Алгоритмика для развлечений

Рекомендация к выполнению

- помним про оформление кода;
- хотелось бы посоветовать как можно большее использование стандартной библиотеки C++, в том числе – контейнерных типов (**vector, map, set, list, stack, queue**) и всяких прикольных штук из **<algorithm>**. Да и про **string** не забываем.

Задания

4.1. Перевести заданное целое число в римскую систему счисления.

Символ	Число
I	1
V	5
X	10
L	50
C	100
D	500
M	1000

В целом, римские числа записываются начиная с больших символов к меньшим. Так, число XXXVIII можно расписать как XXX+V+III, что при переводе даёт $30+5+3=38$. Но имеются следующие исключения:

- символ «I» может стоять перед «V» и «X» для обозначения чисел 4 («IV») и 9 («IX»);
- символ «X» может стоять перед «L» и «C» для обозначения чисел 40 («XL») и 90 («XC»);
- символ «C» может стоять перед «D» и «M» для обозначения чисел 400 («CD») и 900 («CM»).

Ограничение на заданное для перевода число – 3999, поскольку далее в римской системе счисления начинаются обозначения, которые довольно сложно выразить в ASCII-кодировке [1].

4.2. Подстрока максимальной длины.

Найти в произвольной строке максимальную длину **подстроки**, в которой нет ни одного повторяющегося символа.

Примеры.

```
Входной параметр: "trttwe6bm6ad"
Ответ:            "twe6bm"

Входной параметр: "wwww"
Ответ:            "w"

Входной параметр: "wdaakva2"
Ответ (один из):  "wda" или "akv"
```

[1] пример тут и тут

4.3.Преобразование строки под кодовой кличкой «зигзаг». Для входной строки и заданного *целого* числа, представляющего условное количество строк, разбить исходную строку и считать новую строку по следующему примеру:

```
Входные параметры: "stopcrazyworld", 4
Ответ:             "saltrzrdocyopw"

Как получилось:
s   a   l
t  rz  rd
o c y o
p   w
```

Заполнили зигзаг в соответствии с заданным количество строк и считали все непустые символы с первой по последную строки.

4.4. Найти самый длинный палиндром во входящей строке

Задана произвольная строка. Найти в ней подстроку **наибольшей длины**, которая является палиндромом.

```
Входной параметр: "wghgtyk"
Ответ:            "ghg"

Входной параметр: "wtcr6uaba"
Ответ:            "wtc" или "aba"

Входной параметр: "kvaawk"
Ответ (один из):  "aa"
```

4.5. Удалить дубликаты из отсортированного массива.

Дан массив целых чисел, отсортированных по возрастанию. Удалить из него все дубликаты и **модифицировать его** так, чтобы первые k-мест занимали уникальные числа.

Ограничение: не использовать дополнительные массивы (т.е., не разрешено никакое дополнительное выделение памяти).

4.6. Поиск значения в отсортированном, но повёрнутом, массиве

Дан отсортированный по возрастанию массив целых чисел. Он может быть **повёрнут** на произвольное число k такми образом, что его элементы расположились следующим образом:

```
[
  numbers[k], numbers[k + 1], ... , numbers[len - 1],
  numbers[0], ... , numbers[k-1]
]
```

Задано число target для поиска в массиве. Найти его позицию (или вернуть «-1», если не найдена) но с условием: прямой перебор по элементам массива запрещён. Другими словами, алгоритм должен работать быстрее (в среднем), чем $\mathrm{O(len)}$, где len – количество элементов массива.

4.7. Озвучка строки чисел.

Пусть задано преобразование:

```
Берём строку, состоящую из цифр:
"8872224"

произносим её словами:
"две восьмёрки одна семёрка три двойки одна четвёрка"

заменяем слова на числа и получаем новую строку из цифр:
"28173214"
```

Реализовать функцию, осуществляющую описанное преобразование.

4.8. Посчитай-и-озвучь.

Некая последовательность цифр **count-and-say** определяется рекурсивной формулой:

- $count\_and\_say(1) \rightarrow$ "1";
- $count\_and\_say(n) \rightarrow$ способ озвучить строку count_and_say(n-1) (пример озвучки – в задании **4.7**).

Пример:

```
count_and_say(1) == "1"

count_and_say(2) => озвучить count_and_say(1)
                 => одна единица == "11"

count_and_say(3) => озвучить count_and_say(2)
                 => две единицы == "21"

count_and_say(4) => озвучить count_and_say(3)
                 => одна двойка одна единица == "1211"
```

Реализовать функцию **count_and_say(n)**.

4.9. Сумма комбинаций.

Дан массив **различных** целых чисел и некоторое целевое целочисленное значение. Найти **все** комбинации элементов массива, сумма которых даёт искомое значение. Любой элемент массива может быть использован в комбинациях *произвольное число раз*. Две комбинации считаются уникальними, если в них различается частота хождения как минимум **одного числа** из исходного массива. Пример:

```
Входные данные: inputs = [2, 3, 4]; target = 8
Ответ: [[2, 2, 2, 2], [2, 2, 4], [4, 4]];
    2 + 2 + 2 + 2 == 8
    2 + 2 + 4     == 8
    4 + 4         == 8
```

Ограничения:

- числа во входном массиве ограничены диапазоном [1; 200];
- целевой параметр ограничен диапазоном [1; 500].

4.10. Вложенный массив максимальной длины.

Дан массив целых чисел. Реализовать функцию, которая находит в нём вложенный массив с максимальной суммой его элементов (по сравнению с другими кандидатами). Вернуть из функции сам влоденный массив и его сумму.

Под *вложенным массивом* понимается любая ненулевая непрерывная комбинация элементов исходного массива. Пример:

```
Входные данные: inputs = [4, -2, -5, 3, -1, 4, -5];
Ответ: {[3, -1, 4], 6};
```

4.11. Получить уникальные интервалы.

Определим *целочисленный интервал* как массив из двух элементов, содержащих его начало и конец. Пусть задан массив таких интервалов. Реализовать функцию, которая пределит **перекрывающиеся** интервалы, объединит их и вернёт массив *неперекрывающихся* интервалов, который включает в себя весь исходный диапазон. Пример:

```
Входные данные: inputs = [[2, 5], [3, 8], [8, 10], [23, 88]];
Ответ: [[2, 10], [23, 88]];
```

4.12. Найти все корректные IPV4 адреса.

IPV4 адрес – некоторый уникальный номер части устройств в сети интернет. Он состоит из **четырёх** целых чисел, разделённых «точкой». Например, «**192.0.1.165**». Каждое число является *десятичным* целым в диапазоне $[0;\ 255]$.

Реализовать функцию, которая по заданной строке построит список всех корректных IP-адресов.

```
Входные данные: "0000"
Ответ: ["0.0.0.0"]

Входные данные: "000"
Ответ: [] (невозможно построить корректный IP адрес)

Входные данные: "25519201"
Ответ: ["255.192.0.1", "255.19.20.1", "255.1.9.201"]
```

4.13. Игра в ликвидатора.

Пусть задано целое число **N** и *условный массив* в диапазоне $[1;\ N]$, в котором числа идут в возрастающем порядке с шагом 1. Преобраховать этот условный массив по следующему алгоритму:

1. начиная с левой границы удалить все **нечётные** (*согласно позициям, не индексам*) элементы;
2. начиная с правой границы удалить все **нечётные** элементы;
3. повторять шаги **1** и **2** до тех пор, пока не останется только один элемент в массиве.

Реализовать функцию, которая по заданному числу **N** вернёт последний «оставшийся-в-живых» элемент массива.

```
Входные данные: N = 6;
Условный массив: [1, 2, 3, 4, 5, 6]
Шаг 1: [_, 2, _, 4, _, 6] => [2, 4, 6]
Шаг 2: [_, 4, _] => [4]
Ответ: 4
```

4.14. Медиана двух массивов.

Даны два массива целых чисел, упорядоченных *по возрастанию*. Найти **общую медиану** этих двух массивов.

Ограничения: не использовать дополнительный массив.

Примеры:

```
Входные данные: nums1 = [1,5], nums2 = [3];
Ответ: 3.0
Пояснения: условный объеденённый массив будет => [1, 3, 5],
           его медиана равна 3.0 (приведена к действительному типу)


Входные данные: nums1 = [-4, -8, 4], nums2 = [1, 5, 7];
Ответ: 2.5
Пояснения: объеденённый массив => [-4, -8, 1, 4, 5, 7],
           его медиана равна (1 + 4) / 2 = 2.5 (чётное число элементов)
```

4.15. Полные квадраты.

Для заданного числа N найти **наименьшее** число *полных квадратов*, сумма которых равна N.

Под *полным квадратом* понимается число, квадратный корень из которого также является целым числом. Примеры:

```
Входные данные: N = 15;
Ответ: 4
15 = 9 + 4 + 1 + 1

Входные данные: N = 12;
Ответ: 4
12 = 4 + 4 + 4 + 4

Входные данные: N = 17;
Ответ: 2
17 = 16 + 1
```

***Дополнительное развлечение** для интереса, можно само разложение выводить на консоль или получать в текстовом виде из функции.

4.16. Выбор случайного индекса.

Дан массив целых неуникальных чисел и произвольное целое число **target**.

Реализовать функцию, которая вернёт индекс числа **target**. Если во входном массиве число элементов, равных **target**, больше единицы, то все подходящие индексы должны возвращаться функцией **равновероятно**. Пример:

```
Входные данные: nums = [2, 3, 4, 1, 5, 6, 3, 8, 2, 3];
          target = 4
Ответ: 2 (индекс четвёрки)

          target = 3
Ответ: 1, 6 или 9 - один из индексов возвращается случайным образом.
       Вероятность возвращения каждого должна быть одинакова

          target = 2
Ответ: 0 или 8
```

***Дополнительное развлечение** вместо функции реализуйте *функциональный объект*.

4.17. Деление по заказу.

Пусть заданы массив ehfdytybq **equations**, состоящий **из пары** символьных значений, и **values** – состоящий из набора действительных чисел. Размерности обоих массивов должны совпадать. Каждая пара $equations[i] == [A_i, B_i]$ и число $values[i]$ представляют собой уравнение $\frac{A_i}{B_i} = values[i]$.

Также задан массив запросов **queries**, которые, аналогично **equations**, предствляет собой некоторые символьные пары $queries[i] == [Y_i, Z_i]$. Для каждого запроса найти решение уравнения $\frac{Y_i}{Z_i} = ?$. В случаях, если данных недостаточно в качестве ответа вернуть **nan** (**std::nan** из **<cmath>** в C++). Пример:

```
Входные данные:
  equations = [["a", "b"], ["b", "c"]],
  values = [2.0, 3.0],
  queries = [["a", "c"], ["b", "a"], ["a", "bc"],["a", "a"], ["x", "x"]]

Ответ: [6.0, 0.5, nan, 1.0, nan]

Пояснения: первый запрос - a/c. При этом, имеем: a/b == 2.0, b/c == 3.0
Дальше действовать можно по разному, например перемножением a/b * b/c
получаем a/c == 6.0.

"bc", "x" - не присутствуют как составляющие в исходном массиве уравнений,
так что - сразу Not-A-Number, то есть - nan.

a/a - одинаковые символы, принимаем как 1.0
```

4.18. Поиск подмножества.

Задан массив *двоичных чисел*, представленных в виде **строк** (каждый элемент массива состоит только из символов «0» и «1»). Заданы два числа:

- **max_zeros_count** – максимальное количество *нулей*;
- **max_ones_count** – максимальное чколичество *единиц*;

  Найти размер **наибольшего** подмножества из элементов исходного массива, в котором число *нулей* и *единиц* **не превышает** заданных максимальных значений.

```
Входные данные: strs = ["10","0001","111001","1","0"],
            max_zeros_count = 5, max_ones_count = 3
```

Ответ: 4 (элемента)

Пояснения: наибольшее подмножество есть - ["10", "0001", "1", "0"]. В нём число нужлей равно 5, число единиц - 3, что не превышает заданные границы.Примера неподходящего подмножества в данном случае - ["0001","111001"], где число единиц равно 5, что больше заданного порога в максимум 3 штуки. Другие кандидаты - ["0001", "1"] и ["10", "1", "0"], но их размер равен только 2. Поэтому, ответ - 4