

академия
больших
данных

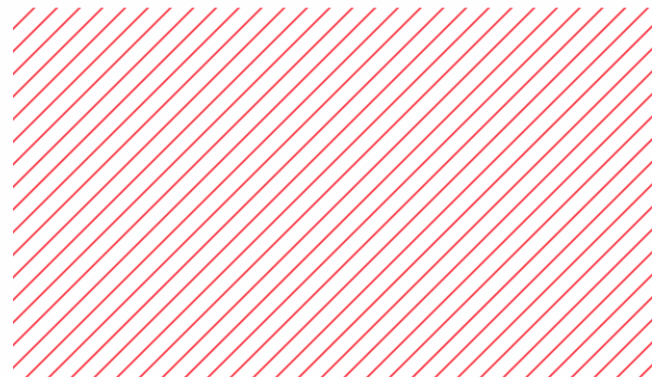
mail.ru
group



Базовые структуры данных

Шовкоплас Григорий

Введение в алгоритмы и структуры данных





DATA STRUCTURE

**DATA STRUCTURE
EVERYWHERE**

makeameme.org

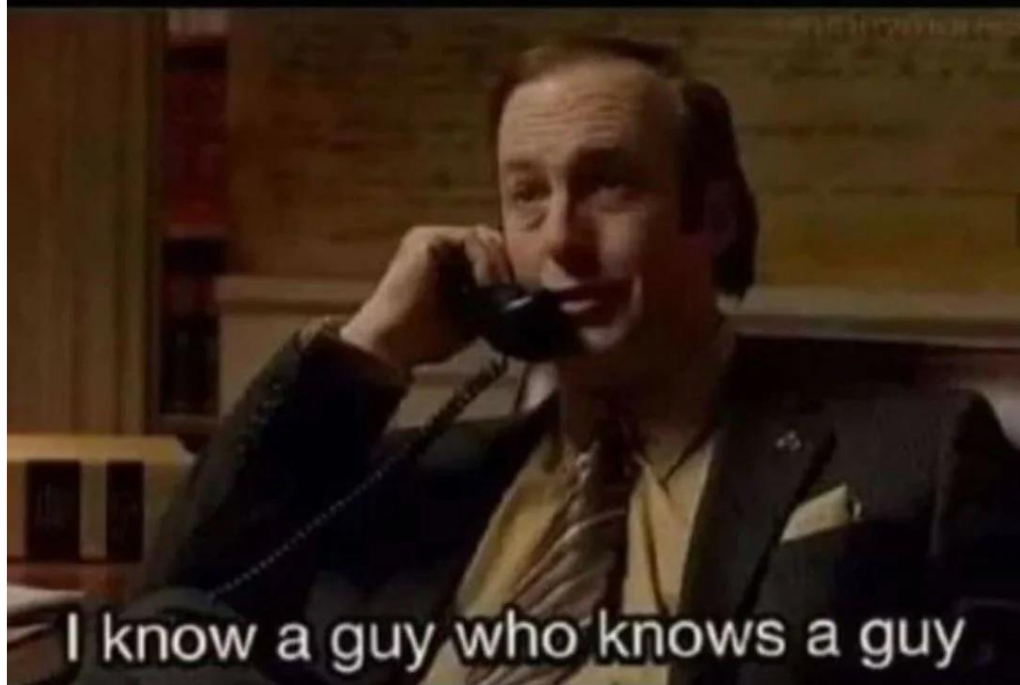
Что такое
структуры данных?



Структуры данных

- «Такой черный ящик, который хранит данные и позволяет нам удобнее пользоваться этими данными»
- Знаем какую-нибудь структуру данных?
- Массив! Простой пример, но очень корректный
- Поехали изучим что-нибудь еще!

Linked List data structures be like:



СВЯЗНЫЙ СПИСОК



СВЯЗНЫЙ СПИСОК

- Каждый элемент списка узел (node)
- Хранит значение
- Знает ссылку на следующий элемент
- В чем различия с массивом?
 - $O(n)$ памяти
 - Нельзя обратиться по индексу
 - Вставка элемента за $O(1)$

СВЯЗНЫЙ СПИСОК

Структура Node

Список задается ссылками на
начало и конец

```
Node:
```

```
    int data
```

```
    Node next
```

```
LinkedList:
```

```
    Node head
```

```
    Node tail
```

```
    int size
```

СВЯЗНЫЙ СПИСОК

Вывод всех элементов списка

```
Node cur = list.head
while cur != null // NULL // None ...
    print(cur)
    cur = cur.next
```

СВЯЗНЫЙ СПИСОК

Вставка элемента в конец

А если список пустой?

```
insert(list, x)
    Node newNode = node(x, null)
    list.tail.next = newNode
    list.tail = newNode
    list.size++
```


СВЯЗНЫЙ СПИСОК

Вставка элемента в конец

```
insert(list, x)
    Node newNode = node(x, null)
    if list.size == 0
        list.head = newNode
        list.tail = newNode
    else
        list.tail.next = newNode
        list.tail = newNode
    list.size++
```

СВЯЗНЫЙ СПИСОК

Удаление элемента без проверки
на дурака

```
erase(list, i)
    if i == 0
        if list.size == 1
            list.head = list.tail = null
        else
            list.head = list.head.next
    else
        ...
```

СВЯЗНЫЙ СПИСОК

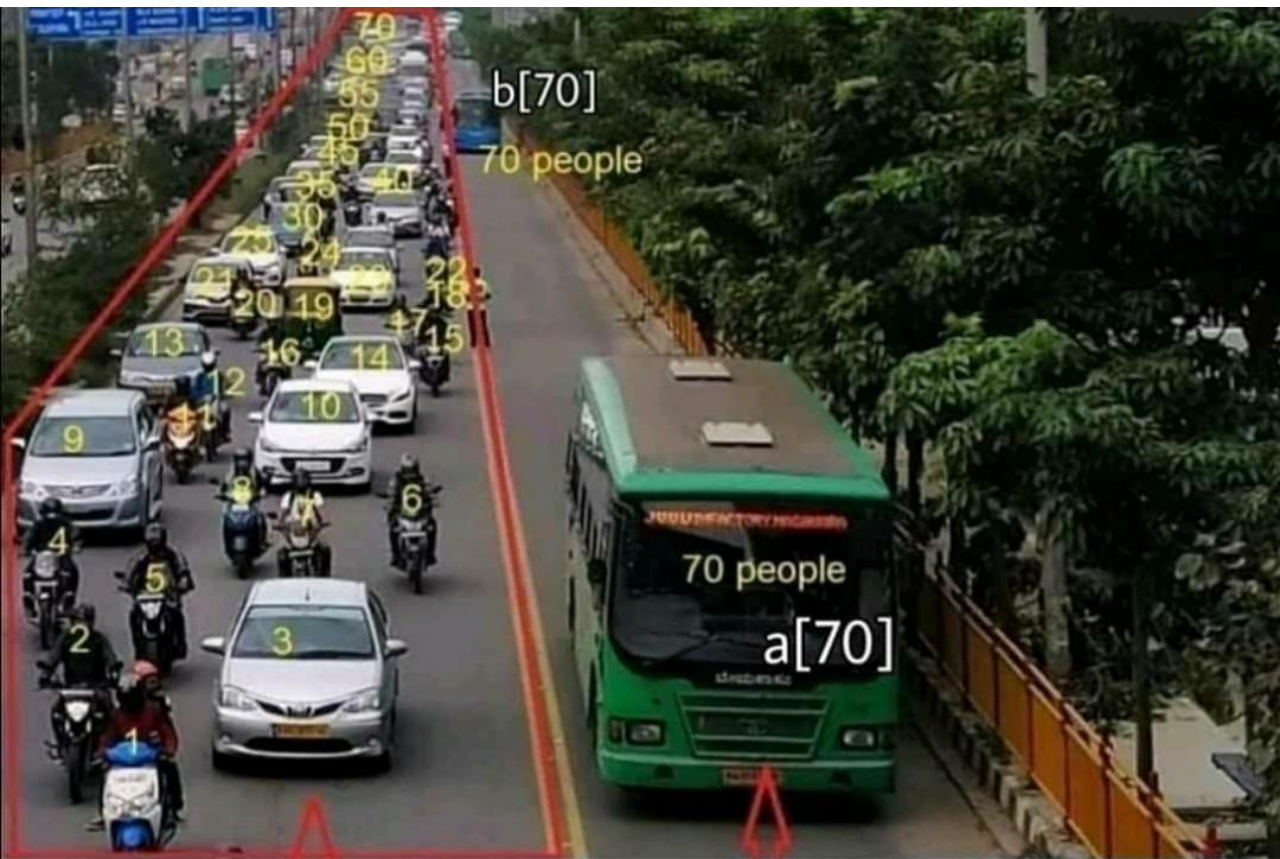
Удаление элемента

```
erase(list, i)
    else
        prev = list.head, cur = prev.next
        j = 1
        while j < i
            prev = cur, cur = prev.next, j++
        prev.next = cur.next
        if cur == list.tail
            list.tail = prev
        list.size--
```



СВЯЗНЫЙ СПИСОК

- На что нужно обратить внимание в зависимости от языка?
 - Нужно чистить память!
- Как удалять из середины, зная ссылку?
 - Двусвязный список



Memory allocation
in Link List

Memory allocation
in Arrays

Саморасширяющийся
массив (Вектор)



Саморасширяющийся массив

- Вот бы можно было пользоваться массивами, но чтобы памяти было $O(n)$
- А так можно сделать!
- Пусть добавляем по элементу в конец массива
- Выделим массив, на константное число элементов
- Если нам этого мало будем увеличивать размер в два раза
- Говорят, что будет работать быстро

Саморасширяющийся массив

Структура + элемент по индексу

Vector:

```
int size  
int capacity  
int[] elements
```

```
get(vector, i)
```

```
if i < 0 or i >= vector.size
```

```
    return null
```

```
    return vector.elements[i]
```

Саморасширяющийся массив

Добавление в конец

А если массив закончился?

```
add(vector, x)
    if vector.size + 1 > vector.capacity
        ensureCapacity(vector)
    vector.elements[vector.size] = x
    vector.size++
```


Саморасширяющийся массив

ensureCapacity

Не забыть почистить память!

```
ensureCapacity(vector)
    vector.capacity *= 2
    newElements = int[vector.capacity]
    for i = 0 to vector.size - 1
        newElements[i] = elements[i]
    vector.elements = newElements
```



Саморасширяющийся массив

- Почему памяти $O(n)$?
- За сколько работают операции?
- Стандартные реализации
 - `vector` (C++)
 - `ArrayList` (Java)
 - `list` (Python)

Саморасширяющийся массив

Удаление из конца без проверки
на дурака

Какое условие `decreaseCapacity`?

```
erase(vector)
    if ???
        decreaseCapacity(vector)
    vector.size--
```

I CAN PEEK FROM A STACK



AND PUSH INTO A QUEUE

memegenerator.net

Стек, Очередь, Дек



Стек

- Представляет собой список элементов, организованных по принципу LIFO (last in — first out)
- Основные методы:
 - push
 - pop
 - size
 - back (top, peek)...
- **Абстракция, существуют различные реализации!**



Стек

- Будем рассматривать те реализации, в которых каждая операция за работает за $O(1)$ и потребляют $O(n)$ памяти
- Стек на саморасширяющемся массиве
- Стек на связном списке



Очередь

- Представляет собой список элементов, организованных по принципу FIFO (first in — first out)
- Основные методы:
 - push
 - pop
 - size
 - front
- **Абстракция, существуют различные реализации!**



Очередь

- Очередь на списке
- Очередь на саморасширяющемся массиве
 - какие проблемы в отличии от стека?
- Очередь на саморасширяющемся циклическом массиве



ДЭК

- Представляет собой список элементов, который позволяет добавлять\удалять элементы, как из начала, так и из конца
- Основные методы:
 - `push_back`, `push_front`
 - `pop_back`, `pop_front`
 - `size`
 - `front`, `back`
- **Абстракция, существуют различные реализации!**



Куча



Приоритетная очередь

- Абстракция, которая хранит множество элементов
- Позволяет добавить элемент множество
- Удаляет элементы в порядке приоритета
 - Например, минимальный элемент множества



Куча

- Способ реализации приоритетной очереди
- Подвешенное двоичное дерево
 - Значение в любой вершине не больше значения в детях (потомках)
 - На i -ом слое 2^i вершин, кроме последнего
 - Последний слой заполнен «слева направо»
- Как хранить такую структуру?
 - Ссылки
 - Массив



Куча

- Куча на массиве
- Корень имеет номер 0
- У вершины с номером i
 - Левый ребенок $2i + 1$
 - Правый ребенок $2i + 2$
- Почему не будет пропусков?
- Как получить номер родителя, зная номер ребенка?
- Тогда i -й элемент кучи — i -й элемент массива



Куча

- Если куча — приоритетная очередь, должны быть операции:
 - insert
 - removeMin
- Как можно их реализовать?
- За сколько они будут работать?

Куча

Вставка элемента

Можно if заменить на and

```
insert(heap, x)
    i = heap.size
    heap.elements[i] = x
    heap.size++

    while i > 0
        if heap.elements[i] <
            heap.elements[(i-1)/2]
            swap(heap.elements, i, (i - 1) / 2)
        i = (i - 1) / 2
```

Куча

Удаление минимального
элемента

Что делать если правого сына
нет?

```
removeMin(heap)

    swap(heap.elements, 0, heap.size - 1)
    heap.size--
    i = 0
    while 2i + 1 < heap.size
        cur = heap.elements[i]
        left = heap.elements[2i + 1]
        right = heap.elements[2i + 2]
        if left < right and left < cur
            swap(heap.elements, i, 2i + 1)
            i = 2i + 1
        else
            ...
```


Куча

Удаление минимального
элемента

Что делать если правого сына
нет?

```
removeMin(heap)

    swap(heap.elements, 0, heap.size - 1)
    heap.size--
    i = 0
    while 2i + 1 < heap.size
        cur = heap.elements[i]
        left = heap.elements[2i + 1]
        if 2i + 2 == heap.size
            right = INF
        else
            right = heap.elements[2i + 2]
        if left < right and left < cur
            ...
```



Сортировка кучей

- Вспомним сортировку выбором
- Вставили все элементы в кучу, последовательно выбираем минимум
- $O(n \log n)$
- Как без доп. памяти?



Построение кучи по массиву

- Последовательный insert всех элементов
 - $O(n \log n)$
- Можно ли быстрее
 - Идем с конца и просеиваем вниз
 - Будет $O(n)$
 - Почему?



Bce!