8-11 классы

Программирование
на Python

Презентация занятия

Алгоритмы и их сложность.

46 занятие

инжиниум®
МГТУ им. Н.Э. Баумана

2023

Python: Алгоритмы и их сложность.

Оценка сложности алгоритма – неотъемлемая часть в разработке ПО. Зачастую, алгоритмы постоянно перерабатываются, для повышения эффективности и уменьшения нагрузки на машину.

Python: Алгоритмы и их сложность.

размер / сложность	10	20	30	40	50	60
n	0,00001 сек.	0,00002 сек.	0,00003 сек.	0,00004 сек.	0,00005 сек.	0,00005 сек.
n^2	0,0001 сек.	0,0004 сек.	0,0009 сек.	0,0016 сек.	0,0025 сек.	0,0036 сек.
n^3	0,001 сек.	0,008 сек.	0,027 сек.	0,064 сек.	0,125 сек.	0,216 сек.
n^5	0,1 сек.	3,2 сек.	24,3 сек.	1,7 минут	5,2 минут	13 минут
2^n	0,0001 сек.	1 сек.	17,9 минут	12,7 дней	35,7 веков	366 веков
3^n	0,059 сек.	58 минут	6,5 лет	3855 веков	$2x10^8$ веков	$1,3x10^{13}$ веков

Python: Алгоритмы и их сложность.

Задача. С списке находится произвольное количество чисел, требуется найти хотя бы одну пару таких чисел, которое дает в сумме заданное число.

Пример. Сумма – 12

```
a=[3,7,23,-1,-10,5,24] -> (7, 5)
```

Python: Алгоритмы и их сложность.

```
def task_1(a:list, sum:int)->tuple:
    for i in range(len(a)):
        for j in range(i+1, len(a)):
            if a[i]+a[j] == sum:
                return a[i], a[j]
```

Сложность данного алгоритма в худшем случае равна $O(n^2)$.
При размере списка 10^3 наша функция может найти пару чисел после выполнения 10^6 операций.

Python: Алгоритмы и их сложность.

Представим что наши числа в списке отсортированы. Для поиска пары для текущего числа можно использовать алгоритм бинарного поиска.

В таком случае, сложность алгоритма равна $O(nlog_2 n)$

```
def task_2(a:list, sum:int)->tuple:
    for i in a:
        low, high = 0, len(a)-1
        while(low < high):
            mid = (low+high)//2
            if a[mid] == sum - i:
                return a[mid], sum - a[mid]
            elif a[mid] < sum - i:
                low = mid+1
            elif a[mid] > sum - i:
                high = mid-1
```

Python: Алгоритмы и их сложность.

Попробуем улучшить наш алгоритм еще больше. Для этого воспользуемся свойством множеств в программировании:

Множества:

- Не итерированы
- Не содержат повторяющихся элементов
- Имеют константный look up(проверка на наличие конкретного элемента)

Python: Алгоритмы и их сложность.

Построим алгоритм для поиска пары следующим образом: Пусть sum – требуемая сумма, i – текущий элемент списка. Для каждого i можно проверить его на наличии в множестве, в случае если его нет, добавим в множество элемент $sum - i$. В случае если элемент существует, тогда мы нашли пару чисел.

Python: Алгоритмы и их сложность.

Пример. Для `a=[-10, -1, 3, 5, 7, 23, 24]`найдем пару чисел для суммы 12. Создадим множество stack, в котором будем хранить его «потенциальные пары».Место для уравнения.

Текущий элемент	Множество stack
-10	{22}
-1	{22, 13}
3	{22, 13, 9}
5	{22, 13, 9, 7}
7	{22, 13, 9, 7}

Python: Алгоритмы и их сложность.

```
def task_3(a:list, sum:int)->tuple:
    stack = set()
    for i in a:
        if i not in stack:
            stack.add(sum-i)
        else:
            return i, sum-i
```

Сложность данного алгоритма в худшем случае равна $O(Cn)$. Несмотря на неизвестное значение в C, наш алгоритм всегда будет быстрее сложности $O(n^2)$.

Python: Алгоритмы и их сложность.

Задача для самостоятельного решения. Петя и Вася играют в теннис. Правила игры таковы: за каждый забитый мяч, игрок получает 15 очков. Если счет двух противников равен 60-60, очки обнуляются счет становится 0-0. Помогите понять кто выиграл. На вход подается строка вида 'aabaa' где каждый символ говорит об игроке забившем мяч в текущем раунде(игрок a или b). Попробуйте оптимизировать ваш алгоритм.

'aabaaa' - выиграл игрок a
'aaaabbbbaabaaa' - выиграл игрок a