in a string are
126     # numbered in Perl.
127     return int rand length $string;
128 }
```



```
1 Mutate DNA
2 Here is the original DNA:
3 AAAAAAAAAAAAAA
4 Here is the mutant DNA:
5 AAAAAAAAAAAAAAGAAAAAAA
6
7 Here are 10 more successive mutations:
8 AAAAAAAAAAAAAAGACAAAAAA
9 AAAAAAAAAAAAAAGACAAAAAA
10 AAAAAAAAAAAAAAGACAAAAAA
11 AAAAAAAAAAAACAAAAAGACAAAAAA
12 AAAAAAAAAAAACAACAAGACAAAAAA
13 AAAAAAAAAAAACAACAAGACAAAAAA
14 AAAAAAAAGAAAACAACAAGACAAAAAA
15 AAAATAAGAAAACAACAAGACAAAAAA
16 AAAATAAGAAAACAACAAGACAAAAAA
```



教学提纲

- 1 引言
- 2 随机数生成器
- 3 随机化程序
- 4 模拟 DNA 突变
 - 伪代码设计
 - 改进设计
 - 组合子程序
 - bug?

- 5 生成随机 DNA
 - 设计理念
 - 伪代码
 - 程序
- 6 分析 DNA
- 7 知识拓展
- 8 回顾和总结
 - 总结
 - 思考题



设计思想

使用随机选取的碱基来替换随机位置上的碱基

替换自身

- 如果随机位置上的碱基和随机选取的碱基是一样的呢？
- 实际就是用一个碱基替换了它本身 (1/4)



设计思想

使用随机选取的碱基来替换随机位置上的碱基

替换自身

- 如果随机位置上的碱基和随机选取的碱基是一样的呢？
- 实际就是用一个碱基替换了它本身 (1/4)



```
1 Select a random position in the string of DNA
2
3 Repeat:
4
5     Choose a random nucleotide
6
7         Until: random nucleotide differs from the
8             nucleotide in the random position
9
10            Substitute the random nucleotide into the
11                random position in the DNA
```



突变和随机化 | 模拟 DNA 突变 | bug? | 修正

```
1 # mutate_better
2 # Subroutine to perform a mutation in a string of DNA--
3 # version 2, in which it is guaranteed that one base will
4 # change on each call
5
6 sub mutate_better {
7     my($dna) = @_;
8     #my(@nucleotides) = ('A', 'C', 'G', 'T');
9     my($position) = randomposition($dna);
10    my($newbase);
11    do {
12        #$newbase = randomnucleotide(@nucleotides);
13        $newbase = randomnucleotide();
14        # Make sure it's different than the nucleotide we're
15        # mutating
16        } until ( $newbase ne substr($dna, $position, 1));
17        substr($dna, $position, 1, $newbase);
18        return $dna;
19 }
```



```
1 Mutate DNA
2 Here is the original DNA:
3 AAAAAA
4 Here is the mutant DNA:
5 AAAAAAAATA
6
7 Here are 10 more successive mutations:
8 AAAAAAAATAACAAAAA
9 AAAATAACAAAAA
10 AAATATAACAAAAA
11 AAATATAACACAAAAA
12 AATTATAACACAAAAA
13 AATTATTACACAAAAA
14 AATTATTACACACACAA
15 AATTATTACACACACAA
16 AATTATTACACACACAA
```



- 对于只在代码块中使用的变量：在代码块中对其进行声明，把它隐藏在代码块中
- 在代码块外也会使用的变量：在代码块外进行声明
- 可以在程序/代码块顶部收集所有变量的声明



- 1 引言
- 2 随机数生成器
- 3 随机化程序
- 4 模拟 DNA 突变
 - 伪代码设计
 - 改进设计
 - 组合子程序
 - bug?

- 5 生成随机 DNA
 - 设计理念
 - 伪代码
 - 程序
- 6 分析 DNA
- 7 知识拓展
- 8 回顾和总结
 - 总结
 - 思考题



目的

- 生成一系列长短不一的随机 DNA 片段

要求

- 设定最大长度
- 设定最小长度
- 设定 DNA 片段的数目



目的

- 生成一系列长短不一的随机 DNA 片段

要求

- 设定最大长度
- 设定最小长度
- 设定 DNA 片段的数目



- 1 引言
- 2 随机数生成器
- 3 随机化程序
- 4 模拟 DNA 突变
 - 伪代码设计
 - 改进设计
 - 组合子程序
 - bug?

- 5 生成随机 DNA
 - 设计理念
 - 伪代码
 - 程序
- 6 分析 DNA
- 7 知识拓展
- 8 回顾和总结
 - 总结
 - 思考题



自上而下 (Top-down)

从主程序开始，当需要子程序时再进行编写。

自下而上 (Bottom-up)

从最基本的砖瓦（子程序）开始，逐步组装成高楼大厦（主程序）。



自上而下 (Top-down)

从主程序开始，当需要子程序时再进行编写。

自下而上 (Bottom-up)

从最基本的砖瓦（子程序）开始，逐步组装成高楼大厦（主程序）。



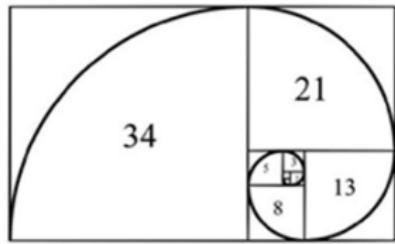
费波那契数列

费波那契数列 (Successione di Fibonacci) , 又译为费波拿契数、费氏数列、黄金分割数列。

在数学上，费波那契数列是以递归的方法来定义：

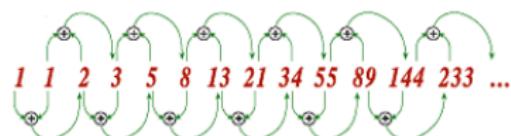
- $F_0 = 0$
- $F_1 = 1$
- $F_n = F_{n-1} + F_{n-2}$ ($n \geq 2$)

用文字来说，就是费波那契数列由 0 和 1 开始，之后的费波那契系数就是由之前的两数相加而得出。



0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144...

$$\begin{aligned}0 + 1 &= 1 \\1 + 1 &= 2 \\2 + 1 &= 3 \\3 + 2 &= 5 \\5 + 3 &= 8 \\8 + 5 &= 13 \\13 + 8 &= 21 \\21 + 13 &= 34 \\34 + 21 &= 55 \\55 + 34 &= 89 \\89 + 55 &= 144\end{aligned}$$



Top-down vs. bottom-up

divide and conquer is a horrible way of finding Fibonacci numbers

- the recursive calls are NOT independent; redundancies build up

```
public static int fib(int n) {  
    if (n <= 1) {  
        return 1;  
    }  
    else {  
        return fib(n-1) + fib(n-2);  
    }  
}
```

```
          fib(5)  
          fib(4)      +      fib(3)  
fib(3) + fib(2)      fib(2) + fib(1)  
          .      .      .  
          .      .      .  
          .      .      .
```

in this case, a bottom-up solution
makes more sense

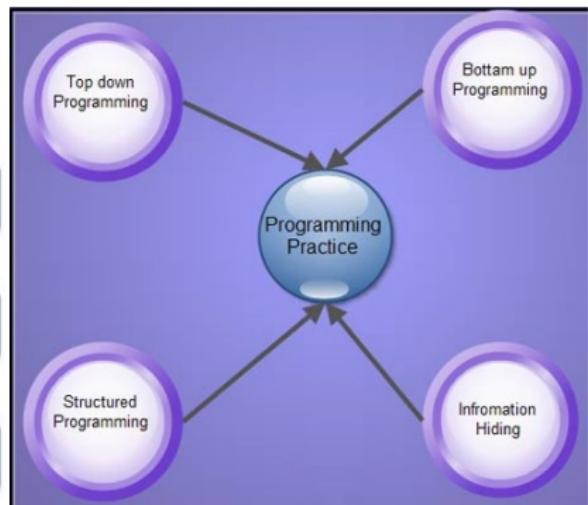
- start at the base cases (the bottom) and work up to the desired number
- requires remembering the previous two numbers in the sequence

```
public static int fib(int n) {  
    int prev = 1, current = 1;  
    for (int i = 1; i < n; i++) {  
        int next = prev + current;  
        prev = current;  
        current = next;  
    }  
    return current;  
}
```





Top Down and Bottom Up Design



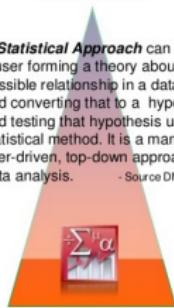
突变和随机化 | 生成随机 DNA | 设计理念

Statistics vs. Data Mining

Top-Down Query, Search

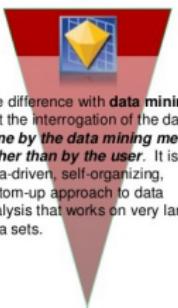
A Statistical Approach can involve a user forming a theory about a possible relationship in a database and converting that to a hypothesis and testing that hypothesis using a statistical method. It is a manual, user-driven, top-down approach to data analysis.

- Source DM Review



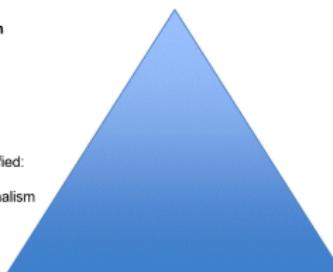
Bottom-Up Data Mining, Text Mining

The difference with **data mining** is that the interrogation of the data is **done by the data mining method rather than by the user**. It is a data-driven, self-organizing, bottom-up approach to data analysis that works on very large data sets.



Top-Down Theory-driven approach

Examples of concepts identified:
Good Customer Service,
Poor management, Professionalism



Data

Examples of concepts identified:
High Cost, Unresponsiveness, Love



Bottom-Up DATA-driven approach



- 1 引言
- 2 随机数生成器
- 3 随机化程序
- 4 模拟 DNA 突变
 - 伪代码设计
 - 改进设计
 - 组合子程序
 - bug?

- 5 生成随机 DNA
 - 设计理念
 - 伪代码
 - 程序
- 6 分析 DNA
- 7 知识拓展
- 8 回顾和总结
 - 总结
 - 思考题



目标

生成长短不一的随机 DNA

途径

使用子程序 make_random_DNA_set 直接生成数据

```
1 @random_DNA = make_random_DNA_set(  
    $minimum_length, $maximum_length,  
    $size_of_set );
```



目标

生成长短不一的随机 DNA

途径

使用子程序 `make_random_DNA_set` 直接生成数据

```
1 @random_DNA = make_random_DNA_set(  
    $minimum_length, $maximum_length,  
    $size_of_set );
```



目标

生成长短不一的随机 DNA

途径

使用子程序 make_random_DNA_set 直接生成数据

```
1 @random_DNA = make_random_DNA_set(  
    $minimum_length, $maximum_length,  
    $size_of_set );
```



目标

编写 make_random_DNA_set 子程序

```
1 repeat $size_of_set times:  
2  
3     $length = random number between minimum and  
4         maximum length  
5  
6     $dna = make_random_DNA ( $length );  
7  
8     add $dna to @set  
9  
10 return @set
```



目标

编写 make_random_DNA_set 子程序

```
1 repeat $size_of_set times:  
2  
3     $length = random number between minimum and  
4         maximum length  
5     $dna = make_random_DNA ( $length );  
6  
7     add $dna to @set  
8 }  
9  
10 return @set
```



目标

编写 make_random_DNA 子程序

```
1 from 1 to $length  
2  
3     $base = randomnucleotide  
4  
5     $dna .= $base  
6 }  
7  
8 return $dna
```

目标

已经编写完了 randomnucleotide 子程序

目标

编写 make_random_DNA 子程序

```
1 from 1 to $length  
2  
3     $base = randomnucleotide  
4  
5     $dna .= $base  
6 }  
7  
8 return $dna
```

目标

已经编写完了 randomnucleotide 子程序

目标

编写 make_random_DNA 子程序

```
1 from 1 to $length
2
3     $base = randomnucleotide
4
5     $dna .= $base
6 }
7
8 return $dna
```

目标

已经编写完了 randomnucleotide 子程序

- 1 引言
- 2 随机数生成器
- 3 随机化程序
- 4 模拟 DNA 突变
 - 伪代码设计
 - 改进设计
 - 组合子程序
 - bug?

- 5 生成随机 DNA
 - 设计理念
 - 伪代码
 - 程序
- 6 分析 DNA
- 7 知识拓展
- 8 回顾和总结
 - 总结
 - 思考题



突变和随机化 | 生成随机 DNA | 程序 7.3.1

```
1 #!/usr/bin/perl -w
2 # Example 7-3 Generate random DNA
3 # using a random number generator to
4 # randomly select bases
5
6 use strict;
7 use warnings;
8
9 # Declare and initialize the variables
10 my $size_of_set      = 12;
11 my $maximum_length  = 30;
12 my $minimum_length  = 15;
```



突变和随机化 | 生成随机 DNA | 程序 7.3.2

```
13 # An array, initialized to the empty list, to
   store the DNA in
14 my @random_DNA = ();
15
16 # Seed the random number generator.
17 # time|$$ combines the current time with the
   current process id
18 srand( time | $$ );
19
20 # And here's the subroutine call to do the
   real work
21 @random_DNA =
22   make_random_DNA_set( $minimum_length,
   $maximum_length, $size_of_set );
```



突变和随机化 | 生成随机 DNA | 程序 7.3.3

```
24 # Print the results, one per line
25 print "Here is an array of $size_of_set
26     randomly generated DNA sequences\n";
27 print " with lengths between $minimum_length
28     and $maximum_length:\n\n";
29
30 foreach my $dna (@random_DNA) {
31
32     print "$dna\n";
33 }
34
35 print "\n";
36
37 exit;
```



突变和随机化 | 生成随机 DNA | 程序 7.3.4

```
51 sub make_random_DNA_set {  
52  
53     # Collect arguments, declare variables  
54     my ( $minimum_length, $maximum_length,  
$size_of_set ) = @_;  
55  
56     # length of each DNA fragment  
57     my $length;  
58  
59     # DNA fragment  
60     my $dna;  
61  
62     # set of DNA fragments  
63     my @set;
```



突变和随机化 | 生成随机 DNA | 程序 7.3.5

```
65      # Create set of random DNA
66      for ( my $i = 0 ; $i < $size_of_set ; ++$i ) {
67
68          # find a random length between min and max
69          $length = randomlength( $minimum_length,
70                                $maximum_length );
71
72          # make a random DNA fragment
73          $dna = make_random_DNA($length);
74
75          # add $dna fragment to @set
76          push( @set, $dna );
77
78      return @set;
79 }
```



突变和随机化 | 生成随机 DNA | 程序 7.3.6

```
94 sub randomlength {  
95  
96     # Collect arguments, declare variables  
97     my ( $minlength, $maxlength ) = @_;  
98  
99     # Calculate and return a random number within  
the  
100    # desired interval.  
101    # Notice how we need to add one to make the  
endpoints inclusive,  
102    # and how we first subtract, then add back,  
$minlength to  
103    # get the random number in the correct  
interval.  
104    return ( int( rand( $maxlength - $minlength +  
1 ) ) + $minlength );  
105 }
```



突变和随机化 | 生成随机 DNA | 程序 7.3.7

```
114 sub make_random_DNA {  
115  
116     # Collect arguments, declare variables  
117     my ($length) = @_;  
118  
119     my $dna;  
120  
121     for ( my $i = 0 ; $i < $length ; ++$i ) {  
122  
123         $dna .= randomnucleotide();  
124     }  
125  
126     return $dna;  
127 }
```



突变和随机化 | 生成随机 DNA | 程序 7.3.8

```
140 sub randomnucleotide {  
141  
142     my (@nucleotides) = ( 'A', 'C', 'G', 'T'  
143 );  
144  
145     # scalar returns the size of an array.  
146     # The elements of the array are numbered  
147     0 to size-1  
148     return randomelement(@nucleotides);  
149 }
```



突变和随机化 | 生成随机 DNA | 程序 7.3.9

```
156 sub randomelement {  
157  
158     my (@array) = @_;  
159  
160     return $array[ rand @array ];  
161 }
```



突变和随机化 | 生成随机 DNA | 程序 7.3 | 输出

```
1 Here is an array of 12 randomly generated DNA
2 sequences
3 with lengths between 15 and 30:
4 TACGCTTGTGTTTCGGGGGAC
5 GGGGTGTGGTAAGGCTGTCTCAGATGTGC
6 TGAACGACAACCTCCTGGACTTTACT
7 ATCTATGCCATTGCTAGT
8 CCGCTCATTCCCTCTCCTCGGC
9 TGTACCCCTAATAACACTTAGCCGAATTAA
10 ATAGGTGGGGCGACAGCGCCGG
11 GATTGACCTCTGTAA
12 AAAATCTCTAGGATCGAGC
13 GTATGTGCTTGGGTAAAT
14 ATGGAGTTGCGAGGAAGTAGCTGAGT
15 GCCCATGACCAGCATCCAGACAGCA
```



- 1 引言
- 2 随机数生成器
- 3 随机化程序
- 4 模拟 DNA 突变
 - 伪代码设计
 - 改进设计
 - 组合子程序
 - bug?

- 5 生成随机 DNA
 - 设计理念
 - 伪代码
 - 程序
- 6 分析 DNA
 - 7 知识拓展
 - 8 回顾和总结
 - 总结
 - 思考题



突变和随机化 | 分析 DNA | 问题

A comparison of part of the mouse and fly genes
(identical regions are highlighted)

mouse gene: GTATCCAACGGTTGTGTGAGTAAAATTCTGGGCAGGTATTACGAGACTGGCTCCATCAGA

fly gene: GTATCAAATGGATGTGTGAGCAAAATTCTCGGGAGGTATTATGAAACAGGAAGCATACGA

These gene sequences are 76.66% similar.

一般性问题

两条 DNA 的相似性如何？（对于两个随机的 DNA 序列，平均来说，它们的碱基相同的百分比是多少？）

具体问题

- 对于两条等长的 DNA 序列，同一位置碱基相同的百分比是多少？
- 对于一个 DNA 序列集来说，上述百分比的平均值是多少？

突变和随机化 | 分析 DNA | 问题

A comparison of part of the mouse and fly genes (identical regions are highlighted)

mouse gene: GTATCCAACGGTTGTGTGAGTAAAATTCTGGGCAGGTATTACGAGACTGGCTCCATCAGA

fly gene: GTATCAAATGGATGTGTGAGCAAAATTCTCGGGAGGTATTATGAAACAGGAAGCATACGA

These gene sequences are 76.66% similar.

一般性问题

两条 DNA 的相似性如何？（对于两个随机的 DNA 序列，平均来说，它们的碱基相同的百分比是多少？）

具体问题

- 对于两条等长的 DNA 序列，同一位置碱基相同的百分比是多少？
- 对于一个 DNA 序列集来说，上述百分比的平均值是多少？

```
1 Generate a set of random DNA sequences, all  
the same length  
2  
3 For each pair of DNA sequences  
4  
5     How many positions in the two sequences are  
identical as a fraction?  
6  
7 }  
8  
9 Report the mean of the preceding calculations  
as a percentage
```



突变和随机化 | 分析 DNA | 伪代码 | 比较相同位置上的核苷酸

```
1 assuming DNA1 is the same length as DNA2,  
2  
3 for each position from 1 to length(DNA)  
4  
5 if the character at that position is the  
6 same in DNA_1 and DNA_2  
7  
8     ++$count  
9 }  
10  
11 return count/length
```



突变和随机化 | 分析 DNA | 程序 7.4.1

```
1 #!/usr/bin/perl -w
2 # Example 7-4 Calculate the average
3 # percentage of positions that are the same
4 # between two random DNA sequences, in a set
5 # of 10 sequences.
6
7
8 # Declare and initialize the variables
9 my $percent;
10 my @percentages;
11 my $result;
```



突变和随机化 | 分析 DNA | 程序 7.4.2

```
13 # An array, initialized to the empty list, to
  store the DNA in
14 my @random_DNA = ();
15
16 # Seed the random number generator.
17 # time|$$ combines the current time with the
  current process id
18 srand( time | $$ );
19
20 # Generate the data set of 10 DNA sequences.
21 @random_DNA = make_random_DNA_set( 10, 10, 10
  );
```



突变和随机化 | 分析 DNA | 程序 7.4.3

```
23 # Iterate through all pairs of sequences
24 for ( my $k = 0 ; $k < scalar @random_DNA - 1
25     ; ++$k ) {
26     for ( my $i = ( $k + 1 ) ; $i < scalar
27         @random_DNA ; ++$i ) {
28
29         # Calculate and save the matching
30         # percentage
31         $percent = matching_percentage(
32             $random_DNA[$k], $random_DNA[$i] );
33         push( @percentages, $percent );
34     }
35 }
```



突变和随机化 | 分析 DNA | 程序 7.4.4

```
33 # Finally, the average result:  
34 $result = 0;  
35  
36 foreach $percent (@percentages) {  
37     $result += $percent;  
38 }  
39  
40 $result = $result / scalar(@percentages);  
41  
42 #Turn result into a true percentage  
43 $result = int( $result * 100 );  
44 print "In this run of the experiment, the  
    average percentage of \n";  
45 print "matching positions is $result%\n\n";  
46  
47 exit;
```



突变和随机化 | 分析 DNA | 程序 7.4.5

```
58 sub matching_percentage {  
59     my ( $string1, $string2 ) = @_;  
60  
61     # we assume that the strings have the  
62     # same length  
63     my ($length) = length($string1);  
64     my ($position);  
65     my ($count) = 0;
```



突变和随机化 | 分析 DNA | 程序 7.4.6

```
67     for ( $position = 0 ; $position < $length  
68     ; ++$position ) {  
69         if (  
70             substr( $string1, $position, 1 )  
eq substr( $string2, $position, 1 )  
71         )  
72         {  
73             ++$count;  
74         }  
75     }  
76     return $count / $length;  
77 }
```



突变和随机化 | 分析 DNA | 程序 7.4.7

```
89 sub make_random_DNA_set {  
90  
91     # Collect arguments, declare variables  
92     my ( $minimum_length, $maximum_length,  
$size_of_set ) = @_;  
93  
94     # length of each DNA fragment  
95     my $length;  
96  
97     # DNA fragment  
98     my $dna;  
99  
100    # set of DNA fragments  
101    my @set;
```



突变和随机化 | 分析 DNA | 程序 7.4.8

```
103     # Create set of random DNA
104     for ( my $i = 0 ; $i < $size_of_set ; ++$i ) {
105
106         # find a random length between min and max
107         $length = randomlength( $minimum_length,
108             $maximum_length );
109
110         # make a random DNA fragment
111         $dna = make_random_DNA($length);
112
113         # add $dna fragment to @set
114         push( @set, $dna );
115
116     }
117 }
```



突变和随机化 | 分析 DNA | 程序 7.4.9

```
127 sub randomlength {  
128  
129     # Collect arguments, declare variables  
130     my ( $minlength, $maxlength ) = @_;  
131  
132     # Calculate and return a random number within  
the  
133     # desired interval.  
134     # Notice how we need to add one to make the  
endpoints inclusive,  
135     # and how we first subtract, then add back,  
$minlength to  
136     # get the random number in the correct  
interval.  
137     return ( int( rand( $maxlength - $minlength +  
1 ) ) + $minlength );  
138 }
```



突变和随机化 | 分析 DNA | 程序 7.4.10

```
147 sub make_random_DNA {  
148  
149     # Collect arguments, declare variables  
150     my ($length) = @_;  
151  
152     my $dna;  
153  
154     for ( my $i = 0 ; $i < $length ; ++$i ) {  
155         $dna .= randomnucleotide();  
156     }  
157  
158     return $dna;  
159 }
```



突变和随机化 | 分析 DNA | 程序 7.4.11

```
168 sub randomnucleotide {  
169  
170     my (@nucleotides) = ( 'A', 'C', 'G', 'T'  
171 );  
172  
173     # scalar returns the size of an array.  
174     # The elements of the array are numbered  
175     0 to size-1  
176     return randomelement(@nucleotides);  
177 }
```



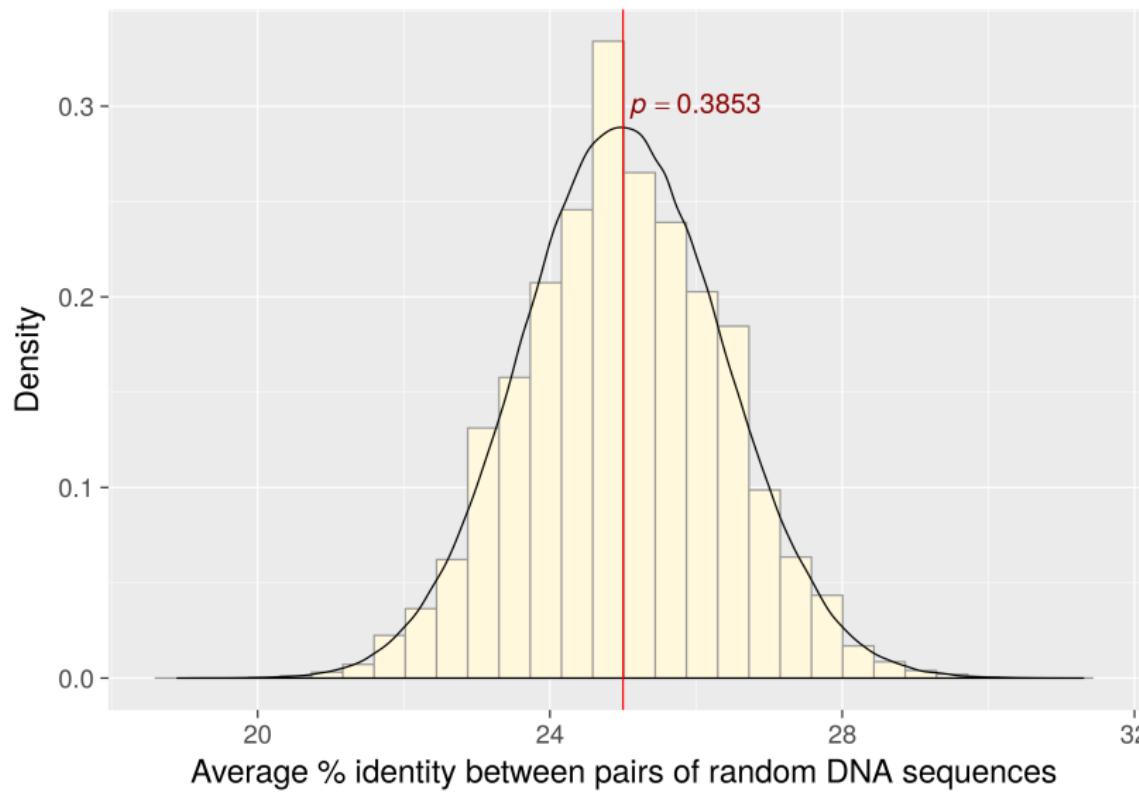
突变和随机化 | 分析 DNA | 程序 7.4.12

```
184 sub randomelement {  
185  
186     my (@array) = @_;  
187  
188     return $array[ rand @array ];  
189 }
```



- 1 In this run of the experiment, the average number of
- 2 matching positions is 24%





```
1 @random_DNA = make_random_DNA_set( 10, 10, 10  
);
```

```
1 my $minimum_length = 10;  
2 my $maximum_length = 10;  
3 my $size_of_set      = 10;  
4  
5 @random_DNA = make_random_DNA_set(  
    $minimum_length, $maximum_length,  
    $size_of_set );
```



```
1 @random_DNA = make_random_DNA_set( 10, 10, 10  
);
```

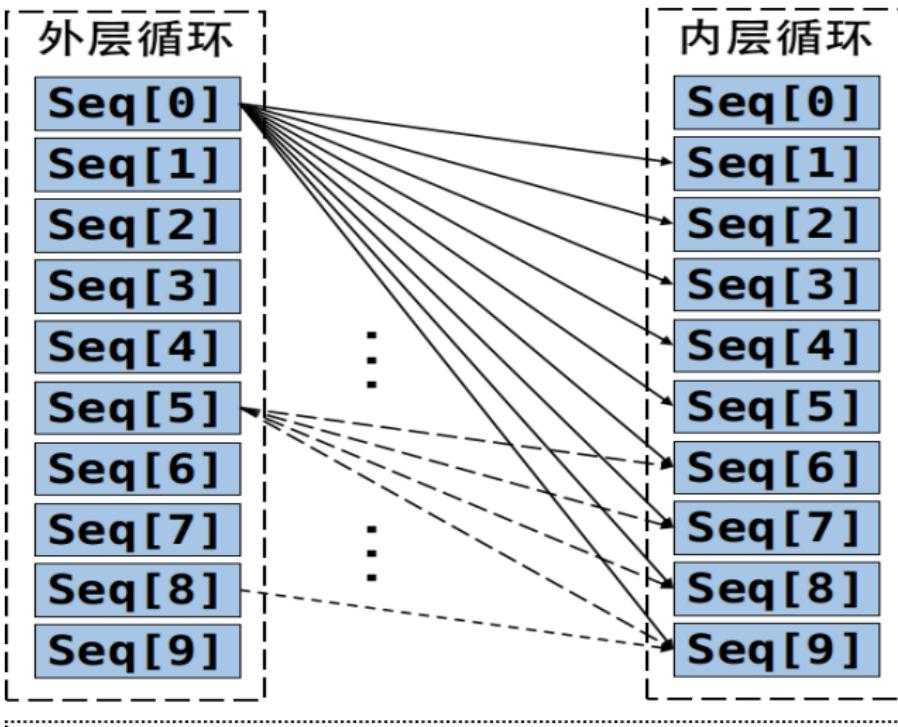
```
1 my $minimum_length = 10;  
2 my $maximum_length = 10;  
3 my $size_of_set      = 10;  
4  
5 @random_DNA = make_random_DNA_set(  
    $minimum_length, $maximum_length,  
    $size_of_set );
```



突变和随机化 | 分析 DNA | 程序 7.4 | 嵌套循环

```
1 # Iterate through all pairs of sequences
2 for (my $k = 0 ; $k < scalar @random_DNA - 1
3     ; ++$k) {
4     for (my $i = ($k + 1) ; $i < scalar
5          @random_DNA ; ++$i) {
6
7         # Calculate and save the matching
8         # percentage
9         $percent = matching_percentage(
10            $random_DNA[$k], $random_DNA[$i]);
11         push(@percentages, $percent);
12     }
13 }
```





突变和随机化 | 分析 DNA | 程序 7.4 | 嵌套循环

Nested Loops	Output	Explanation
<pre>for (i = 1; i <= 3; i++) { for (j = 1; j <= 4; j++) { Print "*" } System.out.println(); }</pre>	***** ***** *****	Prints 3 rows of 4 asterisks each.
<pre>for (i = 1; i <= 4; i++) { for (j = 1; j <= 3; j++) { Print "*" } System.out.println(); }</pre>	*** *** *** ***	Prints 4 rows of 3 asterisks each.



突变和随机化 | 分析 DNA | 程序 7.4 | 嵌套循环

Nested Loops	Output	Explanation
<pre>for (i = 1; i <= 4; i++) { for (j = 1; j <= i; j++) { Print "*" } System.out.println(); }</pre>	<pre>* ** *** ****</pre>	Prints 4 rows of lengths 1, 2, 3, and 4.
<pre>for (i = 1; i <= 3; i++) { for (j = 1; j <= 5; j++) { if (j % 2 == 0) { Print "*" } else { Print "-" } } System.out.println(); }</pre>	<pre>-*-*- -*-*- -*-*-</pre>	Prints asterisks in even columns, dashes in odd columns.
<pre>for (i = 1; i <= 3; i++) { for (j = 1; j <= 5; j++) { if ((i + j) % 2 == 0) { Print "*" } else { Print " " } } System.out.println(); }</pre>	<pre>* * * * * * * *</pre>	Prints a checkerboard pattern.



突变和随机化 | 分析 DNA | 程序 7.4 | 嵌套循环

```
for(int i = 0; i<10; i++)
{
    for(int j = 0; j<10; j++)
    {
        if(j>i)
            System.out.print("* ");
        else
            System.out.print("  ");
    }
    System.out.println();
}
```

```
for(int i = 0; i<10; i++)
{
    for(int j = 0; j<10; j++)
    {
        if(((i-j)/2)*2 == i-j)
            System.out.print("* ");
        else
            System.out.print("  ");
    }
    System.out.println();
}
```

```
* * * * * * * * *
* * * * * * * * *
* * * * * * * * *
* * * * * * * * *
* * * * * * * * *
* * * * * * * * *
* * * * * * * * *
* * * * * * * * *
* * * * * * * * *
* * * * * * * * *
```

```
* * * * *
* * * * *
* * * * *
* * * * *
* * * * *
* * * * *
* * * * *
* * * * *
* * * * *
* * * * *
```

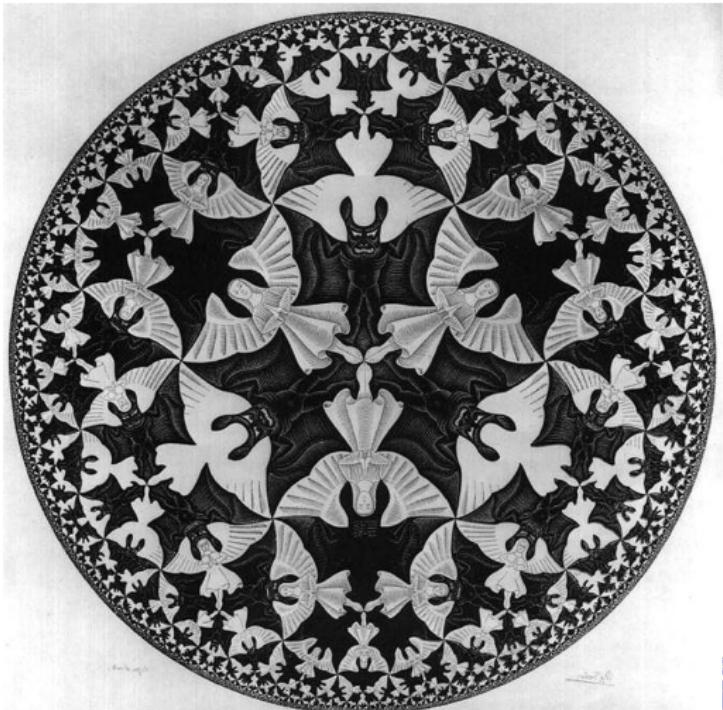
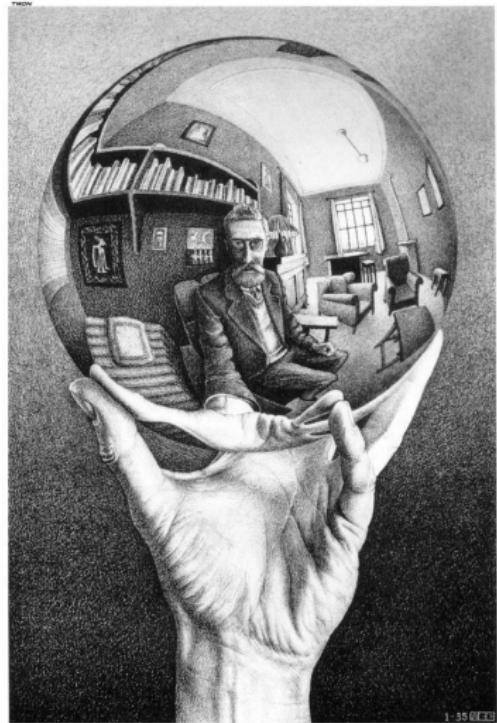


- 1 引言
- 2 随机数生成器
- 3 随机化程序
- 4 模拟 DNA 突变
 - 伪代码设计
 - 改进设计
 - 组合子程序
 - bug?

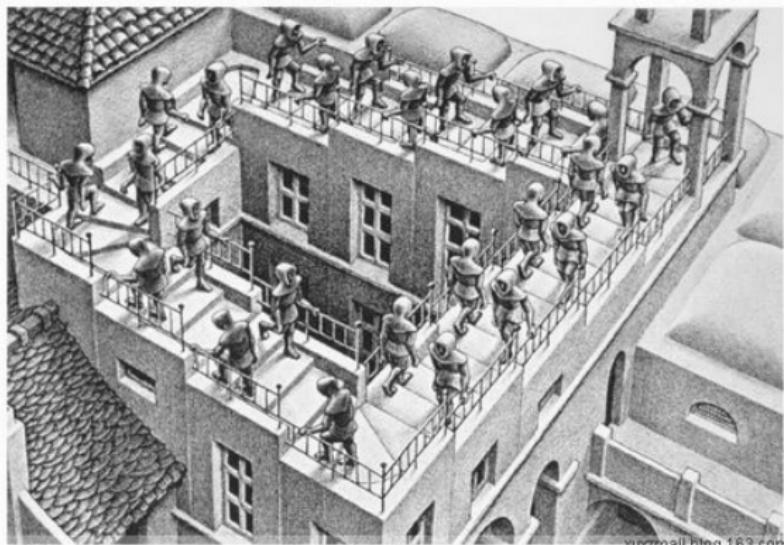
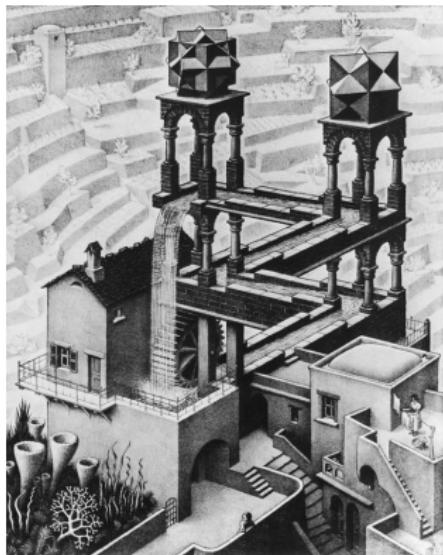
- 5 生成随机 DNA
 - 设计理念
 - 伪代码
 - 程序
- 6 分析 DNA
- 7 知识拓展
- 8 回顾和总结
 - 总结
 - 思考题



突变和随机化 | 知识拓展 | 嵌套



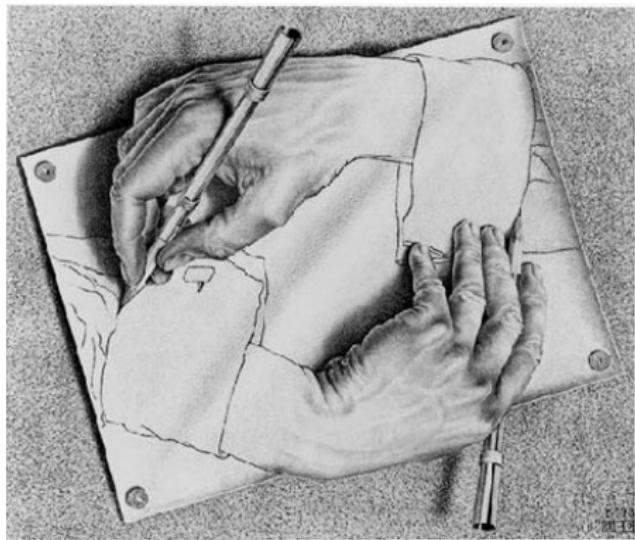
突变和随机化 | 知识拓展 | 死循环



xuxzmail.blog.163.com



突变和随机化 | 知识拓展 | 递归





《哥德尔、埃舍尔、巴赫》

《哥德尔、埃舍尔、巴赫：集异璧之大成》（Gödel, Escher, Bach: an Eternal Golden Braid），是一本赢得普立兹奖的书。它是侯世达的著作，由 Basic Books 出版社在 1979 年出版的。这本书的二十周年版本在 1999 年发行，而且由侯世达加上新的前言。《集异璧之大成》（ISBN 7-100-01323-2）是商务印书馆在 1996 年出版的根据 1995 年英文版翻译的中文版。

本书的英文副标题意译为“一条永恒的金带”，其首字母与哥德尔、埃舍尔、巴赫三人的英文名字首字母 GEB 相同，而商务印书馆中文译本的副标题中的“集异璧”则与 GEB 谐音。本书一共有两篇，上篇译为“集异璧 GEB”，下篇译为“异集璧 EGB”。

本书主要讲述了逻辑学家哥德尔，艺术家埃舍尔，和作曲家巴赫的创造性的成就怎样交织在一起。正如作者所说：“我认识到，哥德尔、埃舍尔和巴赫只是用不同的方式来表达一样相同的本质。我尝试重现这种本质而写出这本书。”

侯世达

侯世达因其著作《哥德尔、埃舍尔、巴赫》获得普立兹奖（非小说类别）和美国国家图书奖（科学类别）。

侯世达定律

侯世达定律（Hofstadter's law）是一句自指的格言，由侯世达在《哥德尔、埃舍尔、巴赫》一书中提出：

侯世达定律：做事所花费的时间总是比你预期的要长，即使你的预期中考虑了侯世达定律。——侯世达，《哥德尔、埃舍尔、巴赫》

侯世达定律指做复杂任务需要花费的时间总是很难预计的。程序员经常会引用这一定律，特别是在进行有关提高效率的讨论时（如《人月神话》和极限编程）。其自指的特征反映了即便意识到任务的复杂性，预计花费的时间仍是困难的。

```
1 #!/usr/bin/perl
2
3 use strict;
4 use warnings;
5
6 sub F {
7     my $n = shift;
8     return 0 if $n == 0;
9     return 1 if $n == 1;
10    return F( $n - 1 ) + F( $n - 2 );
11 }
12
13 print F( $ARGV[0] ), "\n";
```



```
1 #!/usr/bin/perl
2
3 use strict; use warnings;
4
5 {
6     my %fib;
7     sub F {
8         my $n = shift;
9         return 0 if $n == 0;
10        return 1 if $n == 1;
11        if ( not exists $fib{$n} ) {
12            $fib{$n} = F( $n - 1 ) + F( $n - 2 );
13        }
14        return $fib{$n};
15    }
16 }
17
18 print F( $ARGV[0] ), "\n";
```



```
1 #!/usr/bin/perl
2
3 use strict; use warnings;
4 use Memoize;
5
6 memoize('F');
7
8 sub F {
9     my $n = shift;
10    return 0 if $n == 0;
11    return 1 if $n == 1;
12    return F( $n - 1 ) + F( $n - 2 );
13 }
14
15 print F( $ARGV[0] ), "\n";
```



图标和随机化 | 知识拓展 | 斐波那契数列 | 总结

```
1 # 1. Compute Fibonacci numbers
2 sub fib {
3     my $n = shift;
4     return $n if $n < 2;
5     fib($n-1) + fib($n-2);
6 }
7 # 2. Compute Fibonacci numbers, memoized version
8 { my @fib;
9  sub fib {
10      my $n = shift;
11      return $fib[$n] if defined $fib[$n];
12      return $fib[$n] = $n if $n < 2;
13      $fib[$n] = fib($n-1) + fib($n-2);
14  }
15 }
16 # 3. Compute Fibonacci numbers, module version
17 use Memoize;
18 memoize('fib');
```



教学提纲

1 引言

2 随机数生成器

3 随机化程序

4 模拟 DNA 突变

- 伪代码设计

- 改进设计

- 组合子程序

- bug?

5

生成随机 DNA

- 设计理念

- 伪代码

- 程序

6

分析 DNA

7

知识拓展

8

回顾和总结

- 总结

- 思考题



教学提纲

1 引言

2 随机数生成器

3 随机化程序

4 模拟 DNA 突变

- 伪代码设计

- 改进设计

- 组合子程序

- bug?

5

生成随机 DNA

- 设计理念

- 伪代码

- 程序

6

分析 DNA

7

知识拓展

8

回顾和总结

- 总结

- 思考题



知识点

- 随机：数组的随机元素，字符串的随机位置，两个整数间的随机数
- 随机数生成器：伪随机，种子
- 程序设计理念：自上而下，自下而上
- 其他：do-until，变量声明，嵌套循环

技能

- 熟练使用 Perl 语言中的随机数生成器
- 熟练分割任务、设计子程序
- 熟练使用自下而上和自上而下的理念设计程序
- 能够用 Perl 语言编写 DNA 突变相关的程序



知识点

- 随机：数组的随机元素，字符串的随机位置，两个整数间的随机数
- 随机数生成器：伪随机，种子
- 程序设计理念：自上而下，自下而上
- 其他：do-until，变量声明，嵌套循环

技能

- 熟练使用 Perl 语言中的随机数生成器
- 熟练分割任务、设计子程序
- 熟练使用自下而上和自上而下的理念设计程序
- 能够用 Perl 语言编写 DNA 突变相关的程序

教学提纲

- 1 引言
- 2 随机数生成器
- 3 随机化程序
- 4 模拟 DNA 突变
 - 伪代码设计
 - 改进设计
 - 组合子程序
 - bug?

- 5 生成随机 DNA
 - 设计理念
 - 伪代码
 - 程序
- 6 分析 DNA
- 7 知识拓展
- 8 回顾和总结
 - 总结
 - 思考题



- ① 在 Perl 语言中如何设置随机数生成器的种子？
- ② 如何随机选取数组的元素？
- ③ 如何随机选取字符串的位置？
- ④ 如何随机选取两个整数间的一个数字？
- ⑤ 比较自下而上和自上而下两种设计理念。
- ⑥ 解释嵌套循环的工作步骤。



下节预告

计算机

回顾总结常见的排序和查找算法。

生物

回顾遗传密码、翻译和阅读框的相关知识。



Powered by



T_EX L^AT_EX X_ET_EX Beamer