4. Структуры данных

Вопрос 357. Как оценивать и сравнивать между собой скорость работы алгоритмов?

Ответ:

(НЕ ИСПОЛЬЗУЕТСЯ) Сравнение числа арифметичесих действий

(НЕ ИСПОЛЬЗУЕТСЯ) Измерение фактического времени выполнения программы (не используется потому, что все зависит от параметров компьютера и количества входных данных, а все компьютеры разные)

Big O - скорость работы программы как верхнаяя граница динамики изменения вычислительной сложности алгоритма в зависимости от размера входных данных.

_____

Вопрос 358. Что такое Big O?

Ответ: Big O - это зависимость изменения вычислительной сложности от размера входных данных

_____

Вопрос 359. Какие ресурсы компьютера требуются для выполнения алгоритма?

Ответ:

- потребляемая память - является самым главным

- процессорное время

_____

Вопрос 360. Как записывается Big O?

Ответ: В общем случае Big O записывается следующим образом, как пример, O(n^2) - это означает, что верхняя граница динамики вычислительной сложности алгоритма составляет n^2 - это позволяет адекватно представить изменение объема вычисления при изменении размера входных данных.

_____

Вопрос 361. Перечислите классические асимптотики для алгоритмов?

Ответ:

- O(Const) или O(1) - за константное время, самый быстрый

- O(log n) - логарифмическая сложность

- O(n) - линейная сложность (однопроходные алгоритмы - один цикл)

- O(n + m) - линейная сложность (многопроходные алгоритмы - множество циклов без вложенных циклов)

- O (n * log n) - квадратично-логарифмическая сложность (быстрая сортировка)

- O(n ^ 2) - квадратичная сложность (есть вложенные циклы)

- O(n ^ 3) - кубическая сложность

- O (2 ^ n) - экспоненциальная сложность (NP-полные задачи)

- O (n!) - факториальная сложность (NP-полные задачи)

———

Вопрос 362. Расскажите про сложность алгоритма O(Const) или O(1)?

Ответ: В Python мы выполняем операции (присвоение, вывод в консоль, переопределение), которые не зависят от размера входных данных:

var_a = 10 # O(1) - переменная ссылается на объект 10

inf = float('inf') # O(1) - переменная ссылается на + бесконечность

print(inf) # O(1) - в консоль выводится значение переменной

Каждая операция выполняется за одно и тоже константное время** - её константа 1 записывается в скобках. 1 - выполнение 1 операции. O от любой константы всегда будет O(1)

O(1 + 1 + 1) = O(3) = O(1)

———

Вопрос 363. Расскажите про сложность алгоритма O(n)?

Ответ: Если дан цикл for число итераций будет зависеть от размерности массива(списка, кортежа). Чем больше n, тем больше итераций. Цикл зависит от размера входных данных.

lst = [1, 4, 10, -5, 0, 2, 3...] # O(1) - присвоение переменной значения

for x in lst: # число итераций зависит от объема входных данных

  print(x) # O(n) - вывод в консоль в цикле

  x += 1   # O(n) - инкремент элемента списка без сохранения

В цикле выполняются две операции O(n). По правилу сложений:

O(1) + O(n) + O(n) = O(1 + n + n) = O(1 + 2n) = O(n)

———

Вопрос 364. Какая сложность Big(O) если у нас есть несколько циклов от одного и того же массива?

Ответ:

```
lst = [1, 4, 10, -5, 0, 2, 3...] # O(1)
for i, d in enumerate(lst):
    lst[i] += 1 # O(n)
for x in lst:
    print(x) # O(n)
```

Получаем: O(1) + O(n) + O(n) = O(1 + n + n) = O(1 + 2n) = O(n)

———

Вопрос 365. Какая сложность Big(O) если у нас есть несколько циклов от двух разных массивов?

Ответ:

```
a = [1, 4, 10, -5, 0, 2, 3...]  # O(1)
b = [9, 8, 7, 6, 5, 4, 3]  # O(1)
for i, d in enumerate(a):
    a[i] += 1  # O(n) - цикл для множества a

for x in b:
    print(x)  # O(m) - цикл для множества b
```

O(1) + O(n) + O(1) + O(m) = O(n + m + 2) = O(n + m)

———

Вопрос 366. Расскажите про сложность алгоритма O(n^2)?

Ответ:

```
a = [(1,2), (3,4), (5,6)...] # O(1)
for i in a: # O(n)
    for j in i: # O(n)
        print(j)
```

При таком вложении существует умножение:

O(1) + O(n * n) = O(n^2 + 1) = O(n^2)

———

Вопрос 367. Расскажите про сложность алгоритма O(log n)?

Ответ:

Логарифмическая сложность O(log n) уступает только O(1) в своем выполнении.

ТАМ ГДЕ ПРОИСХОДИТ ИЕРАРХИЧЕСКОЕ РАЗБИЕНИЕ ДАННЫХ И ПОСЛЕДУЮЩАЯ ОБРАБОТКА ОСТАВШИХСЯ ЧАСТЕЙ, ТО ПОЛУЧАЕТСЯ ЛОГАРИФМИЧЕСКАЯ СЛОЖНОСТЬ АЛГОРИТМА.

Пример: алгоритм бинарного поиска

———

Вопрос 368. Расскажите про сложность алгоритма O(n!)?

Ответ: Существуют NP-полные задачи, в которых приходится делать полный NP-перебор всех вариантов, чтобы найти искомое решение. Это факториальная сложность O(n!) - надо перебрать все возможные варианты. Эта самая высокая сложность алгоритма. Её надо избегать и по возможности использовать не точное, но близкое к точному решению.

———

Вопрос 369. Что такое массив?

Ответ: Массив — это непрерывный участок памяти, содержащий последовательность объектов одинакового типа, обозначаемый одним именем.

Вопрос 370. Что такое статический массив?

Ответ: это массив, размер которого НЕ может изменяться во время исполнения программы.

———

Вопрос 371. Перечислите отличительные признаки статического массива:

Ответ:

1. Все элемента статического массива должны иметь единый тип данных
2. У статического массива фиксированное число элементов
3. Все элементы статического массива располагаются в памяти компьютера последовательно друг за другом без каких-либо пропусков, начиная с начального адреса или с начальной ячейки

———

Вопрос 372. Является ли кортеж Python (tuple) статическим массивом?

Ответ: нет, не является. Потому что кортеж – это неизменяемая коллекция, которая состоит из ссылок на объекты в памяти компьютера и эти ссылки не могут ссылаться на другие объекты, а в статическом массиве мы можете менять значение элементов массива.

———

Вопрос 373. Как в Python можно реализовать статический массив?

Ответ:

В Python статический массив можно реализовать с помощью модуля array. Модуль array предоставляет тип данных, который представляет собой упакованный массив однородных элементов.

Вот пример создания статического массива с использованием модуля array:

```
import array

# Создание статического массива целых чисел
static_array = array.array('i', [1, 2, 3, 4, 5])

# Доступ к элементам массива
print(static_array[0])  # Вывод: 1
print(static_array[2])  # Вывод: 3
```

———

Вопрос 374. Преимущества статических массивов?

Ответ:

- скорость доступа к произвольному элементу O(1) для записи или чтения значения

- просто реализуется и удобен для небольших наборов данных

———

Вопрос 375. Недостатки статических массивов?

Ответ:

- хранение данных выполняется в непрерывной области памяти. Не всегда эффективно для очень больших объемов данных

- Статический массив не может менять число своих элементов в процессе работы программы. Если зарезервированного места окажется недостаточно, то данные могут потеряться

- вставка и удаление элементов выполняется за время O(n). Может быть критично при больших n

___

Вопрос 376. Что такое динамический массив?

Ответ: Динамический массив - это массив, который может менять число своих элементов в процессе работы программы. Динамические массивы реализуются на основе обычных статических массивов и хранят данные в непрерывной области памяти. Операции доступа выполняются O(1).

___

Вопрос 377. Отличие динамического массива от статического?

Ответ: У статического массива неизменяемое число элементов. Программист должен указать сколько элементов должен иметь статический массив и не всегда можно указать разумное значение. Если создать динамический массив размером в 100 элементов и вдруг этого размера недостаточно размер памяти будет увеличен

___

Вопрос 378. Объясните механизм увеличения размера массива?

Ответ:

Если все элементы оказались заняты, а мы хотим в динамический массив добавить еще один элемент - тогда выделяется память под новый статический массив длиною в 2 -3 раза больше первоначального maxCapacity * 2 и туда копируются все элементы первого массива.

Также удвоение физического размера может происходить и когда он заполнен на 90% процентов

___

Вопрос 379. Зачем под новый массив выделять именно в 2 раза больше элементов, чем в прежний, а не увеличить его на 1-2 элемента?

Ответ:

Это связано с тем, что если нам не хватило места под первый массив, то с высокой вероятностью нам не хватит места и для нового массива с одним дополнительным элементом. А значит операции выделения памяти и копирования всех предыдущих значений в новый массив снова придётся повторять, это приводит к неоправданному расходу процессорного времени.

___

Вопрос 380. Можно ли считать список Python (list) динамическим массивом?

Ответ: да, можно. В Python динамический массив представлен списком list. При увеличении размера списка не всегда происходит его удвоение. Так никакой памяти не хватит. Если в списке становится меньше элементов, то выделяемая память уменьшается. В Python внутри списка хранятся не сами данные, а ссылки на эти данные.

Сами ссылки - это переменные фиксированного размера (в памяти), которые хранят адрес объекта, на который ссылаются. А ссылки могут быть связаны с объектом разного типа. Отсюда получается, что список содержит только ссылки, ведущие на разные объекты. В динамическом

массиве ссылок на следующие элементы отсутствуют, т.к. все они следуют друг за другом в непрерывной области памяти. Но изначально длина динамического массива не равна нулю, занимает сколько-то байт и кроме того, там еще есть вспомогательные переменные, отвечающие за логический и физический размеры этого массива и некоторые другие служебные.

———

Вопрос 381. Перечислите несколько методов списков Python и назовите их сложность Big O?

Ответ:

Operation	Examples	Complexity class	
		Average case	Amortised Worst case
Append	**l.append(item)**	**O(1)**	**O(1)**
Clear	l.clear()	O(1)	O(1)
Containment	item in/not in l	O(N)	O(N)
Copy	l.copy()	O(N)	O(N)
Delete	del l[i]	O(N)	O(N)
Extend	l.extend(...)	O(N)	O(N)
Equality	l1==l2, l1!=l2	O(N)	O(N)
Index	l[i]	O(1)	O(1)
Iteration	for item in l:	O(N)	O(N)
Length	len(l)	O(1)	O(1)
Multiply	k*l	O(k*N)	O(k*N)
Min, Max	min(l), max(l)	O(N)	O(N)
Pop from end	l.pop(-1)	O(1)	O(1)
Pop intermediate	l.pop(item)	O(N)	O(N)
Remove	l.remove(...)	O(N)	O(N)
Reverse	l.reverse()	O(N)	O(N)
Slice	l[x:y]	O(y-x)	O(y-x)
Sort	l.sort()	O(N*log(N))	O(N*log(N))
Store	l[i]=item	O(1)	O(1)

———

Вопрос 382. Что происходит при создании среза списка Python?

Ответ: Создается новый динамический массив в памяти и туда копируются данные из указанного среза.

———

Вопрос 383. Какой главный недостаток динамического массива и как его можно разрешить с помощью другой структуры данных?

Ответ:

Проблема динамических массивов в том, что в памяти создается новый массив в 2 раза больше, при превышении лимита элементов - O(n), причем все данные хранятся в непрерывной области памяти.

Эту проблему можно разрешить:

если число элементов подходит к пределу - в где-то памяти выделяется такой же физический размер, но данные находятся в разных областях. Скорость добавления элементов происходит за время O(1)

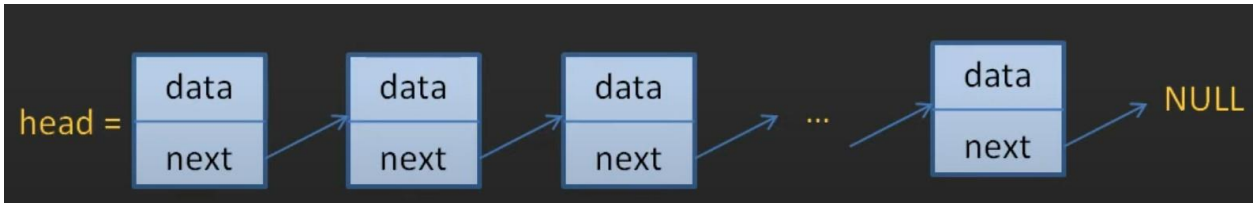
По последний элемент массива - это будет ссылка на новую область памяти, куда он будет перенесен. Реализуется с помощью односвязного списка.

___

Вопрос 384. Что такое односвязный список?

Ответ:

Односвязные список- это структура данных, в которой элементы ссылаются друг на друга по-порядку в одном направлении, начиная от первого и до последнего.



___

Вопрос 385. Расскажите механизм добавления нового элемента в конец односвязного списка?

Ответ:

На первый элемент односвязного списка ссылается переменная head

На последний элемент односвязного списка ссылается переменная tail

Чтобы добавить элемент в конец - его нужно создать

По ссылке tail мы выбераем последний элемент этого списка, и ссылку next мы переопределяем на вновь добавляемый объект.

И в конце надо переменной tail присвоить значение node.

Скорость выполнения операций составляет O(1)

___

Вопрос 386. Перечислите все операции с односвязным списком и назовите их сложность?

Ответ:

	Команда	Big O
Добавление в конец	push_back()	O(1)
Добавление в начало	push_front()	O(1)
Удаление с конца	pop_back()	O(n)
Удаление с начала	pop_front()	O(1)
Вставка элемента	insert()	O(n)
Удаление промеж. элементов	erase()	O(n)
Доступ к элементу	iterator	O(n)

———

Вопрос 387. Как реализовать односвязный список в Python с помощью ООП?

Ответ:

```
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None

class LinkedList:
    def __init__(self):
        self.head = None

    def append(self, data):
        new_node = Node(data)
        if self.head is None:
            self.head = new_node
        else:
            last_node = self.head
            while last_node.next:
                last_node = last_node.next
            last_node.next = new_node

    def display(self):
        current_node = self.head
        while current_node:
            print(current_node.data, end=" -> ")
            current_node = current_node.next
        print("None")
```

———

Вопрос 388. Что такое двусвязный список7

Ответ:

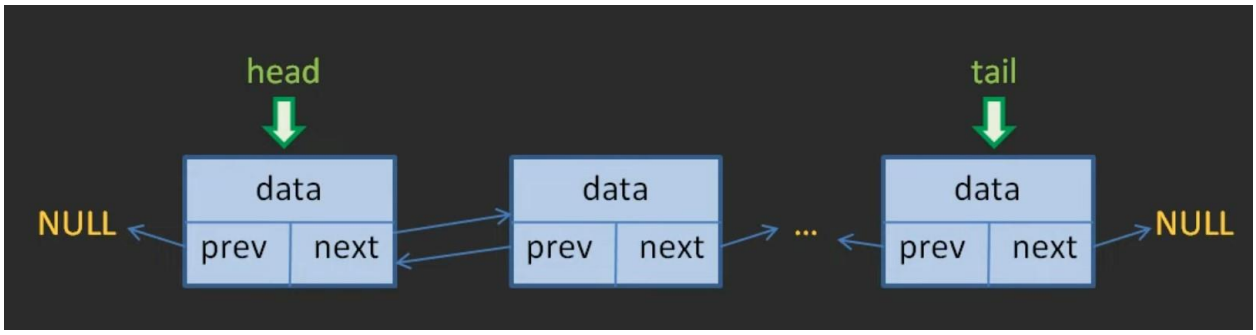
Двусвязный список - это такая структура данных, каждый элемент которого имеет 2 ссылки - на предыдущий (prev) и последующий (next) элементы.

head ссылается на первый элемент

tail ссылается на последний

head.prev = NULL

tail.next = NULL



___

Вопрос 389. Перечислите все возможные операции на двусвязном списке и их сложность Big O?

Ответ:

	Команда	Big O
Добавление в конец	push_back()	O(1)
Добавление в начало	push_front()	O(1)
Удаление с конца	pop_back()	O(1)
Удаление с начала	pop_front()	O(1)
Вставка элемента	insert()	O(n)
Удаление промеж. элементов	erase()	O(n)
Доступ к элементу	at()	O(n)

___

Вопрос 390. Назовите несколько примеров, где используется двусвязный список?

Ответ:

- хранение информации по товарам Интернет-магазина

- хранение котировок пары доллар/евро по дням

- хранение записей (данных) в базе данных

- хранение значений функции f(x) при постоянно увеличивающемся аргументе x

____

Вопрос 391. Реализуйте двусвязный список на Python с помощью ООП?

Ответ:

Для этого объявите класс LinkedList, который будет представлять связный список в целом и иметь набор следующих методов:

add_obj(self, obj) - добавление нового объекта obj класса ObjList в конец связного списка;
remove_obj(self) - удаление последнего объекта из связного списка;
get_data(self) - получение списка из строк локального свойства __data всех объектов связного списка.

И в каждом объекте этого класса должны создаваться локальные публичные атрибуты:

head - ссылка на первый объект связного списка (если список пустой, то head = None);
tail - ссылка на последний объект связного списка (если список пустой, то tail = None).

Объекты класса ObjList должны иметь следующий набор приватных локальных свойств:

__next - ссылка на следующий объект связного списка (если следующего объекта нет, то __next = None);
__prev - ссылка на предыдущий объект связного списка (если предыдущего объекта нет, то __prev = None);
__data - строка с данными.

Также в классе ObjList должны быть реализованы следующие сеттеры и геттеры:

set_next(self, obj) - изменение приватного свойства __next на значение obj;
set_prev(self, obj) - изменение приватного свойства __prev на значение obj;
get_next(self) - получение значения приватного свойства __next;
get_prev(self) - получение значения приватного свойства __prev;
set_data(self, data) - изменение приватного свойства __data на значение data;
get_data(self) - получение значения приватного свойства __data.

Сам код:

```
class LinkedList:

    def __init__(self):
        self.head = None
        self.tail = None

    def add_obj(self, obj):
        if self.tail:
            self.tail.set_next(obj)
        obj.set_prev(self.tail)
        self.tail = obj
        if not self.head:
            self.head = obj

    def remove_obj(self):
        if self.tail is None:
            return
        prev = self.tail.get_prev()
        if prev:
            prev.set_next(None)
        self.tail = prev
        if self.tail is None:
            self.head = None

    def get_data(self):
        s = []
        h = self.head
        while h:
            s.append(h.get_data())
            h = h.get_next()
        return s

class ObjList:

    def __init__(self, data):
        self.__data = data
        self.__next = self.__prev = None

    def set_next(self, obj):
        self.__next = obj

    def set_prev(self, obj):
        self.__prev = obj

    def get_next(self):
        return self.__next

    def get_prev(self):
        return self.__prev

    def set_data(self, data):
        self.__data = data
```

```
    def get_data(self):
        return self.__data
```

Вопрос 392. что такое абстрактные структуры данных?

Ответ: это математические модели данных, которые определяют операции, которые можно выполнять с этими данными, но не определяют конкретную реализацию этих операций.

Вопрос 393. Перечислите все абстрактные структуры данных?

Ответ:

Примеры абстрактных структур данных:

Стек (Stack): LIFO (Last In, First Out) структура данных, где элементы добавляются и удаляются только с одного конца.

Очередь (Queue): FIFO (First In, First Out) структура данных, где элементы добавляются в конец и удаляются из начала.

Список (List): Структура данных, которая позволяет хранить и управлять коллекцией элементов.

Вопрос 394. Что такое очередь?

Ответ: Очередь - это абстрактная структура данных, организованная по принципу "первый пришел - первый вышел" [FIFO] или "первый пришел - после вышел"

Вопрос 395. Где используется очередь, приведите несколько примеров?

Ответ:

1. Буфер приема и передачи данных
2. Порядок обработки заказов в интернет-магазине

Вопрос 396. С помощью каких структур данных можно реализовать очередь?

Ответ:

- односвязные списки

- двусвязные списки

- гибридные структуры двусвязных списков и динамического массива

- динамический массив

Вопрос 397. Что такое deque?

Ответ: очередь, в основе которой лежит двусвязный список

———

Вопрос 398. Расскажите о реализации очереди в Python с помощью стандартной библиотеки collections?

Ответ: В стандартной библиотеке Python collections есть класс deque. Этот deque реализует двусвязный список со всеми наборами операций - добавления удаления граничных элементов.

Метод	Описание	Объем вычислений
append()	Добавление нового элемента справа (в конец списка)	O(1)
appendleft()	Добавление нового элемента слева (в начало списка)	O(1)
pop()	Удаление элемента справа (с конца списка)	O(1)
popleft()	Удаление элемента слева (с начала списка)	O(1)
extend()	Добавление нескольких m элементов справа (в конец списка)	O(m)
extendleft()	Добавление нескольких m элементов слева (в начало списка)	O(m)
insert()	Вставка нового элемента в произвольную позицию	O(n)
remove()	Удаление элемента списка с указанным значением (удаляется первый найденный элемент с заданным значением)	O(n)
clear()	Удаление всех элементов списка	O(1)
copy()	Создание копии двусвязного списка	O(n)

Задать очередь определенной длины

```
from collections import deque


dq = deque([1, 2, 3, 4, 5], maxlen=5)
print(dq)
```

———

Вопрос 399. Что такое стек?

Ответ:

Стек - это такая структура данных, построенная по принципу очереди LIFO, но отличающаяся тем, что мы в стеке мы можем читать, добавлять и удалять значение ТОЛЬКО ВЕРХНЕГО элемента, но перейти к произвольному элементу мы не можем, т.к. стек не поддерживает такую операцию.

———

Вопрос 400. Зачем стек нужен как отдельная структура данных?

Ответ:

На практике встречается много задач, где данные удобно представлять в виде стека,т.к. в стеке все операции O(1), а очередь LIFO будет избыточной - в ней есть O(n)

На базе каких структур данных реализуется стек?

Стек реализуется на базе односвязного и двусвязного списков. На вершину ставится указатель top и реализуется добавление и удаление элемента на вершине

Но также стек можно реализовать на базе динамического массива с добавлением и удалением последних элементов. Понятие "верх стека" – условно

___

Вопрос 401. Приведите практические примеры использования стека?

Ответ:

- заливка областей произвольной формы

- хранение истории посещения пользователем веб-страниц (для функционала кнопки back в браузере)

- хранение последних выполненных команд в графическом редакторе

___

Вопрос 402. Перечислите все операции, которые можно выполнять для стека?

Ответ:

Метод	Описание	Сложность
top()	Возвращает значение верхнего элемента стека	O(1)
push()	Добавляет новый элемент на вершину стека	O(1)
pop()	Удаляет элемент с вершины стека	O(1)
size()	Возвращает число элементов в стеке	O(1)
empty()	Возвращает True, если стек пуст, и False в противном случае	O(1)

___

Вопрос 403. Как реализуется стек в Python с использованием стандартной библиотеки collections?

Ответ:

ПРИ БОЛЬШИХ ОБЪЁМАХ ИНФОРМАЦИИ, стек реализуется специальным классом из стандартной библиотеки collections

Пример стека, где верхний элемент - последний элемент

```
from collections import deque

stack = deque([1, 2, 3])
stack.append(4)
stack.append(5)
value = stack.pop()
print(value, stack)
```

Если многопоточное программирование, то:

import queue

q = queue.LifoQueue(maxsize=0)

___

Вопрос 404. что такое дерево в программировании?

Ответ: Дерево - это связный граф, в котором нет циклов.

1) Между любыми вершинами есть только одна цепь (маршрут, где нет повторяющихся ребер)

2) нет простых циклов (нет повторяющихся ребер)

Как только у дерева появляется корень - оно становится иерархичным.

———

Вопрос 405. Что такое бинарное дерево?

Ответ:

Бинарное дерево — иерархическая структура данных, в которой каждый узел имеет не более двух потомков (детей).

———

Вопрос 406. Из чего состоит бинарное дерево?

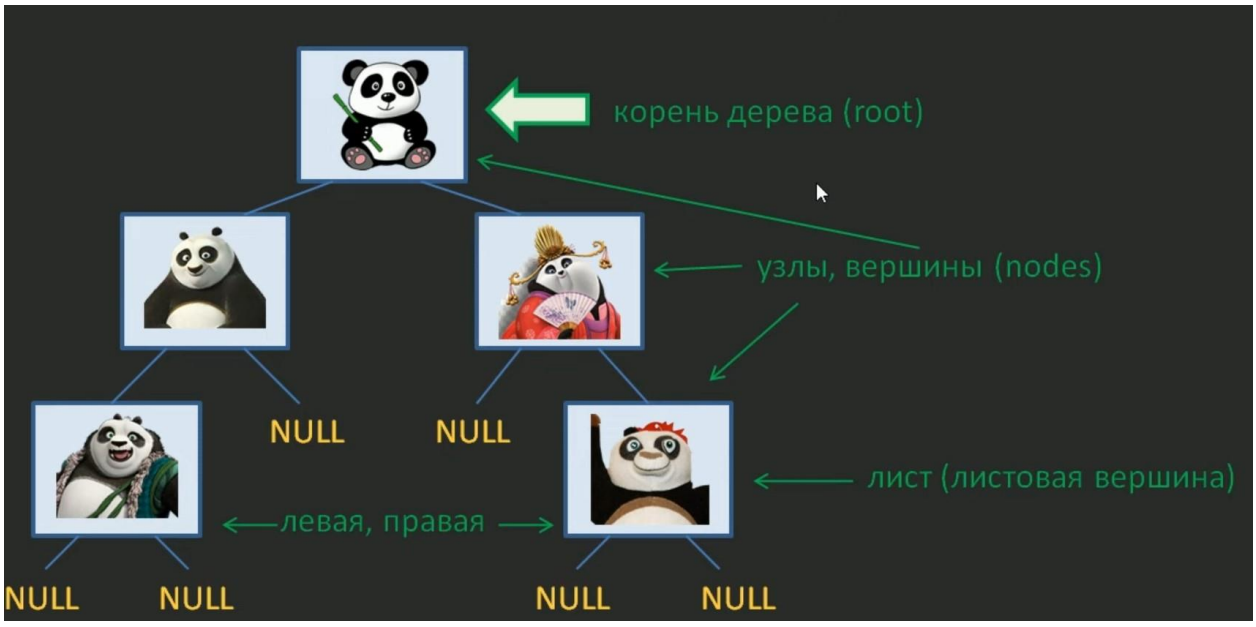
Ответ:

Бинарное дерево состоит из:

root - начальная вершина дерева, от которой следуют все остальные, называют корнем дерева.

nodes - узлы дерева (элементы, вершины). Узел слева - левый, узел справа - правой.

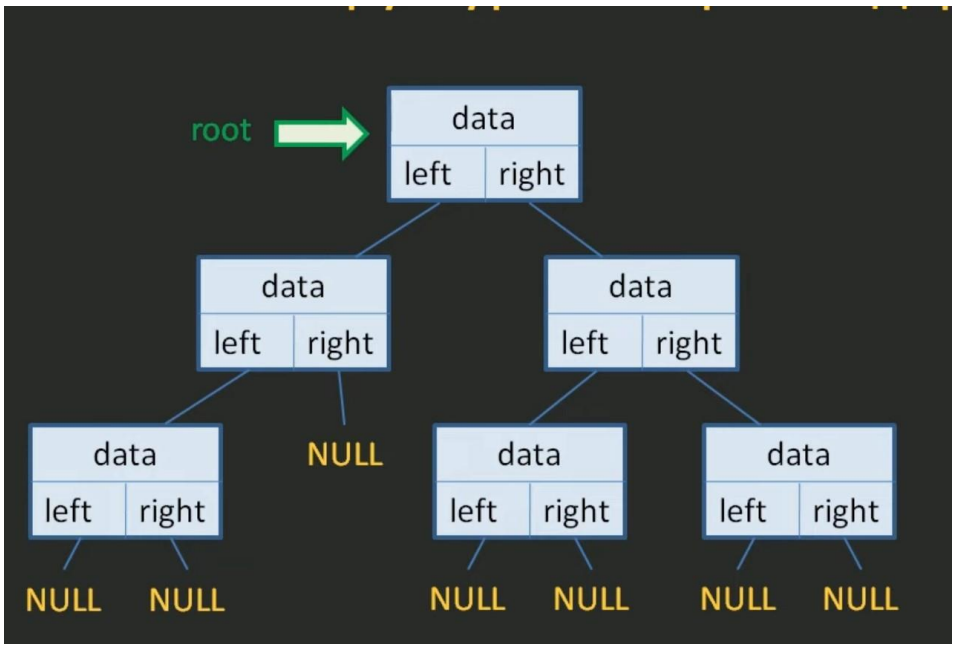
Листовая вершина- вершина, у которой нет ни одного потомка (ссылки left и right ведут на значение NULL)



———

Вопрос 407. Расскажите структуру бинарного дерева

Ответ:



data - поле для хранения каких-либо данных

left - указатель, который ссылается на левую вершину

right - указатель, который ссылается на правую вершину

root - указатель вершины дерева. Через этот указатель происходит вся работа с бинарным деревом. Из root всегда можно пройти для любого другого элемента в этом дереве.

___

Вопрос 408. Что делается через указатель root в бинарном дереве?

Ответ:

- удаление элементов

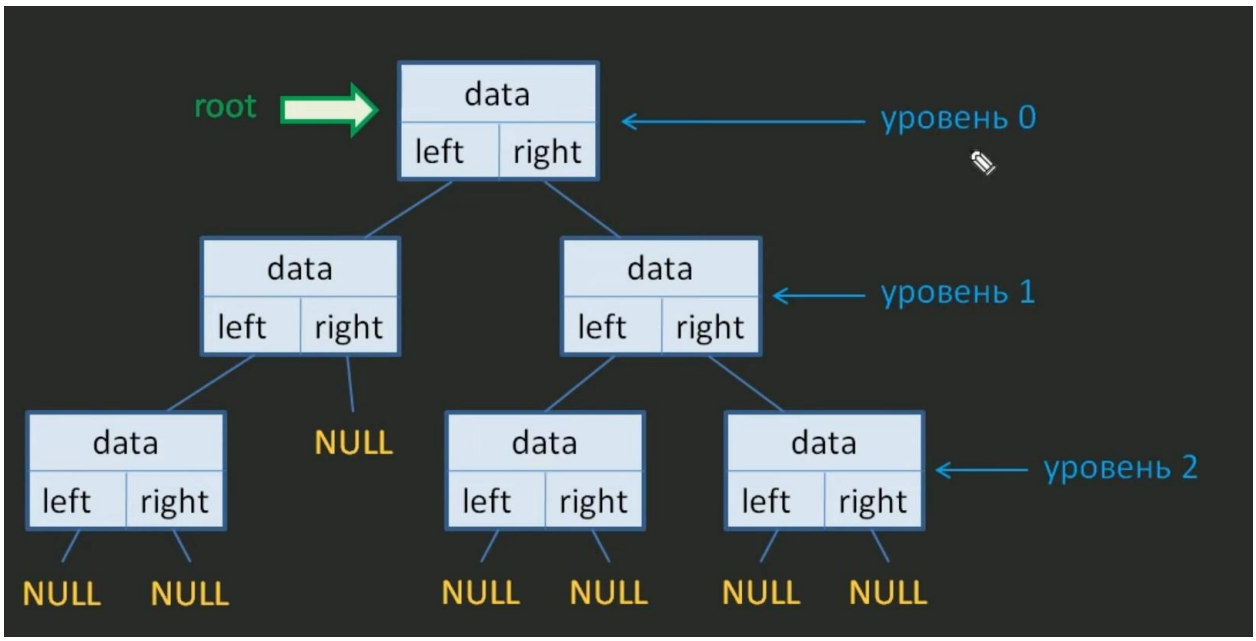
- добавление элементов

- обход всех узлов дерева

___

Вопрос 409. Что такое уровни бинарного дерева?

Ответ:



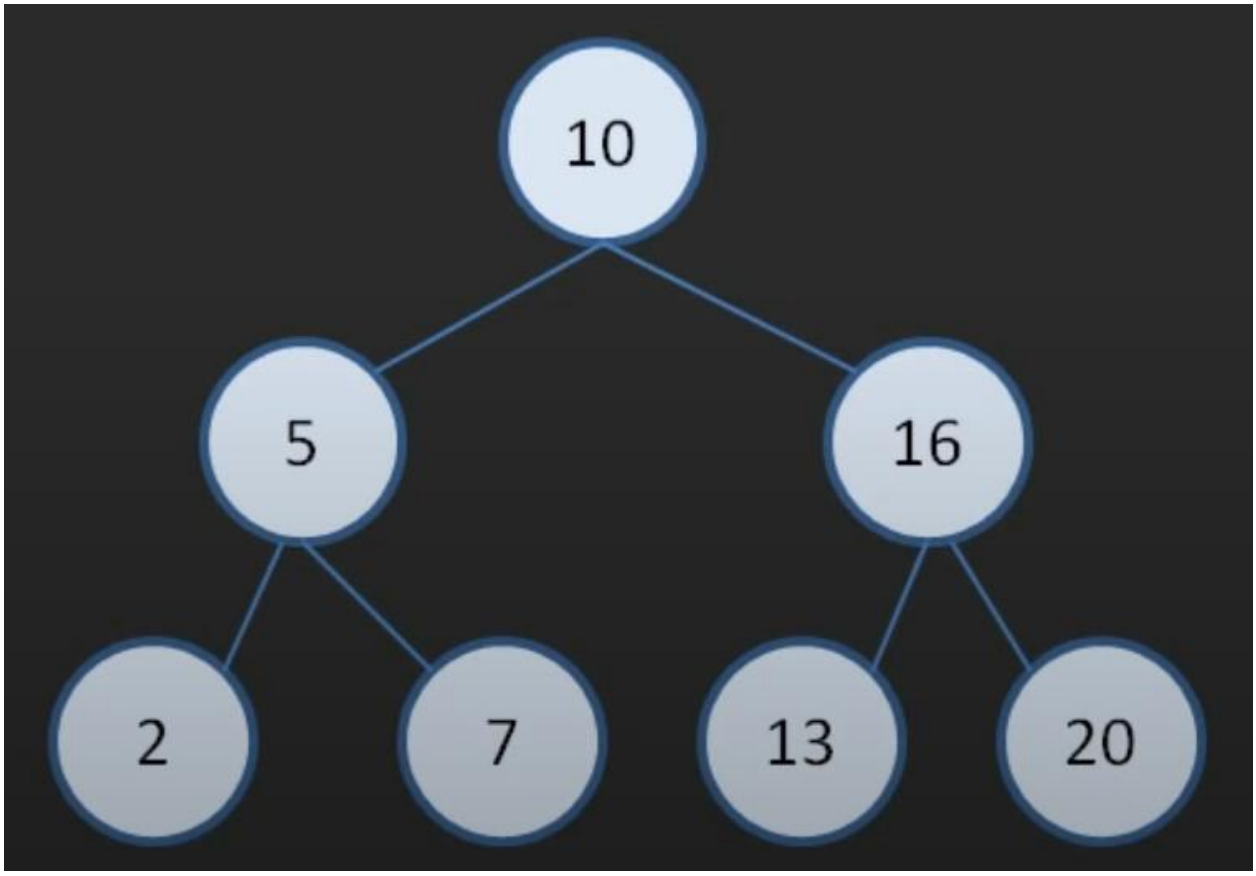
———

Вопрос 410. Расскажите о правилах формирования бинарного дерева?

Ответ:

- если добавляемое значение меньше значения в родительском узле, то новая вершина добавляется в левую ветвь, иначе - в правую

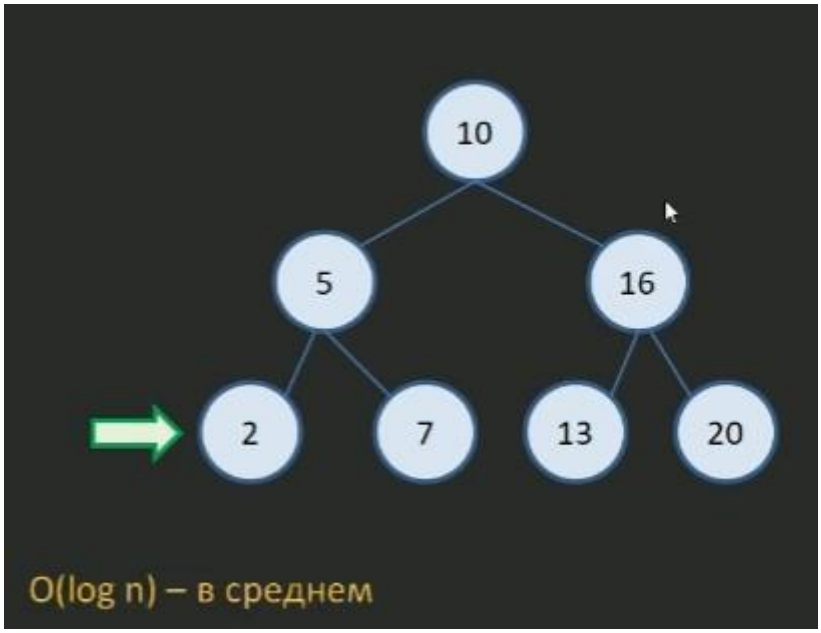
- если добавляемое значение уже присутствует в дереве, то оно игнорируется (то есть, дубли отсутствуют)



———

Вопрос 411. Расскажите о поиске значений в бинарном дереве и сложность Big O?

Ответ: Поиск в бинарном дереве сильно отличается от поиска в динамическом массиве тем, что мы сразу делим дерево на 2 части, т.е. у нас уже отпадает половина элементов

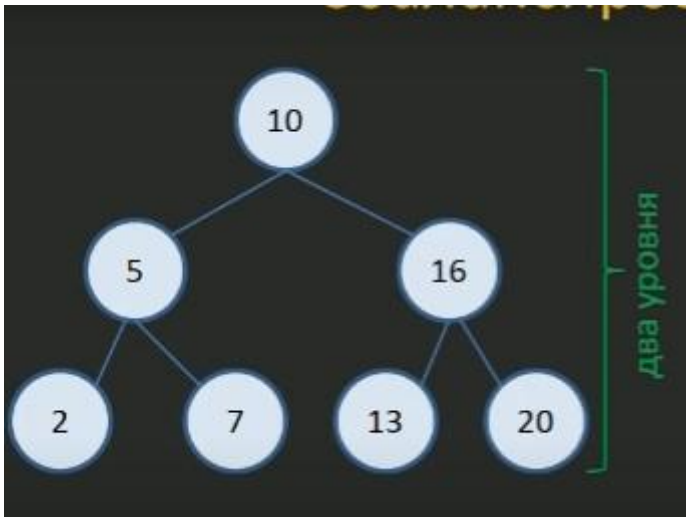


———

Вопрос 412. Что такое сбалансированное бинарное дерево?

Ответ:

Сбалансированное дерево - это дерево, если в нем поддеревья от одной вершины отличаются не более чем на один уровень. Именно в сбалансированном времени сложность поиска O(log(n))

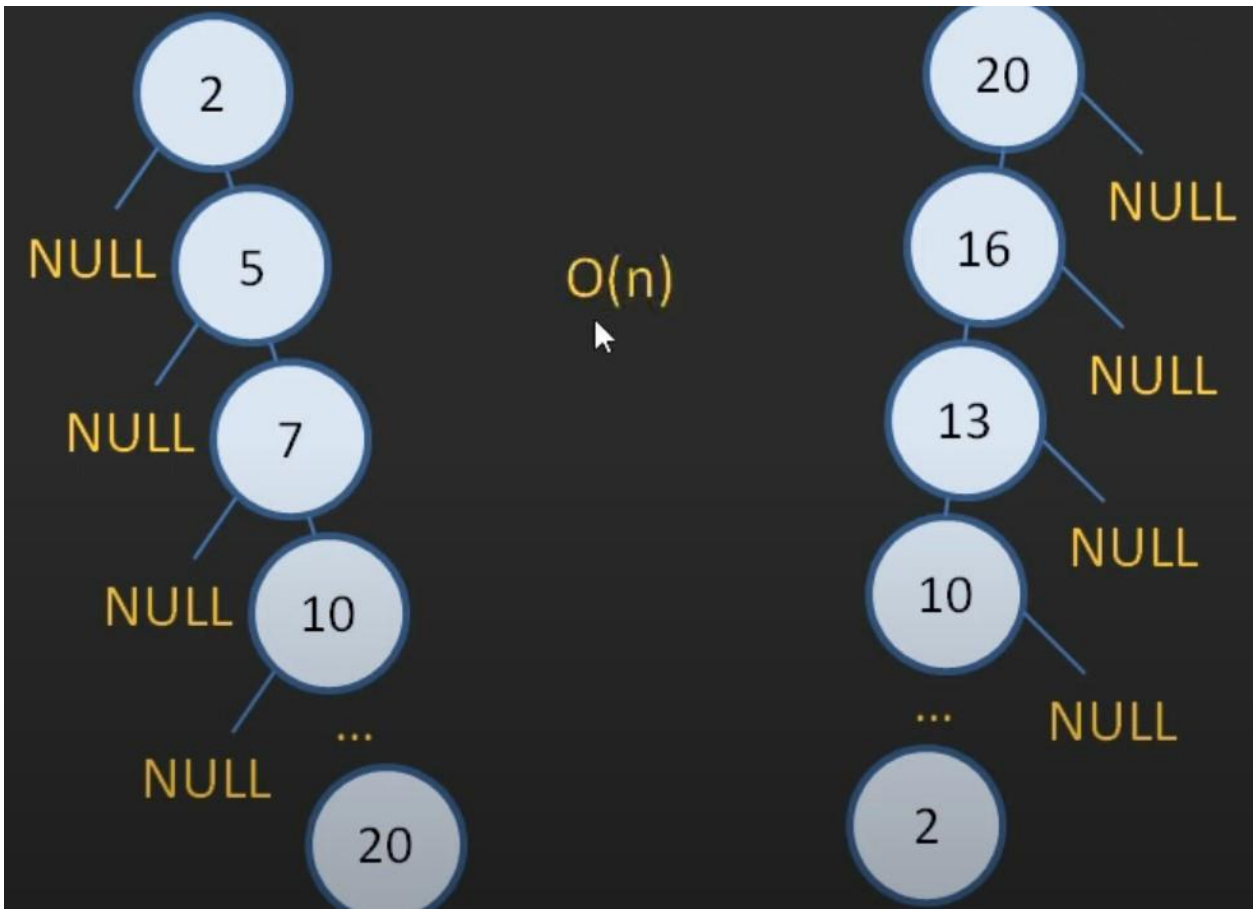


———

Вопрос 413. Что такое несбалансированное бинарное дерево?

Ответ:

Несбалансированные деревья. Если числа при формировании дерева поступают в порядке возрастания или убывания, то деревья выраждаются в односвязный список. И сложность поиска составляет O(n)



___

Вопрос 414. Перечислите методы балансировки деревьев?

Ответ:

- АВЛ-дерево

- красно-черное дерево

- расширяющееся или косое дерево

___

Вопрос 415. Перечислите способы обхода бинарного дерева?

Ответ:

- обход вершин дерева в ширину (breadth-first)

- обход вершин дерева в глубину (3 вида данного обхода)

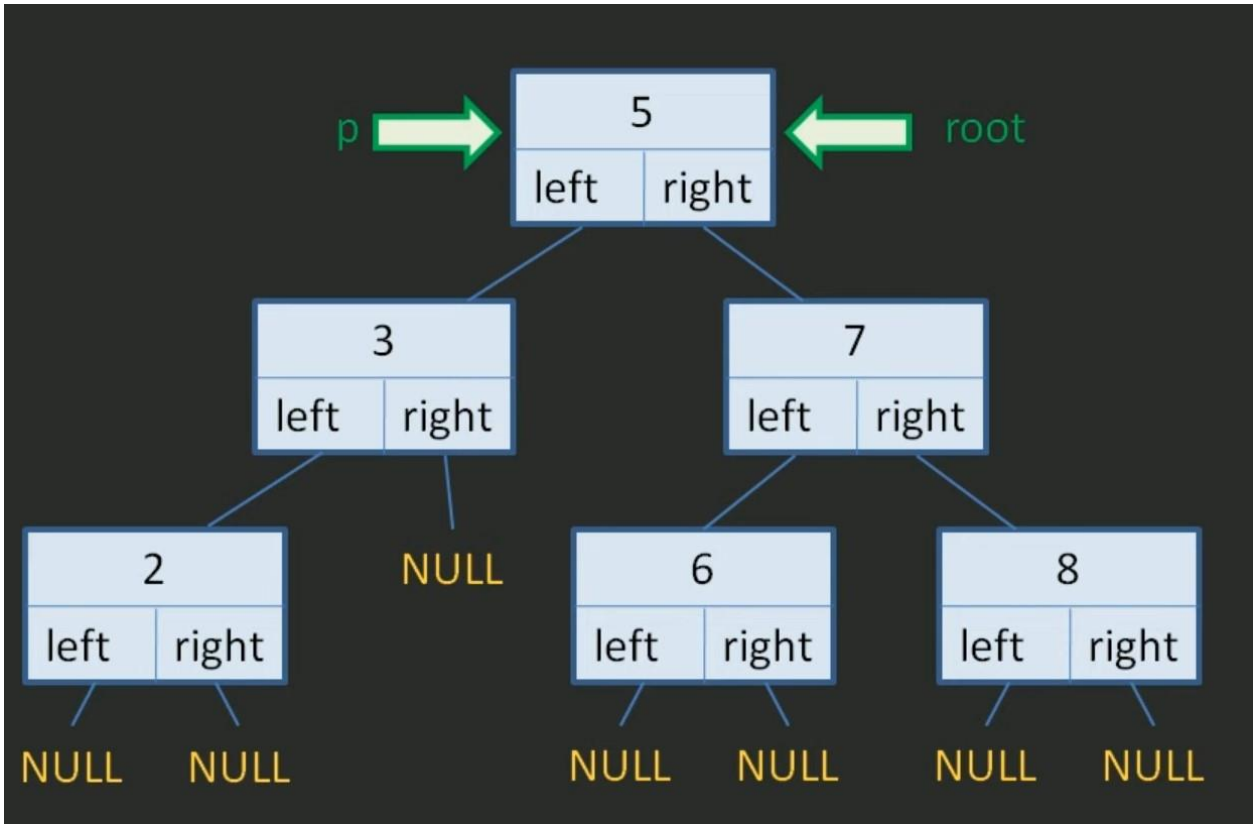
___

Вопрос 416. Опишите обход вершин бинарного дерева в ширину?

Ответ:

Перебор узлов дерева выполняется по уровням слева-направо, начиная от корневой вершины. Пока у нас эти узлы существуют.

Надо задать 2 указателя p и root. v = список, состоящий из вершин текущего уровня




```
class Tree:
    def __init__(self, data, left=None, right=None):
        self.data = data
        self.left = left
        self.right = right

    def get_left(self, left):
        self.left = left

    def get_right(self, right):
        self.right = right


v1, v2, v3 = Tree(5), Tree(3), Tree(7)
v4, v5, v6 = Tree(2), Tree(6), Tree(8)
v1.get_left(v2)
v1.get_right(v3)
v2.get_left(v4)
v3.get_left(v5)
v3.get_right(v6)

p = v1
v = [v1]
while v:
    vn = []
```

```
    for x in v:
        print(x.data, end=" ")
        if x.left:
            vn += [x.left]
        if x.right:
            vn += [x.right]
    v = vn
```

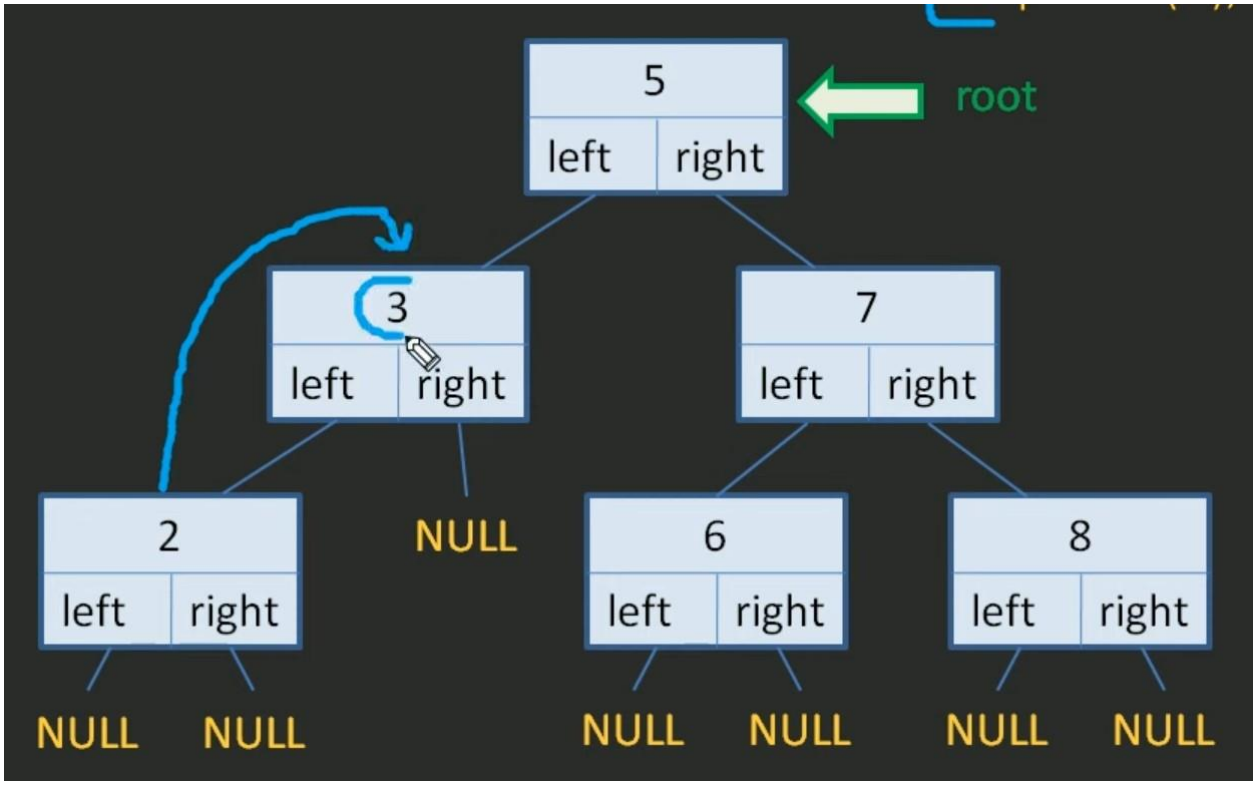
Результат 5 3 7 2 6 8

———

Вопрос 417. Опишите обход вершин бинарного дерева в глубину?

Ответ:

Доходим до листовой вершины и обрабатываем её, далее на уровень выше и обрабатываем её. В глубину дерево обходят с помощью рекурсивной функции.



```
class Tree:
    def __init__(self, data, left=None, right=None):
        self.data = data
        self.left = left
        self.right = right

    def get_left(self, left):
        self.left = left

    def get_right(self, right):
        self.right = right

    def show_tree(self, node):
        if node is None:
            return

        self.show_tree(node.left)
        print(node.data, end=" ")
        self.show_tree(node.right)


v1, v2, v3 = Tree(5), Tree(3), Tree(7)
v4, v5, v6 = Tree(2), Tree(6), Tree(8)
v1.get_left(v2)
v1.get_right(v3)
v2.get_left(v4)
v3.get_left(v5)
v3.get_right(v6)
v1.show_tree(v1)
```

Результат: 2 3 5 6 7 8

———

Вопрос 418. Что такое ассоциативные массивы?
Ответ:
Ассоциативные массивы - это массивы, доступ к значениям которых осуществляется по ключам. А в обычных массивах - по индексу.

———

Вопрос 419. Что используется, чтобы представить ассоциативные массивы в памяти компьютера?
Ответ:
Чтобы представлять ассоциативные массивы в памяти компьютера используются префиксные деревья.
В каждом элементе такого дерева хранится ровно один символ ключа
- префиксное дерево еще называют нагруженным или Trie деревом
- в качестве ключей префиксных деревьев, как правило, используется байтовая последовательность символов
- вершины префиксного дерева всегда хранят только один символ ключа
- по расходу памяти префиксное дерево вполне может оказаться эффективнее других структур данных
- для любой начальной последовательности символов ключа можно легко получить наборы ключей (слов), которые соответствуют этому префиксу
- с помощью префиксного дерева удобно формировать ассоциативные массивы

———

Вопрос 420. Что такое хеш-таблица?
Ответ:
Хэш-таблица — это структура данных, реализующая интерфейс ассоциативного массива, а именно, она позволяет хранить пары (ключ, значение) и выполнять три операции: операцию добавления новой пары, операцию удаления и операцию поиска пары по ключу.

———

Вопрос 421. Какие Big O для операций в хеш-таблицах?
Ответ:

	Хэш-таблица		Динамические массивы	Связные списки
	В среднем	В худшем		
Вставка	O(1)	O(n)	O(n)	O(1)
Чтение	O(1)	O(n)	O(1)	O(n)
Удаление	O(1)	O(n)	O(n)	O(1)

———

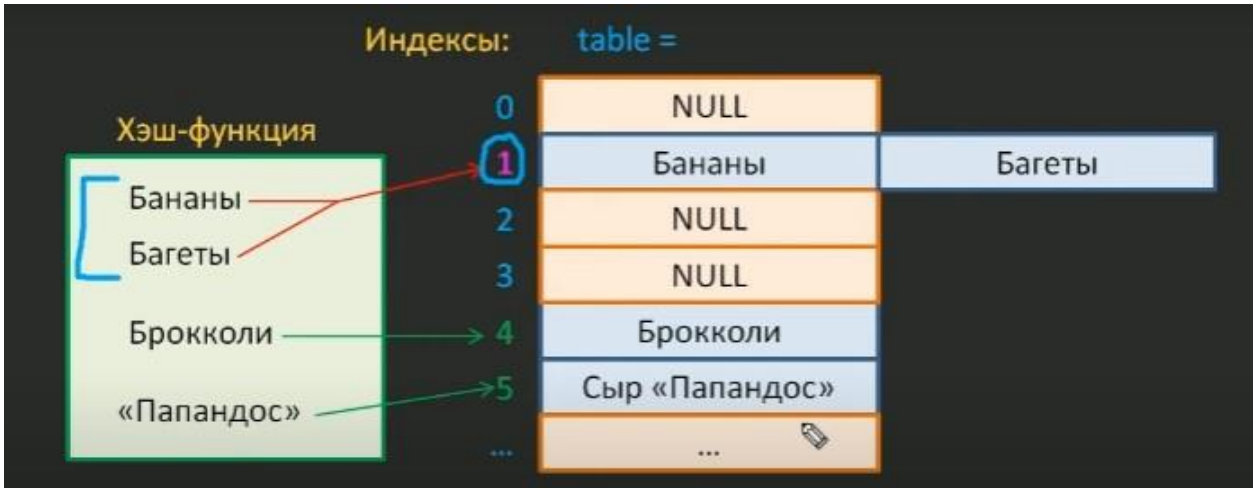
Вопрос 422. Что такое хеш-функция и хеширование?
Ответ:
Алгоритм преобразования некоторой строки в индекс массива получил название хэш-функции, а сам процесс преобразование - хеширование.

———

Вопрос 423. Что такое хеш-функция?
Ответ:
Хэш-функция преобразовывает ключи в индексы ячейки, где хранятся данные для соответствующих ключей

———

Вопрос 424. Что такое коллизии в хеш-таблицах?
Ответ:
Если число ключей стремится к бесконечности, хэш-функция назначает на один и тот же индекс 2 ключа



Чтобы разрешить коллизии существует:
- метод цепочек (в ключе создается двусвязный список, и по одному и тому же индексу мы храним несколько ключей)
- метод открытой адресации

———

Вопрос 425. Как происходит поиск ключей в хеш-таблице?
Ответ: Через хеш-функцию вначале мы получаем индекс в хеш-таблице. Внутри этого индекса может быть двусвязный список.
Далее создается временный указатель p и затем последовательно переходить по элементам двусвязного списка.

———

Вопрос 426. Какие типы данных Python используют хеш-таблицы?
В Python хэш-таблицы реализованы с множествах set() и dict().
В качестве ключей словаря могут быть все хешируемые объекты, к которым относятся все неизменяемые типы данных

- [ ] значения множества set сохраняются как значения в хэш-таблице с автоматически сгенерированными ключами
- [x] ключи словаря dict сохраняются как ключи хэш-таблицы
- [ ] ключи словаря dict сохраняются как значения в хэш-таблице в формате пары "ключ-значение"
- [x] словари dict в языке Python реализованы на базе хэш-таблиц
- [x] значениями множества set являются ключи хэш-таблицы
- [x] множества set в языке Python реализованы на базе хэш-таблиц