

Чего хотят на алгоритмическом собеседовании?

1. Умение писать работающий код
2. Умение тестировать свой код
3. Умение писать эффективный код

Как этого достичь?

1. Написать 10000 строк работающего, эффективного и протестированного кода
2. Найти задачи для тренировок можно на leetcode.com, codeforces.com и т.д.

Тема 1: Сложность, тестирование, особые случаи

Сложность алгоритмов

Сложность алгоритма – порядок количества действий, которые алгоритм выполняет.

Фраза «сложность алгоритма есть $O(f(n))$ » означает, что с увеличением параметра n , характеризующего количество входной информации алгоритма, время работы алгоритма будет возрастать не быстрее, чем умноженная на некоторую константу $f(n)$. (Асимптотическая сложность)

Например, в программе два вложенных цикла, каждый от 1 до N , тогда сложность алгоритма $O(N^2)$.

$100 * N = O(N)$, $2 * N = O(N)$. Здесь 100 и 2 константы, не зависящие от размера входных данных. Константы не так сильно влияют на скорость работы алгоритма при больших параметрах.

Существует «пространственная сложность» - количество использованной памяти. Асимптотическая сложность всегда больше пространственной, так как процесс сохранения данных тоже занимает время.

Задача 1. Дана строка (в кодировке UTF-8). Найти самый часто встречающийся в ней символ. Если несколько символов встречаются одинаково часто, то можно вывести любой.

Решение 1. Переберём все позиции и для каждой позиции в строке ещё раз переберём все позиции и в случае совпадения прибавим к счётчику единицу. Найдём максимальное значение счетчика. Сложность алгоритма $O(N^2)$, так как тут 2 вложенных цикла.

```
s = input()
ans = ''
anscnt = 0
for i in range(len(s)):
    nowcnt = 0
    for j in range(len(s)):
        if s[i] == s[j]:
            nowcnt += 1
    if nowcnt > ans cnt:
        ans = s[i]
        ans cnt = nowcnt
print(ans)
```

Во всех алгоритмических задачах консольный ввод и вывод.

Решение 2. Переберём все символы встречающиеся в строке, а затем переберём все позиции и в случае совпадения увеличим счетчик на 1. Найдём максимальное значение счетчика. Сложность алгоритма $O(N * k)$, где k - количество различных символов в строке.

```

s = input()
ans = ''
anscnt = 0
for now in set(s):
    nowcnt = 0
    for j in range(len(s)):
        if now == s[j]:
            nowcnt += 1
    if nowcnt > anscnt:
        ans = now
        anscnt = nowcnt
print(ans)

```

Немного улучшили решение, так как $k \leq N$.

Решение 3. Заведём словарь, где ключом является символ, а значением — сколько раз он встретился. Если символ встретился впервые — создаём элемент словаря с ключом, совпадающим с этим символом и значением ноль. Прибавляем к элементу словаря с ключом, совпадающим с этим символом, единицу.

```

s = input()
ans = ''
anscnt = 0
dct = {}
for now in s:
    if now not in dct:
        dct[now] = 0
    dct[now] += 1
for key in dct:
    if dct[key] > anscnt:
        anscnt = dct[key]
        ans = key
print(anscnt, ans)

```

Сложность алгоритма $O(N+k) = O(N)$, где k - количество различных символов в строке.

Сравнение сложности

Обозначим за N длину строки, а за K — количество различных символов в ней

| Решение | Время | Память |
|---|----------|-------------------|
| #1 (проход по строке для каждого символа в строке) | $O(N^2)$ | $O(N)$ |
| #2 (проход по строке для каждого символа в множестве) | $O(NK)$ | $O(N + K) = O(N)$ |
| #3 (использование словаря) | $O(N)$ | $O(K)$ |

Особые случаи

Рассмотрим решение 3 с некоторой модификацией (убрали строку `ans = ""`):

```
s = input()
anscnt = 0
symcnt = {}
for now in s:
    if now not in symcnt:
        symcnt[now] = 0
    symcnt[now] += 1
    if symcnt[now] > anscnt:
        ans = now
        anscnt = symcnt[now]
print(ans)
```

Теперь при вводе пустой строки будет ошибка, так как мы будем использовать неинициализированную переменную (`print(ans)`).

Пример. Сумма последовательности

```
seq = list(map(int, input().split()))
if len(seq) == 0:
    print(0)
else:
    seqsum = seq[0]
    for i in range(1, len(seq)):
        seqsum += seq[i]
    print(seqsum)

seq = list(map(int, input().split()))
seqsum = 0
for i in range(len(seq)):
    seqsum += seq[i]
print(seqsum)
```

Тут мы прописали случай, когда последовательность пуста, однако этого можно было не делать (и это рекомендуется не делать), так как усложняется код и ухудшается его читабельность.

Пример. Максимум последовательности

```
seq = list(map(int, input().split()))
seqmax = 0
for i in range(len(seq)):
    if seq[i] > seqmax:
        seqmax = seq[i]
print(seqmax)

seq = list(map(int, input().split()))
if len(seq) == 0:
    print('-inf')
else:
    seqmax = seq[0]
    for i in range(1, len(seq)):
        if seq[i] > seqmax:
            seqmax = seq[i]
    print(seqmax)
```

В данном случае необходимо обработать случай, когда все элементы последовательности < 0 . Иначе получим неправильный ответ (0).

Тестирование

Что нужно протестировать?

1. Тесты из условий (если есть)
2. Общие случаи
3. Особые случаи

Тесты для примера максимум последовательности:

1. 1 3 2 – общий случай
2. **1 2 3** – краевой случай
3. **3 2 1** – краевой случай
4. 1 1 1
5. **1** – один элемент
6. " – пустая последовательность
7. -2 -1 -3

Советы по составлению тестов

- › Если есть примеры — реши их руками и сверь ответ. Если не совпадает, то либо правильных ответов может быть несколько, либо ты неправильно понял задачу
- › Сначала составь несколько примеров и реши задачу руками, чтобы лучше понять условие и чтобы потом было с чем сравнить
- › Проверь последовательность из одного элемента и пустую последовательность
- › «Краевые эффекты» — проверь, что программа работает корректно в начале и конце последовательности, сделай тесты, чтобы ответ находился на первом и на последнем месте в последовательности
- › Составь покрытие всех ветвлений, так чтобы был тест, который входит в каждый if и else
- › Подбери тесты чтобы не было ни одного входа в цикл
- › Один тест — одна возможная ошибка

Покрытие тестами

Задача: даны три целых числа a , b , c . Найти все корни уравнения $ax^2 + bx + c = 0$ и вывести их в порядке возрастания.

Решение 1:

```
d = b ** 2 - 4 * a * c
x1 = -b - sqrt(d) / 2 * a
x2 = -b + sqrt(d) / 2 * a
print(x1, x2)
```

```
› Ввод: 1 -2 0
› Вывод: 1.0 3.0
› Ответ: 0.0 2.0
```

Забыли скобочки в выражениях

Решение 2:

```
d = b ** 2 - 4 * a * c
x1 = (-b - sqrt(d)) / (2 * a)
x2 = (-b + sqrt(d)) / (2 * a)
print(x1, x2)
```

```
> Ввод: 1 2 1
> Вывод: -1.0 -1.0
> Ответ: -1.0
```

Забыли случай, когда
кратность корня = 2.

Решение 3:

```
d = b ** 2 - 4 * a * c
if d == 0:
    x1 = -b / (2 * a)
    print(x1)
else:
    x1 = (-b - sqrt(d)) / (2 * a)
    x2 = (-b + sqrt(d)) / (2 * a)
    print(x1, x2)
```

```
> Ввод: 1 1 1
> Вывод: Runtime Error
> Ответ:
```

Забыли случай
когда $D < 0$.

Решение 4:

```
d = b ** 2 - 4 * a * c
if d == 0:
    x1 = -b / (2 * a)
    print(x1)
elif d > 0:
    x1 = (-b - sqrt(d)) / (2 * a)
    x2 = (-b + sqrt(d)) / (2 * a)
    print(x1, x2)
```

```
> Ввод: 0 1 1
> Вывод: Runtime Error
> Ответ: -1
```

Забыли случай,
когда $a = 0$.

Решение 5:

```
if a == 0:
    print(-c / b)
else:
    d = b ** 2 - 4 * a * c
    if d == 0:
        x1 = -b / (2 * a)
        print(x1)
    elif d > 0:
        x1 = (-b - sqrt(d)) / (2 * a)
        x2 = (-b + sqrt(d)) / (2 * a)
        print(x1, x2)
```

```
> Ввод: 0 0 1
> Вывод: Runtime Error
> Ответ:
```

Забыли случай,
когда $a = b = 0$.

Решение 6:

```
if a == 0:
    if b != 0:
        print(-c / b)
    else:
        d = b ** 2 - 4 * a * c
        if d == 0:
            x1 = -b / (2 * a)
            print(x1)
        elif d > 0:
            x1 = (-b - sqrt(d)) / (2 * a)
            x2 = (-b + sqrt(d)) / (2 * a)
            print(x1, x2)
```

> Ввод: 0 0 0
> Вывод:
> Ответ: Infinite number of solutions

Забыли случай, когда
 $a = b = c = 0$.

Решение 7:

```
if a == 0:
    if b != 0:
        print(-c / b)
    if b == 0 and c == 0:
        print('Infinite number of solutions')
    else:
        d = b ** 2 - 4 * a * c
        if d == 0:
            x1 = -b / (2 * a)
            print(x1)
        elif d > 0:
            x1 = (-b - sqrt(d)) / (2 * a)
            x2 = (-b + sqrt(d)) / (2 * a)
            print(x1, x2)
```

> Ввод: -5 4 1
> Вывод: 1.0 -0.2
> Ответ: -0.2 1.0

Забыли вывести в
возрастающем порядке.
Надо было проверить с
отрицательными корнями.

Решение 8:

```
if a == 0:
    if b != 0:
        print(-c / b)
    if b == 0 and c == 0:
        print('Infinite number of solutions')
    else:
        d = b ** 2 - 4 * a * c
        print(sqrt(d))
        if d == 0:
            x1 = -b / (2 * a)
            print(x1)
        elif d > 0:
            x1 = (-b - sqrt(d)) / (2 * a)
            x2 = (-b + sqrt(d)) / (2 * a)
            if x1 < x2:
                print(x1, x2)
            else:
                print(x2, x1)
```

Ура!

Задача. Поиск самого частого символа (была выше)

Заведем словарь, где ключом является символ, а значением — сколько раз он встретился. Если символ встретился впервые — создаем элемент словаря с ключом, совпадающим с этим символом и значением ноль. Прибавляем к элементу словаря с ключом, совпадающим с этим символом, единицу

```
s = input()
ans = ''
anscnt = 0
symcnt = {}
for now in s:
    if now not in symcnt:
        symcnt[now] = 0
    symcnt[now] += 1
    if symcnt[now] > anscnt:
        ans = now
print(ans)
```

Забыли поменять anscnt при проверке количества вхождений с anscnt (if symcnt[now] > anscnt).