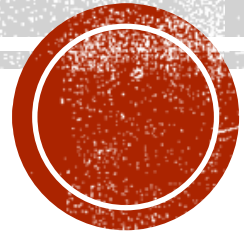
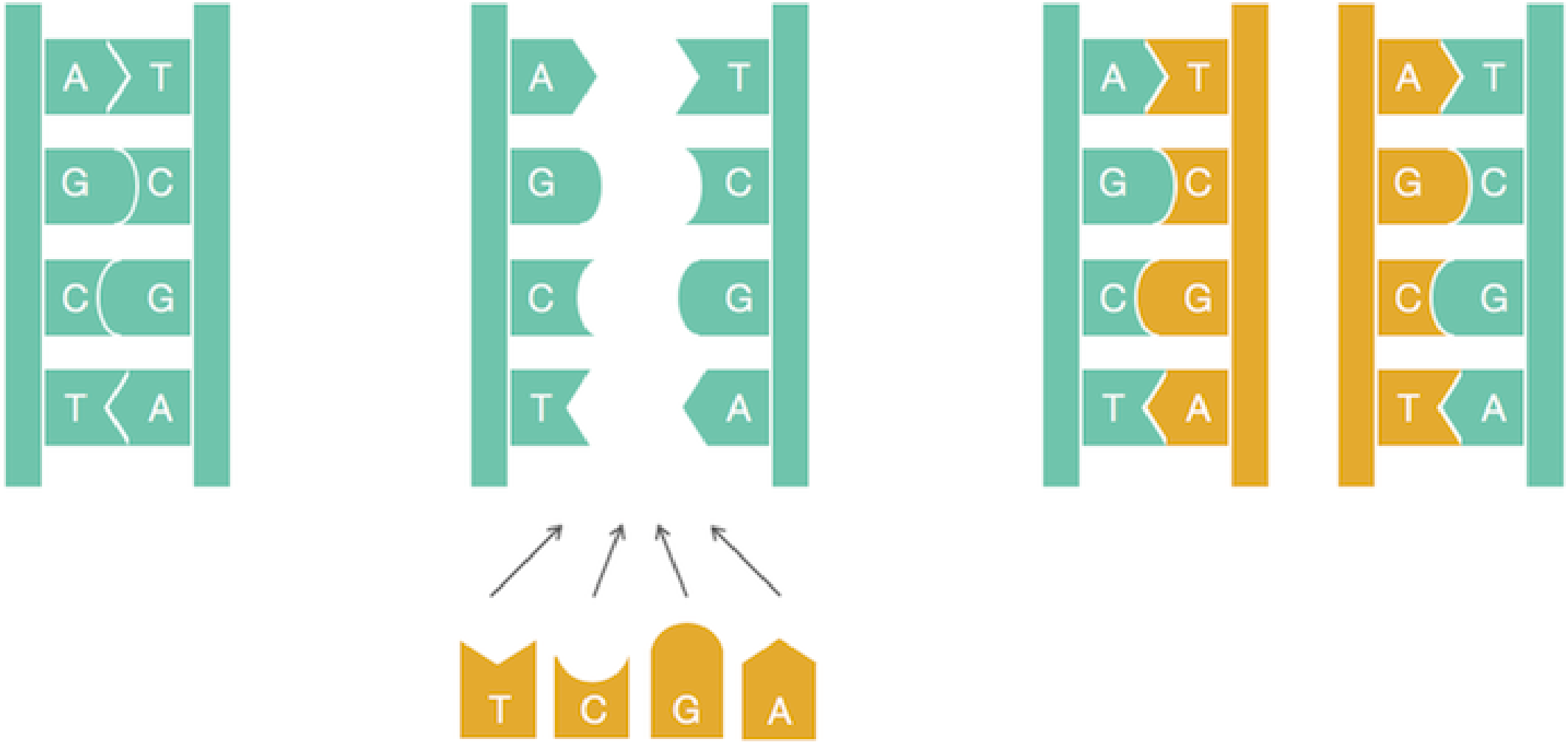
ПОДГОТОВКА К ЕГЭ ПО ИНФОРМАТИКЕ



Задачи на строки

12 марта 2023 г.

БИОИНФОРМАТИКА



https://www.bioinformaticsalgorithms.org

БИОИНФОРМАТИКА

Фрагмент ДНК холерного вибриона:

atcaatgatcaacgtaagcttctaagcatgatcaaggtgctcacacagtttatccacaac
ctgagtggatgacatcaagataggtcgttgtatctccttcctctcgtactctcatgacca
cggaaagatgatcaagagaggatgatttcttggccatatcgcaatgaatacttgtgactt
gtgcttccaattgacatcttcagcgccatattgcgctggccaaggtgacggagcgggatt
acgaaagcatgatcatggctgttgttctgtttatcttgttttgactgagacttgttagga
tagacggtttttcatcactgactagccaaagccttactctgcctgacatcgaccgtaaat
tgataatgaatttacatgcttccgcgacgatttacctcttgatcatcgatccgattgaag
atcttcaattgttaattctcttgcctcgactcatagccatgatgagctcttgatcatgtt
tccttaaccctctatttttacggaagaatgatcaagctgctgctcttgatcatcgtttc

НАИБОЛЕЕ ЧАСТАЯ ПОДСТРОКА ДЛИНЫ K

Пусть n – длина строки text, k – длина подстрок.

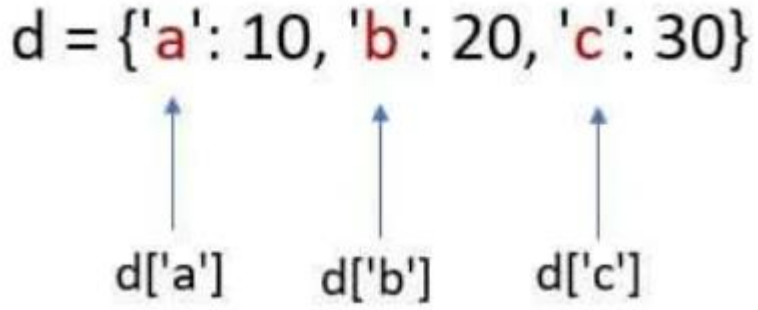
Алгоритм

```
1.  cnts ← пустой словарь
2.  для i ← 0 до n - k делай
3.    tmpl ← text[i:i+k-1]
4.    если tmpl нет в cnts то
5.      cnts[tmpl] ← 1
6.    иначе
7.      cnts[tmpl] ← cnts[tmpl] + 1
8.  вернуть ключи cnts с максимальным значением
```

НАИБОЛЕЕ ЧАСТАЯ ПОДСТРОКА ДЛИНЫ K

```python
def FrequentWords(text, k):
    cnts = {}
    n = len(text)
    for pos in range(n - k + 1):
        Pattern = text[pos:pos + k]
        if Pattern not in cnts:
            cnts[Pattern] = 1
        else:
            cnts[Pattern] += 1
    max_count = max(cnts.values())
    result = []
    for key, value in cnts.items():
        if value == max_count:
            result.append(key)
    return sorted(result)
```

Структура данных «словарь» в питоне



1. ПРЕОБРАЗОВАНИЕ СТРОКИ ДНК В ЧИСЛО

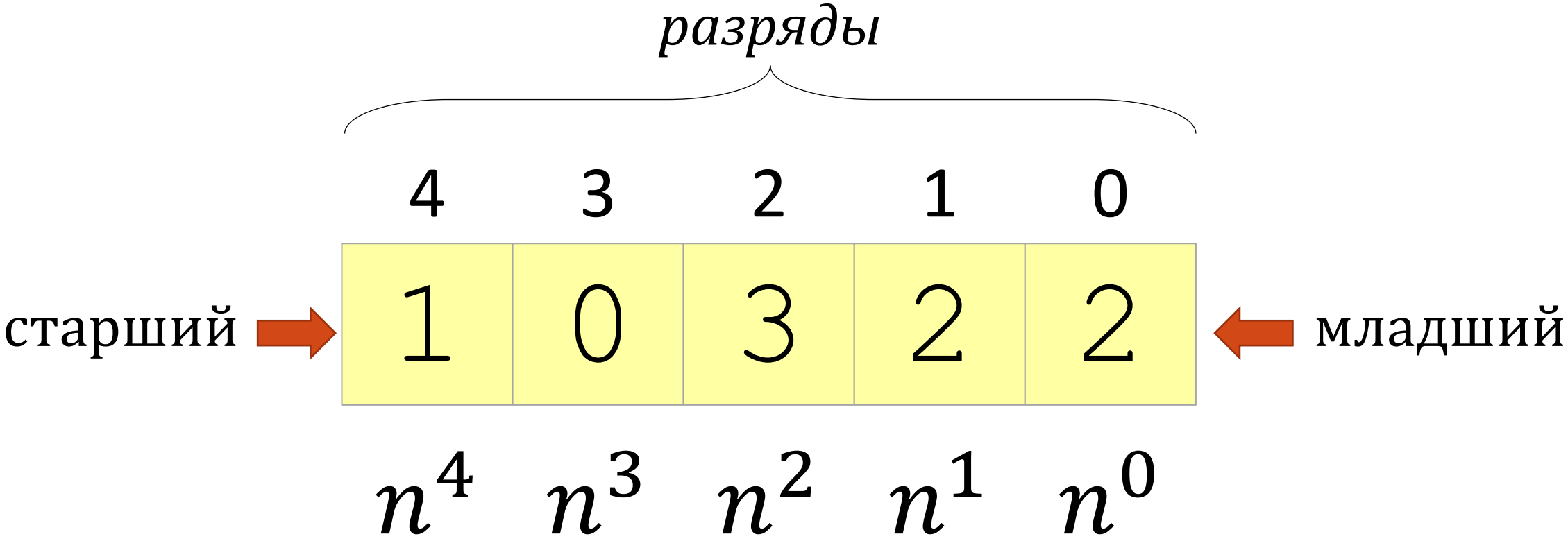
Мы можем сопоставить буквам A, C, G, T, соответственно, цифры 0, 1, 2 и 3. Тогда цепочку нуклеотидов можно трактовать как число, записанное в четверичной системе счисления. Реализуйте функцию `PatternToNumber(dna)`, которая выполняет преобразование строки ДНК в десятичное число.

Вход: Строка dna.

Выход: Десятичное число, соответствующее dna.

СИСТЕМЫ СЧИСЛЕНИЯ

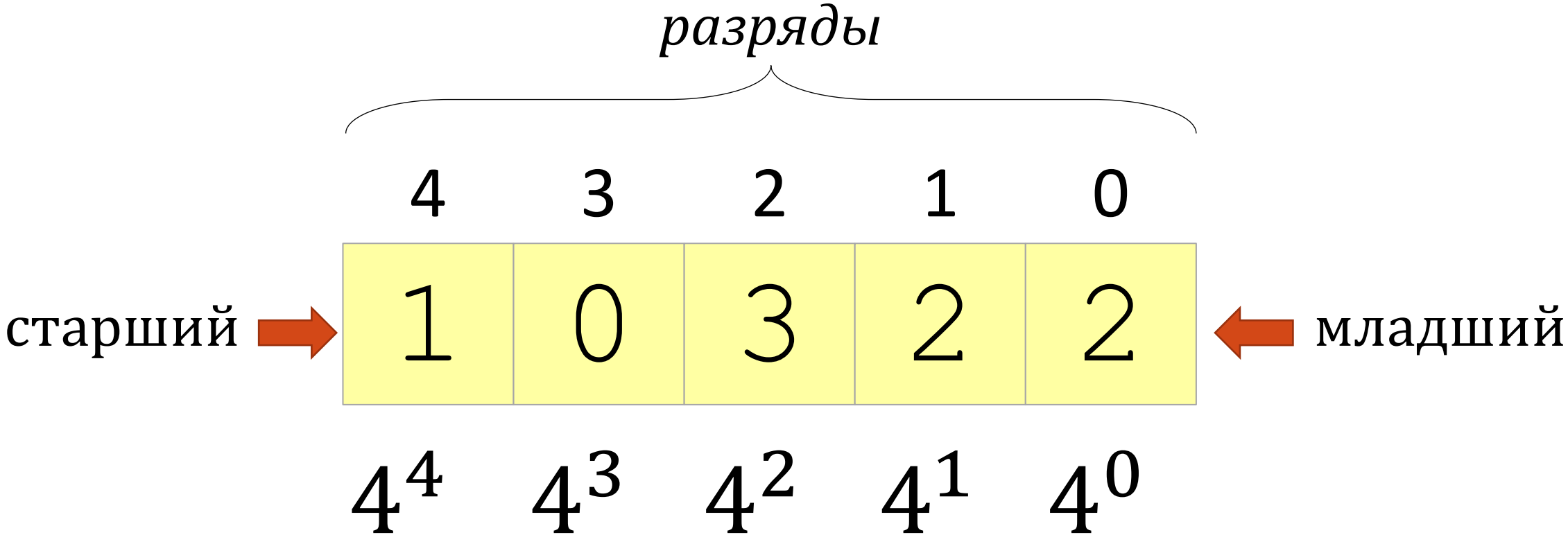
Строка **CATGG**



$$1n^4 + 0n^3 + 3n^2 + 2n^1 + 2n^0$$

СИСТЕМЫ СЧИСЛЕНИЯ

Строка **CATGG**



$$1 \cdot 256 + 0 \cdot 64 + 3 \cdot 16 + 2 \cdot 4 + 2 \cdot 1 = 314$$

СИСТЕМЫ СЧИСЛЕНИЯ

Строка $\mathbf{CATGG}_4$
или 10322_4

$$1n^4 + 0n^3 + 3n^2 + 2n^1 + 2 =$$
$$(1n^3 + 0n^2 + 3n^1 + 2)n + 2 = \cdots$$

В скобах получили число на один разряд меньше.

Пример: $1024 = 102 * 10 + 4$, в свою очередь $102 = 10 * 10 + 2$

То же самое в других системах счисления:
Число $\mathbf{CATGG}_4$ равно $\mathbf{CATG}_4 * 4_{10} + \mathbf{G}_4$,
в свою очередь число $\mathbf{CATG}_4$ равно $\mathbf{CAT}_4 * 4_{10} + \mathbf{G}_4$

1. ПРЕОБРАЗОВАНИЕ СТРОКИ ДНК В ЧИСЛО

Пусть |text| – длина строки text.

Алгоритм Pattern2Number(text)

```
1.  если |text| = 0 то
2.    вернуть 0
3.  chr ← последний символ text
4.  удаляем из text последний символ
5.  вернуть Pattern2Number(text) * 4 + число(chr)
```

Здесь число(chr) – функция, возвращающая числа 0, 1, 2 и 3 по символам A, C, G и T, соответственно.

1. ПРЕОБРАЗОВАНИЕ СТРОКИ ДНК В ЧИСЛО

```
1  def char(letter):
2      letters = {'A': 0, 'C': 1, 'G': 2, 'T': 3}
3      return letters[letter]
4
5
6  def pattern_to_number(dna):
7      if dna == '':
8          return 0
9      return pattern_to_number(dna[:-1]) * 4 + char(dna[-1])
```

2. ПРЕОБРАЗОВАНИЕ ЧИСЛА В СТРОКУ ДНК

Реализуйте функцию `NumberToPattern(index, k)`, которая переводит число index в четверичную систему счисления, используя в качестве цифр 0, 1, 2 и 3 символы A, C, G, T, соответственно. Обратим внимание, что, например, строкам AGGT и GGT соответствует одно и то же число 43. Справиться с этой неоднозначностью помогает параметр k, задающий количество разрядов формируемого числа (т.е. длину результирующей цепочки ДНК).

Вход: Числа **index** и **k**.

Выход: Строка dna.

СИСТЕМЫ СЧИСЛЕНИЯ

Умножение числа на n^x сдвигает на **x** разрядов **влево** запись этого числа в **n**-ричной системе счисления

Пример 3

$156_{10} \cdot 10^5 = 15600000_{10}$

Пример 4

$156_{10} \cdot 2^5 = 4992_{10}$

получаем:

$156_{10} = 10011100_2, 4992_{10} = 1001110000000$

СИСТЕМЫ СЧИСЛЕНИЯ

Целочисленное деление числа на n^x сдвигает на **x** разрядов **вправо** запись этого числа в **n**-ричной системе счисления

Пример 5

$$\frac{156010_{10}}{10^3} = 156_{10}$$

Пример 6

$$\frac{4572_{10}}{2^5} = 142_{10} \text{ (целая часть) и } 28_{10} \text{ (остаток)}$$

получаем:

$$4572_{10} = 1000111011100_2, 142_{10} = 10001110, 28_{10} = 11100_2$$

2. ПРЕОБРАЗОВАНИЕ ЧИСЛА В СТРОКУ ДНК

Получаем алгоритм

n	Quotient$(n, 4)$	Remainder$(n, 4)$	NumberToSymbol
9904	2476	0	A
2476	619	0	A
619	154	3	T
154	38	2	G
38	9	2	G
9	2	1	C
2	0	2	G

2. ПРЕОБРАЗОВАНИЕ ЧИСЛА В СТРОКУ ДНК

Алгоритм Number2Pattern(num, k)

```
1.  если k = 1 то
2.    вернуть символ(num)
3.  q ← результат целочисленного деления num на 4
4.  r ← остаток от деления num на 4
5.  вернуть Number2Pattern(q, k-1) + символ(r)
```

Здесь символ(num) – функция, возвращающая символы A, C, G и T по числам 0, 1, 2 и 3, соответственно.

В строка 5 операция + это конкатенация двух строк.

2. ПРЕОБРАЗОВАНИЕ ЧИСЛА В СТРОКУ ДНК

```
1   def symbol(num):
2       numbers = {0: 'A', 1: 'C', 2: 'G', 3: 'T'}
3       return numbers[num]
4
5
6   def number_to_pattern(num, k):
7       if k == 1:
8           return symbol(num)
9       q = num // 4
10      r = num % 4
11      return number_to_pattern(q, k-1) + symbol(r)
```

3. МАССИВ ЧАСТОТ ПОДСТРОК

Реализуйте функцию `ComputingFrequencies(genome, k)`, вычисляющую массив частот для всех строк из нуклеотидов длины k. Данная функция возвращает список из 4^k чисел, где i-тое число обозначает, сколько раз строка i встречается в строке genome.

Вход: Строка genome и число k.

Выход: Массив частот подстрок длины k в строке genome. Все числа разделены одним пробелом.

3. МАССИВ ЧАСТОТ ПОДСТРОК

Алгоритм ComputingFrequencies(text, k)

```
1.  для i ← 0 до 4^k - 1 делай
2.    frequencyArray[i] ← 0
3.  для i ← 0 до |text| - k делай
4.    pattern ← text[i:i+k-1]
5.    num ← Pattern2Number(pattern)
6.    frequencyArray[num] ← frequencyArray[num] + 1
7.  вернуть frequencyArray
```

3. МАССИВ ЧАСТОТ ПОДСТРОК

```
1   def char(letter):...
4
5   def pattern_to_number(dna):...
9
10  def compute_frequencies(genome, k):
11      frequencies = [0] * 4 ** k
12
13      for i in range(len(genome) - k + 1):
14          word = genome[i:i+k]
15          n = pattern_to_number(word)
16          frequencies[n] += 1
17
18      return frequencies
```

4. НАИБОЛЕЕ ЧАСТАЯ ПОДСТРОКА (БЫСТРО)

Реализуйте функцию `FasterFrequentWords(text, k)`, которая находит в строке наиболее частую подстроку заданного размера с помощью массива частот (используйте функцию из упр. 3). Ответ выдайте в лексикографическом порядке.

Вход: Строка text и число k.

Выход: Все наиболее частые подстроки длины k из строки text, отсортированные в лексикографическом порядке.

4. НАИБОЛЕЕ ЧАСТАЯ ПОДСТРОКА ДЛИНЫ K

Алгоритм FasterFrequentWords(text, k)

```
1.  result ← пустой список
2.  frequencyArray ← ComputingFrequencies(text, k)
3.  maxCount ← максимум в массиве frequencyArray
4.  для i ← 0 до 4^k - 1 делай
5.    если frequencyArray[i] = maxCount то
6.      Pattern ← Number2Pattern(i, k)
7.      добавить Pattern в result
8.  вернуть result
```

4. НАИБОЛЕЕ ЧАСТАЯ ПОДСТРОКА ДЛИНЫ K

```
1   def char(letter):...
4   def pattern_to_number(dna):...
8   def symbol(num):...
11  def number_to_pattern(num, k):...
17  def compute_frequencies(genome, k):...
26
27  def find_frequent_words(genome, k):
28      frequencies = compute_frequencies(genome, k)
29      maximum = max(frequencies)
30      result = []
31
32      for i in range(len(frequencies)):
33          if frequencies[i] == maximum:
34              result.append(number_to_pattern(i, k))
35
36      return result
```

5. ПОИСК ГРУПП

Будем говорить, что подстрока длины k образует (L,t)-группу в строке genome, если существует интервал из L символов в строке genome, в котором данная подстрока встречается не менее t раз. Например, TGCA образует (25,3)-группу в следующей строке genome:

gatcagcataagggtccCTGCAATGCATGACAAGCCTGCAGTtgttttac

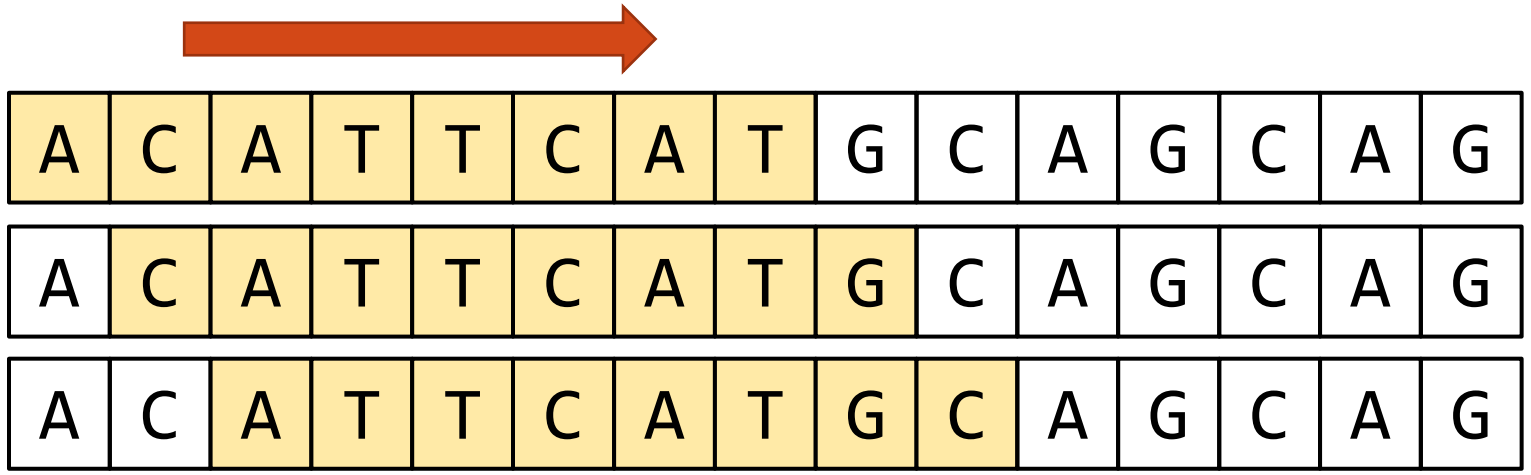
Реализуйте функцию `BetterClumpFinding(genome, k, t, L)`.

Вход: Строка genome и числа k, t и L.

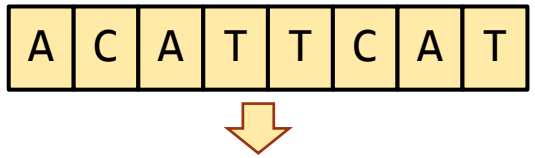
Выход: Все уникальные подстроки длины k, образующие (L,t)-группы в строке Genome, отсортированные в алфавитном порядке.

5. ПОИСК ГРУПП (ИДЕЯ АЛГОРИТМА)

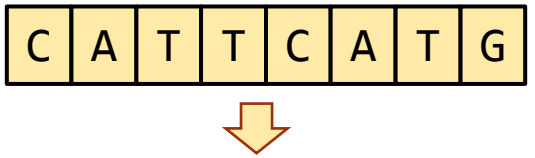
1) Окно размером **L** символов смещаем вправо шаг за шагом



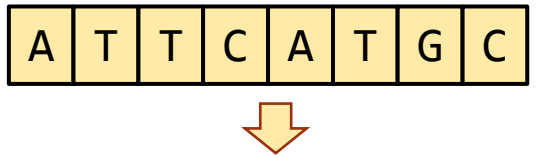
2) Для каждой полученной подстроки строим словарь частот



Слово	Частота
AA	0
AC	1
AG	0
…	…



Слово	Частота
AA	0
AC	0
AG	0
…	…



Слово	Частота
AA	0
AC	0
AG	0
…	…

3) Если в любом из словарей какая-либо подстрока встречается не менее **t** раз, запоминаем ее.

5. ПОИСК ГРУПП

Алгоритм ClumpFinding(text, k, t, L)

```
1.  result ← пустой список
2.  для i ← 0 до 4^k - 1 делай
3.    clump[i] ← ложь
4.  для i ← 0 до |text| - L делай
5.    template ← text[i:i+L-1]
6.    frequencyArray ← ComputingFrequencies(template, k)
7.    для j ← 0 до 4^k - 1 делай
8.      если frequencyArray[j] >= t то
9.        clump[j] ← истина
10. для i ← 0 до 4^k - 1 делай
11.   если clump[i] = истина то
12.     добавить Number2Pattern(i, k) в result
13. вернуть result
```

clump – это **массив меток**, clump[i] == True означает, что строка с номером i входит в (L,t)-группу.

5. ПОИСК ГРУПП

Очень долго!!!

Алгоритм ClumpFinding(text, k, t, L)

```
1.  result ← пустой список
2.  для i ← 0 до 4^k - 1 делай
3.    clump[i] ← ложь
4.  для i ← 0 до |text| - L делай
5.    template ← text[i:i+L-1]
6.    frequencyArray ← ComputingFrequencies(template, k)
7.    для j ← 0 до 4^k - 1 делай
8.      если frequencyArray[j] >= t то
9.        clump[j] ← истина
10. для i ← 0 до 4^k - 1 делай
11.   если clump[i] = истина то
12.     добавить Number2Pattern(i, k) в result
13. вернуть result
```

5. ПОИСК ГРУПП

Пусть text=**AA**GCAAAGGTGG**GC**, k = 2, L = 13.

text[0:L] = **AA**GCAAAGGTGG**G**, text[1:L+1] = **A**GCAAAGGTGG**GC**

k-mer	**AA**	AC	AG	AT	CA	CC	CG	CT	GA	**GC**	GG	GT	TA	TC	TG	TT
INDEX	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
frequency	**3**	0	2	0	1	0	0	0	0	**1**	3	1	0	0	1	0
frequency'	**2**	0	2	0	1	0	0	0	0	**2**	3	1	0	0	1	0

То есть в результате смещения окна на один шаг вправо, одна подстрока пропадет (k-префикс старой строки), а одна добавится (k-суффикс новой строки).

5. ЭФФЕКТИВНЫЙ ПОИСК ГРУПП

Алгоритм BetterClumpFinding(text, k, t, L)

```
1.  result ← пустой список
2.  frequencyArray ← ComputingFrequencies(text[0:L-1], k)
3.  для i ← 0 до 4^k - 1 делай
4.    если frequencyArray[i] >= t то clump[i] ← истина
5.    иначе clump[i] ← ложь
6.  для i ← 1 до |text| - L делай
7.    prefNum ← Pattern2Number(text[i-1:i+k-2])
8.    suffNum ← Pattern2Number(text[i+L-k:i+L-1])
9.    frequencyArray[prefNum] ← frequencyArray[prefNum] – 1
10.   frequencyArray[suffNum] ← frequencyArray[suffNum] + 1
11.   если frequencyArray[suffNum] >= t то clump[suffNum] ← истина
12. для i ← 0 до 4^k - 1 делай
13.   если clump[i] = истина то
14.     добавить Number2Pattern(i, k) в result
15. вернуть result
```

5. ЭФФЕКТИВНЫЙ ПОИСК ГРУПП

```
1   ...
26  def better_clump_finding(genome, k, t, L):
27      clump = [False] * 4 ** k
28      result = []
29      frequencies = compute_frequencies(genome[:L], k)
30
31      for i in range(4 ** k):
32          if frequencies[i] >= t: clump[i] = True
33
34      for i in range(1, len(genome) - L + 1):
35          prefix = pattern_to_number(genome[i-1:i+k-1])
36          suffix = pattern_to_number(genome[i+L-k+1:i+L+1])
37          frequencies[prefix] -= 1
38          frequencies[suffix] += 1
39          if frequencies[suffix] >= t: clump[suffix] = True
40
41      for i in range(4 ** k):
42          if clump[i]: result.append(number_to_pattern(i, k))
43
44      return result
```

ПРОДОЛЖЕНИЕ

Основы Python на Академии Яндекса:

https://academy.yandex.ru/handbook/python