Лекция по АиСД

Амортизационный анализ. Простые структуры данных.

Зачем нужны разные структуры данных?

Разная эффективность операций
1. Добавление/удаление
2. Поиск
3. Random access (доступ по индексу)
4. Поиск максимума/минимума

и т. д.

https://wiki.python.org/moin/TimeComplexity

Динамический массив (list)

1. Добавление/удаление в конец – O(1)
2. Добавление/удаление в произвольное место – O(N)
3. Random access (доступ по индексу) – O(1)
4. Поиск элемента – O(N)

Динамический массив (list)

```
a = [1, 2, 3]
# Доступ по индексу
a[0] = 3 # O(1)

# Добавление/удаление в конец
a.append(4) # O(1)
a.pop() # O(1)

# Добавление/удаление в произвольное место
a.insert(1, 5)
a.pop(1)

a.remove(2) # Удаление по значению включает в себя поиск! O

# Поиск
a.index(3) # O(N)
```

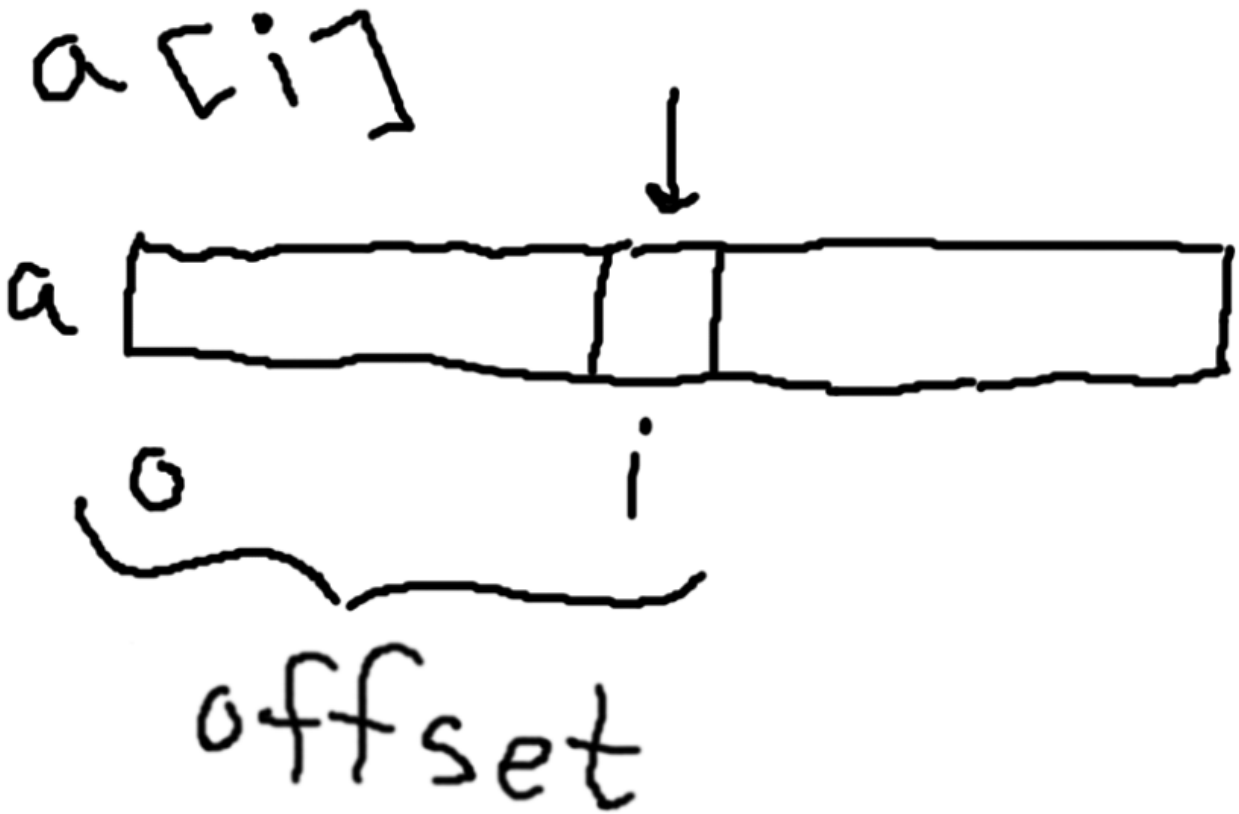
Как могло бы быть устроено удаление по значению?

```
a = [1, 2, 3]
def remove(value):
    # Поиск
    index = -1
    for i in range(len(a)):
        if a[i] == value:
            index = i
            break
    if index == -1:
        raise ValueError("...")

    # Копирование "хвоста"
    for i in range(index, len(a) - 1):
        a[i] = a[i + 1]
    a.pop()

remove(2)
print(a) # [1, 3]
```

Как устроен динамический массив?

Динамический массив – непрерывный участок памяти
В переменной на самом деле храним адресс начала массива
Адресс i-го элемента можно посчитать

Доступ по индексу



Адресс a[i] = Адресс a[0] + offset

Копирование

```
a = [1, 2, 3]
b = a # O(1)
b[1] = 4
print(a) # prints [1, 4, 3]
```

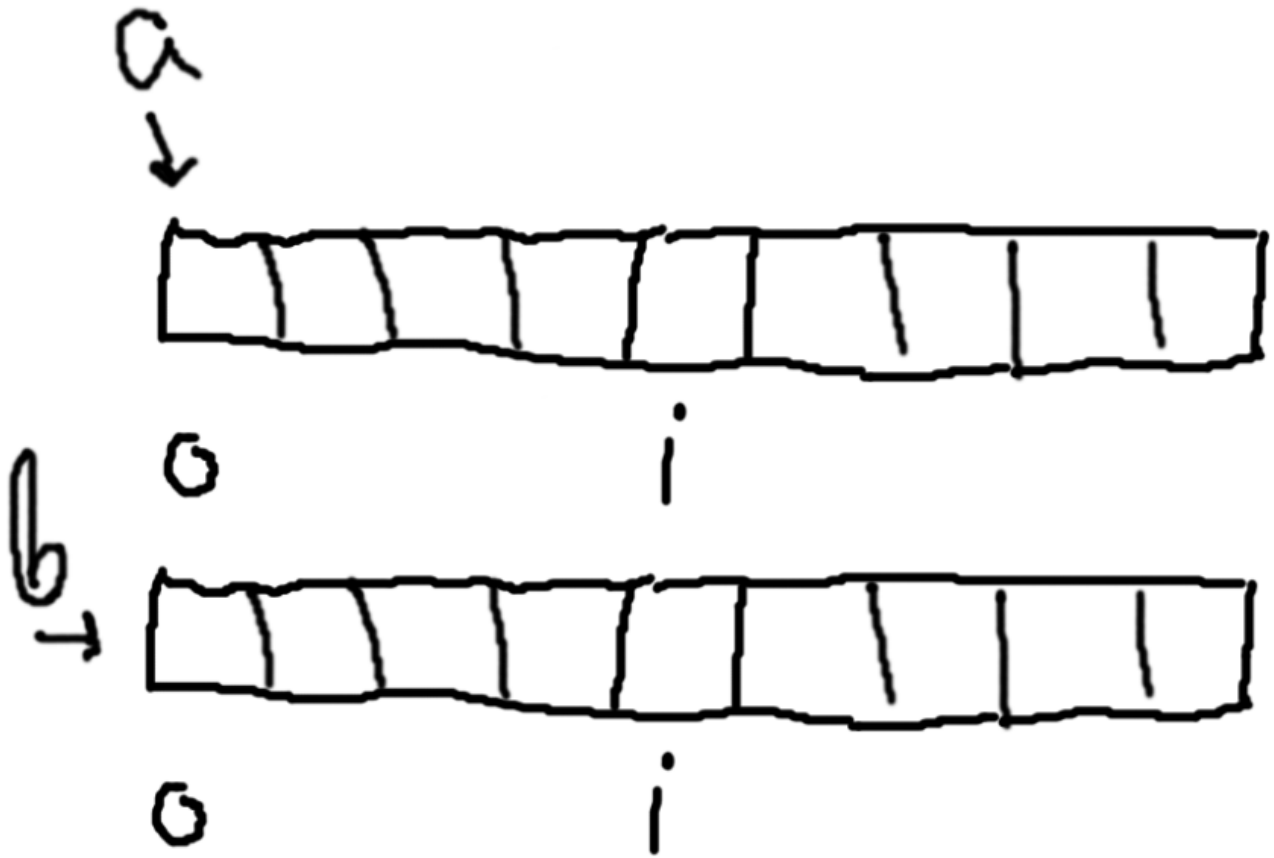
Копирование

a и b указывают на один и тот же участок памяти!

Копирование

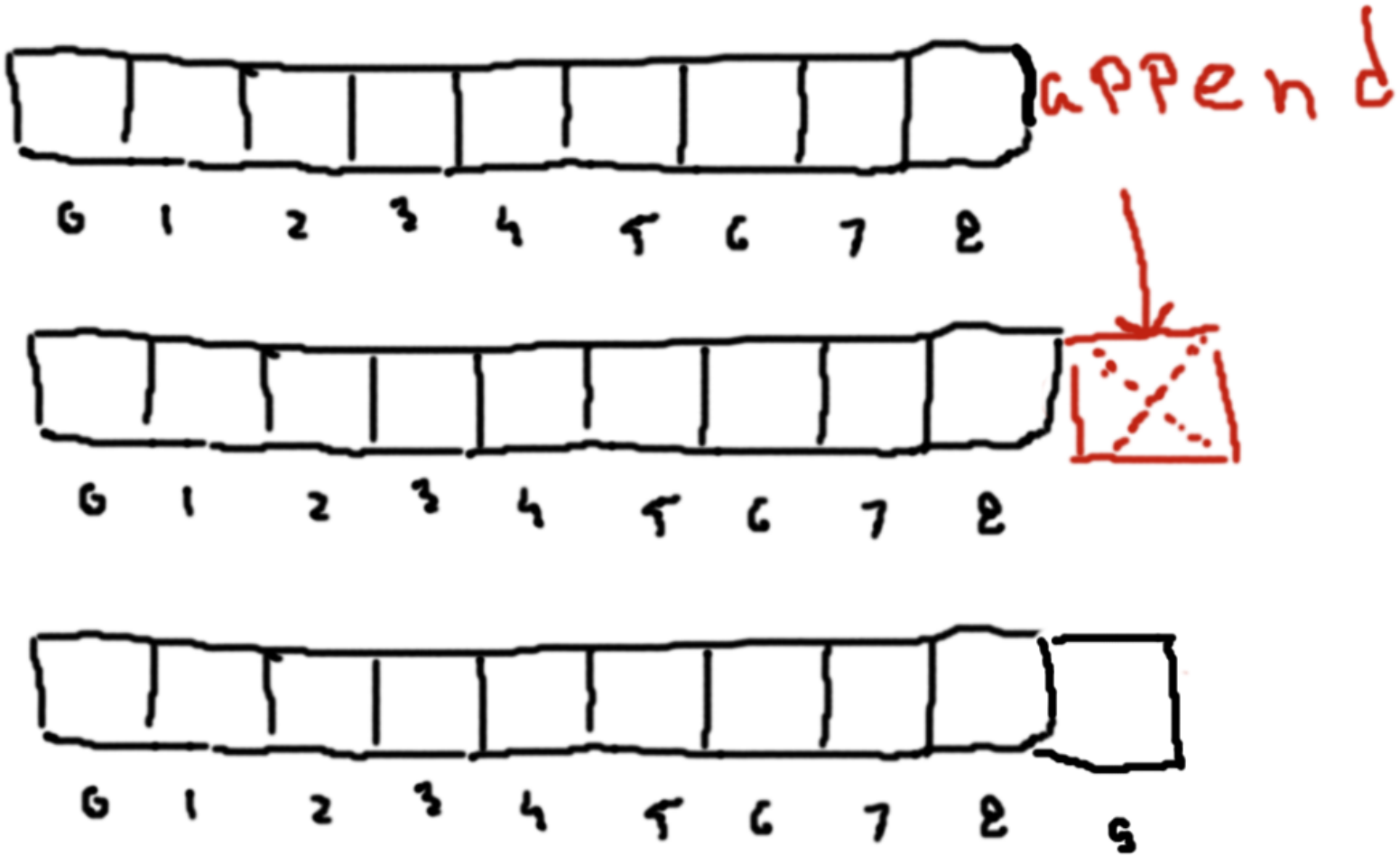
```
a = [1, 2, 3]
b = a.copy() # или b = a[:], O(N)
b[1] = 4
print(a) # prints [1, 2, 3]
```

Копирование



Теперь это копия

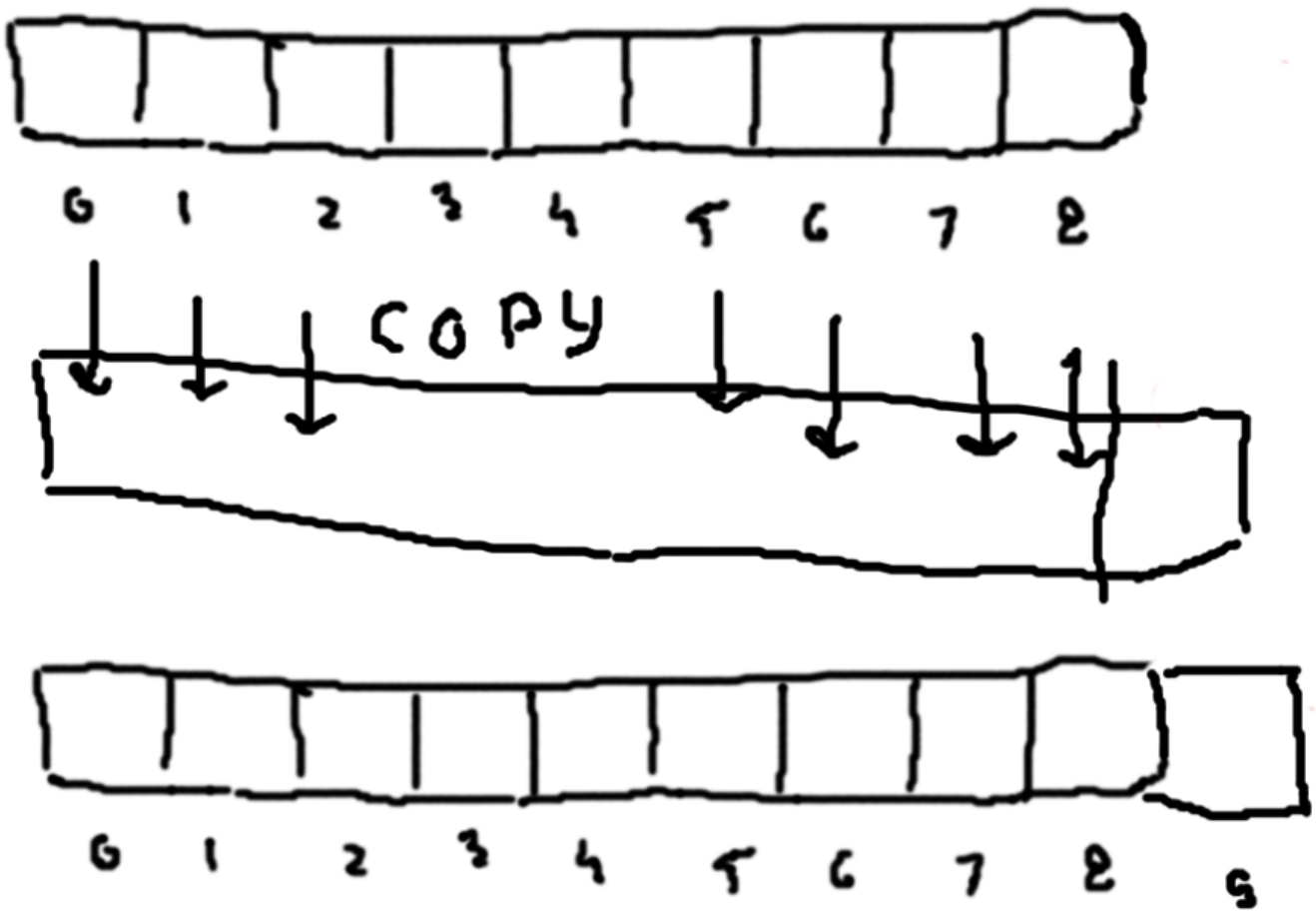
Как устроен динамический массив?



Выделение памяти не бесплатное!

Как устроен динамический массив?

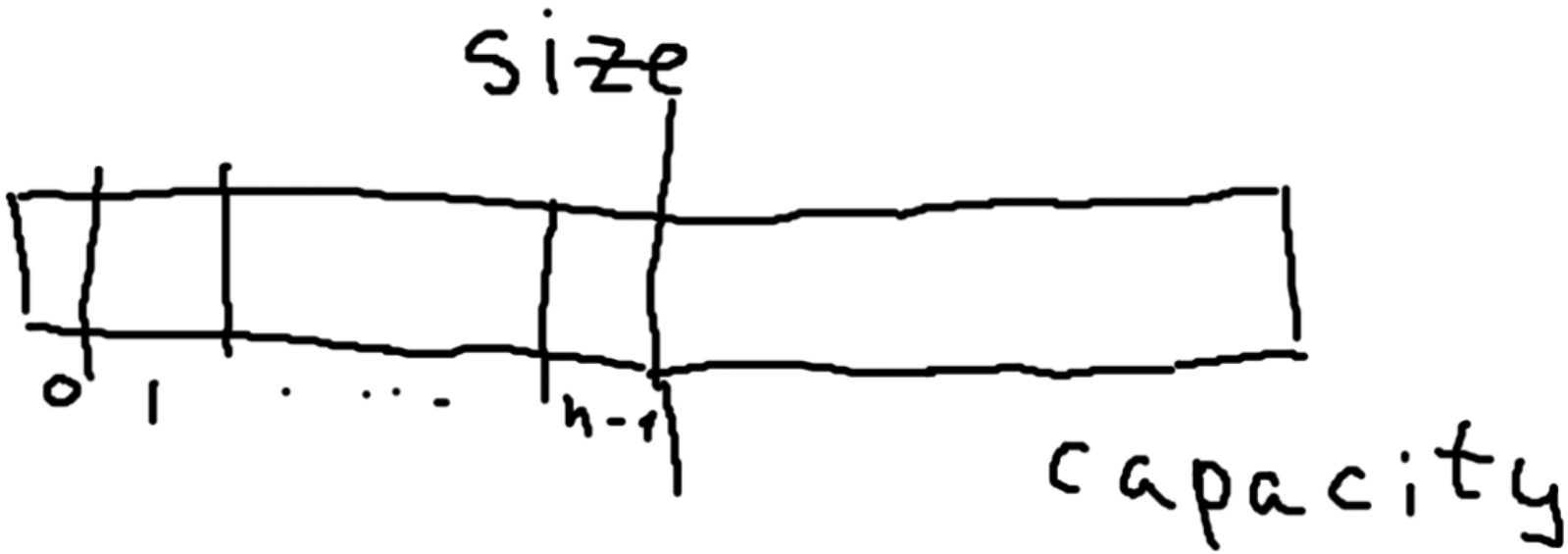
Если каждый раз, когда нам нужно вставить элемент, перевыделять память, мы получим сложность вставки O(N)



Size и capacity

Size – реальный размер массива, количество элементов в нем
Capacity – зарезервированная под массив память
Size <= Capacity

Size и capacity



Амортизационный анализ

Анализ в худшеем случае - сколько времени нужно на одну операцию?

Амортизационный анализ – сколько времени нужно на последовательность из N операций?

Амортизационный анализ

Если на последовательность из N операций нам нужно O(N) времени, то в среднем операция выполняется за O(1)

Доступ по индексу – O(1) в худшем случае

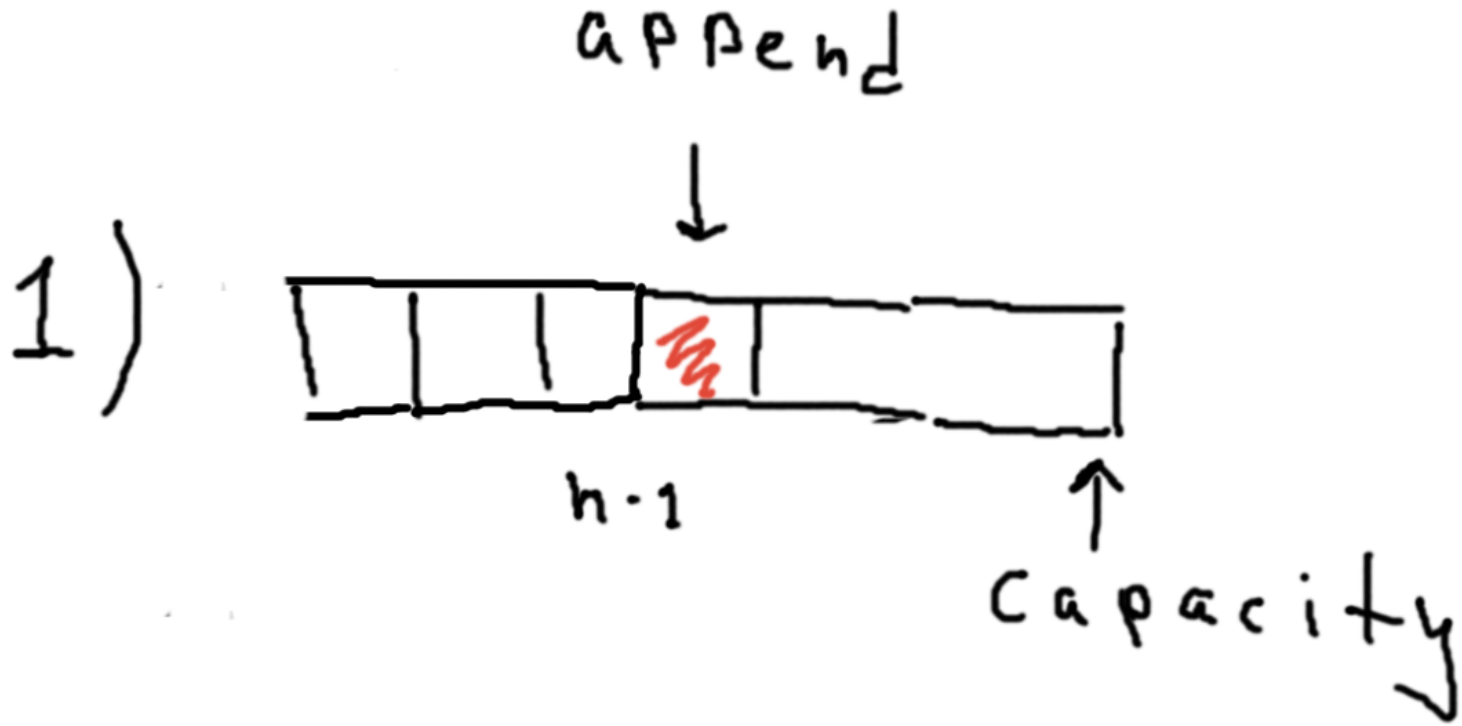
Вставка в конец – O(1) в среднем (амортизированная или учетная стоимость)

Вставка в конец

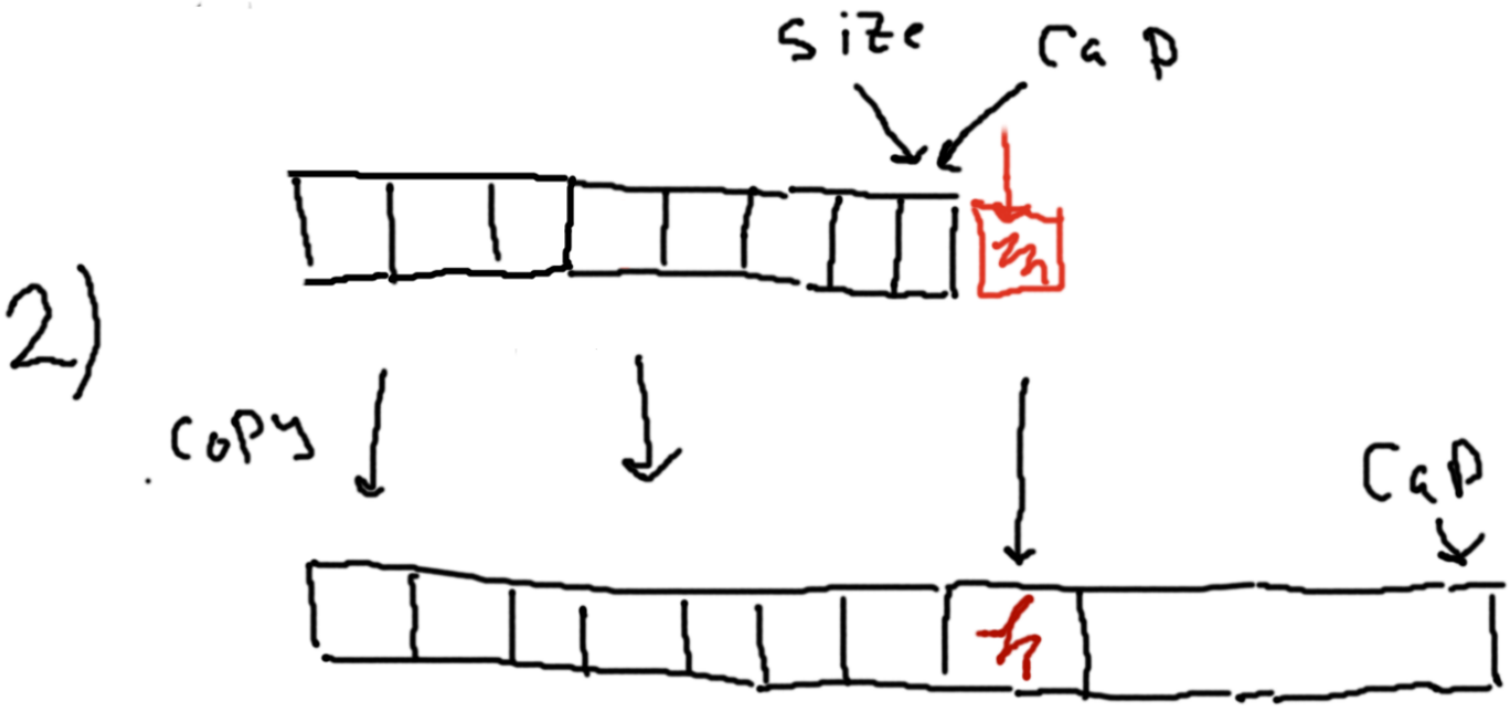
Если size $<$ capacity, просто добавляем элемент (память у нас еще есть)

Если size $==$ capacity, происходит перевыделение памяти

Вставка в конец (простой случай)



Вставка в конец (сложный случай)



Банковский метод

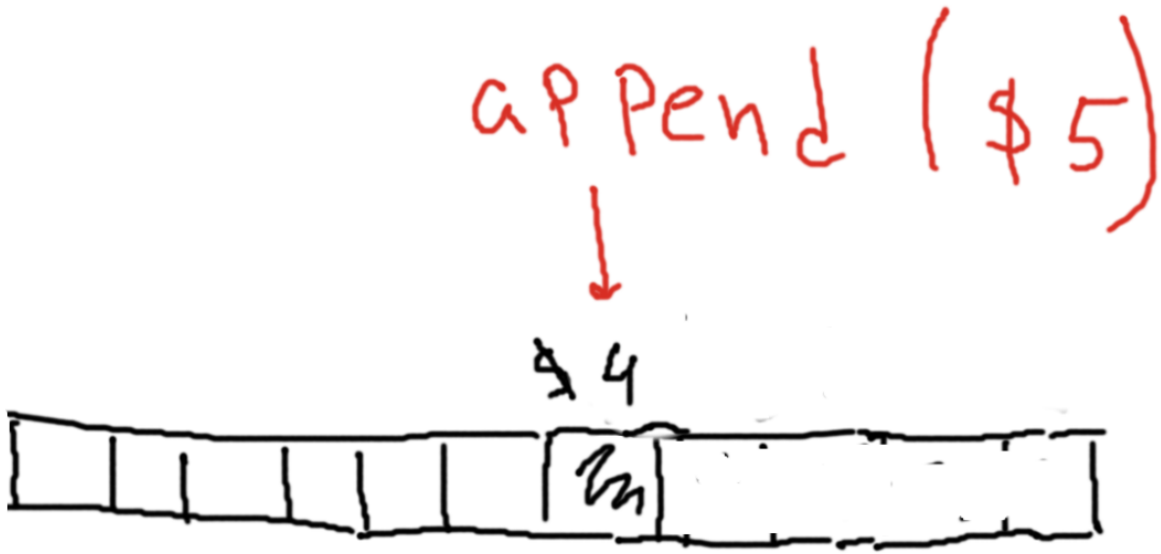
c_i — — стоимость i-й операции
g_i — — учетная стоимость операции (сколько мы за нее платим)

Если $g_i > c_i$, кладем разницу в "банк"
Если $g_i < c_i$, забираем разницу из "банка"
Нельзя уходить в минус

Вставка в конец: банковский метод

За каждый append будем просить \$5
В простом случае (capacity $>$ size) \$1 потратим на саму вставку,
а еще \$4 положим в банк (оставим их там до сложного случая)

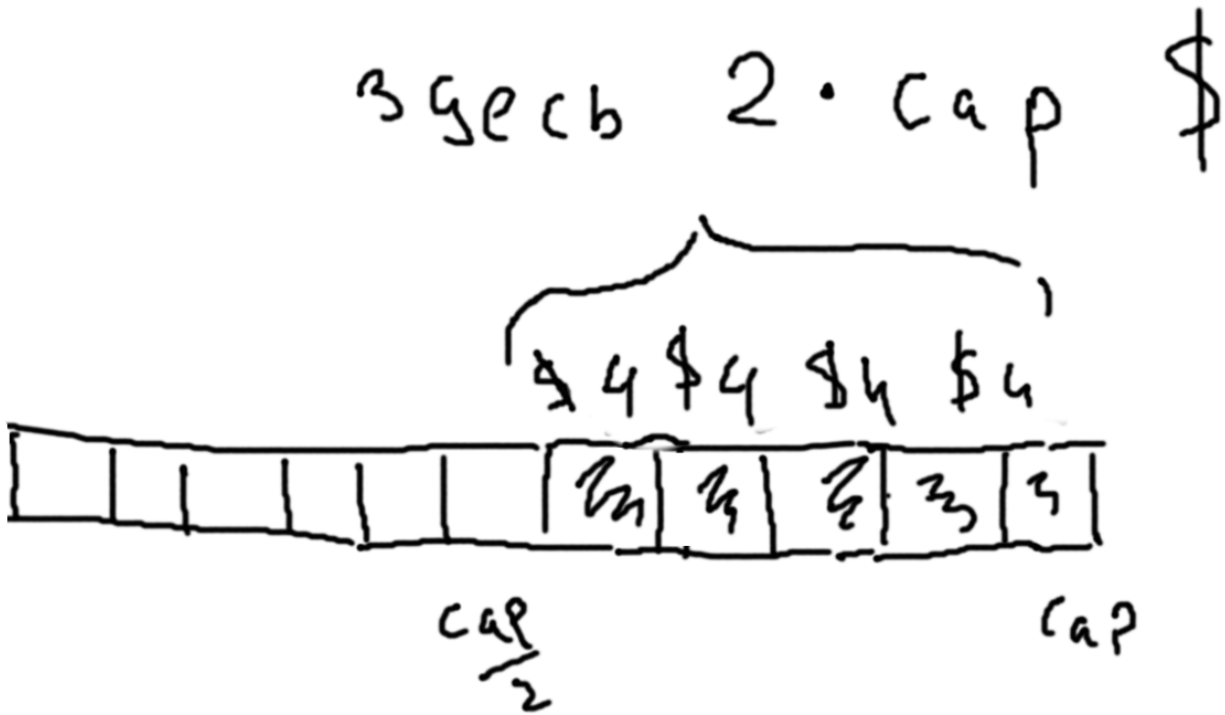
Вставка в конец: банковский метод



Вставка в конец: банковский метод

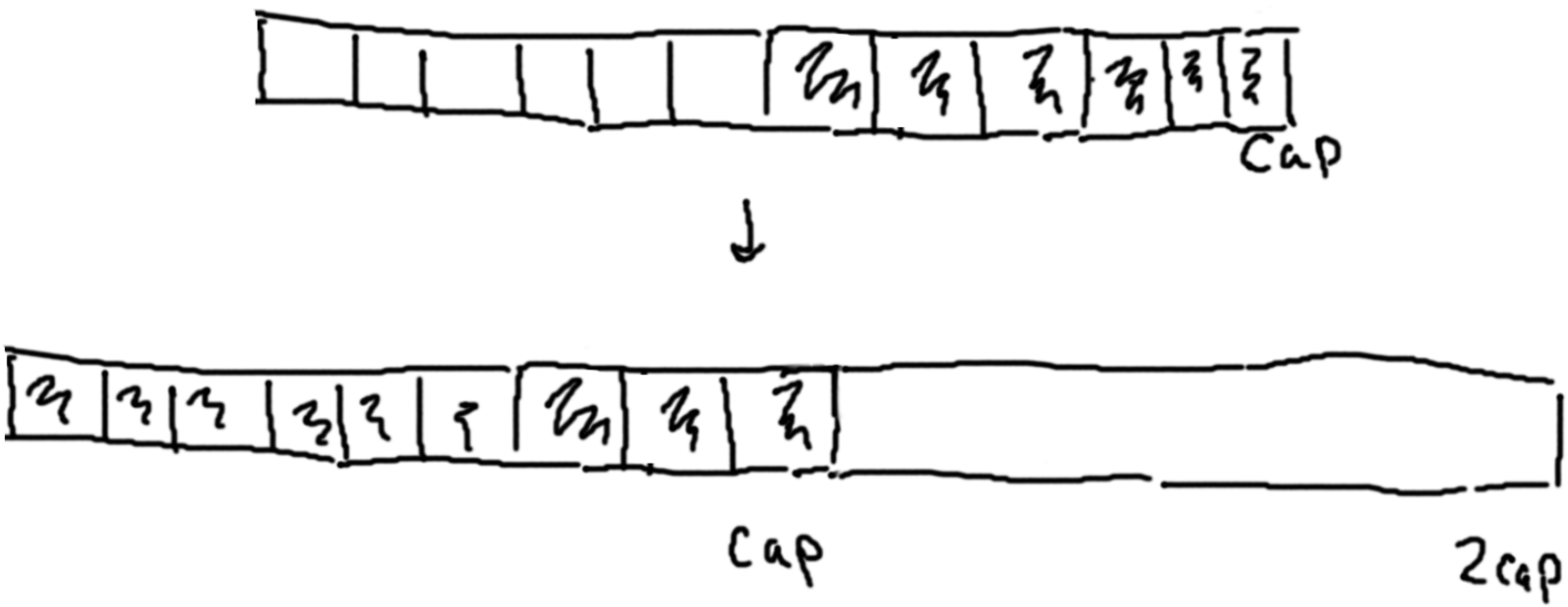
Каждый раз будем увеличивать capacity вдвое
На копирование элементов тратим деньги из банка

Вставка в конец: банковский метод



В банке \$2 * cap, как раз хватит на перевыделение памяти

Вставка в конец: банковский метод



Стек

1. Добавление/удаление в конец – O(1)

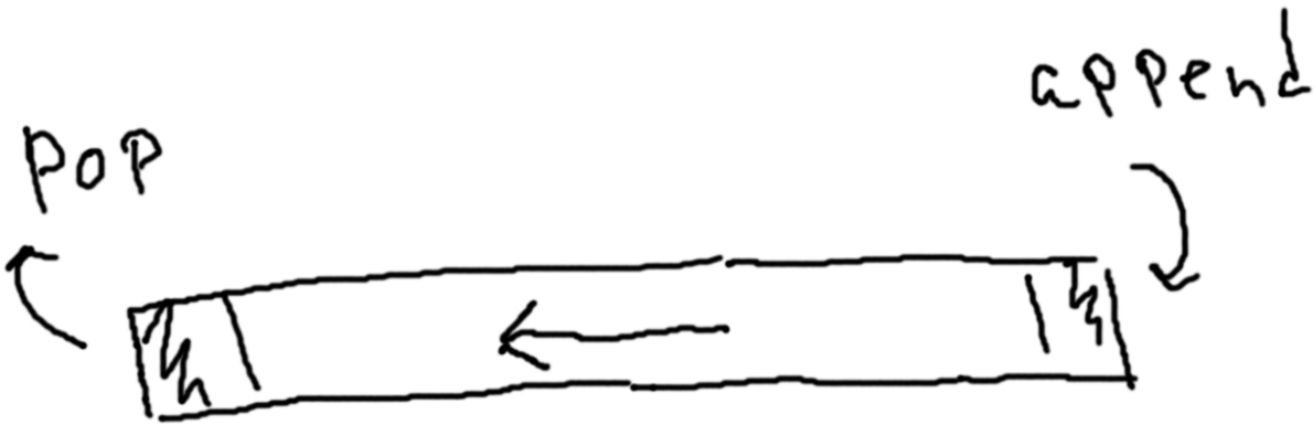
Если вам нужен стек, можно пользоваться list-ом

Очередь (queue)

1. Добавление в конец – O(1)
2. Удаления из начала – O(1)

В питоне есть https://docs.python.org/3/library/queue.html, но лучше пользоваться deque из collections

queue



Двусторонняя очередь (дек, deque)

1. Добавление/удаление в конец – O(1)
2. Добавление/удаление в начало – O(1)
3. Random access (доступ по индексу) – O(1)

from collections import deque

deque

```
from collections import deque
a = deque([1, 2, 3])

# Все операции ниже работают за O(1)

a.append(4) # [1, 2, 3, 4]
a.appendleft(5) # [5, 1, 2, 3, 4]

a.pop() # [5, 1, 2, 3]
a.popleft() # [1, 2, 3]

a[2] = 4
```