







# Workshop

Урок 4



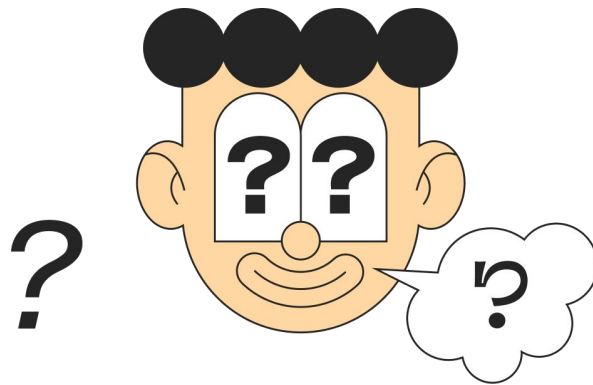


## Регламент

-  Получаем задание
-  Ученики разбиваются на группы в сессионные комнаты зум
-  Один из учеников в группе расшаривает экран
-  Выполняем задание на определенное время
-  Проверяем правильность выполнения
-  Переходим к новому заданию



Все ли было понятно в лекции?





# Задания





## Задание 1 (тайминг 10 минут)

1. Начинаем реализацию хэш-таблицы с подготовки структуры и необходимых классов.
2. Давайте напишем реализацию односвязного списка, в котором мы и будем хранить пары ключ-значение.
3. Стоит обратить внимание, что можно использовать как дженерики, для обобщения возможных типов ключей и значений, так и заранее определить для себя конкретные типы, которые будут использоваться в качестве ключа и значения. Оба подхода допустимы для реализации.





## Задание 2 (тайминг 15 минут)

1. Добавляем массив связанных списков с фиксированным размером (массив бакетов), либо передаваемым в конструкторе.
2. Хэш-таблица оперирует индексами, потому массив будет идеальным вариантом для представления бакетов.
3. Также реализуем метод вычисления индекса на основании хэш-кода ключа.





## Задание 3 (тайминг 10 минут)

1. Реализуем метод поиска данных по ключу в хэш-таблице.
2. Теперь, когда у нас есть базовая структура нашей хэш-таблицы, можно написать алгоритм поиска элементов, включающий в себя поиск нужного бакета и поиск по бакету.





## Задание 4 (тайминг 15 минут)

1. Необходимо реализовать методы добавления элементов в связный список, если там еще нет пары с аналогичным ключом и удаления элемента с аналогичным ключом из списка.
2. Все значения ключей в хэш-таблице уникальны, а значит и в каждом из связных список это правило будет также выполняться.







## Задание 5 (тайминг 5 минут)

1. Реализуем алгоритм добавления и удаления элементов из хэш-таблицы по ключу.





## Задание 6 (тайминг 15 минут)

1. Добавляем информацию о размере хэш-таблицы, а также алгоритм увеличения количества бакетов при достижении количества элементов до определенного размера относительно количества бакетов (load factor).
2. Чтобы хэш-таблица сохраняла сложность поиска близкой к  $O(1)$ , нам необходимо контролировать количество бакетов, чтобы в них не скапливалось слишком много элементов, которые способны увеличить длительность операции поиска и добавления.
3. В Java load factor для хэш-таблицы – 0.75, что значит, что при достижении количества значений 75% от общего количества бакетов – это количество необходимо увеличить. Это позволяет минимизировать шансы, что в бакетах будет больше 1-2 значений, а значит сохранит скорость поиска на уровне сложности  $O(1)$ .





## Задание 7 (тайминг 10 минут)

1. Реализуем структуру бинарного дерева.
2. Для бинарного дерева характерно наличия двух потомков, где левый меньше родителя, а правый – больше.
3. Для реализации можно использовать как и простое числовое дерево, так и обобщенный тип. Учитывая, что мы строим именно бинарное дерево, то при использовании обобщенных типов убедитесь, что значение поддерживает сравнение (интерфейс Comparable)





## Задание 8 (тайминг 15 минут)

1. Реализуем алгоритм поиска элементов по дереву (поиск в глубину).
2. Для работы с бинарным деревом необходимо как минимум организовать метод поиска.





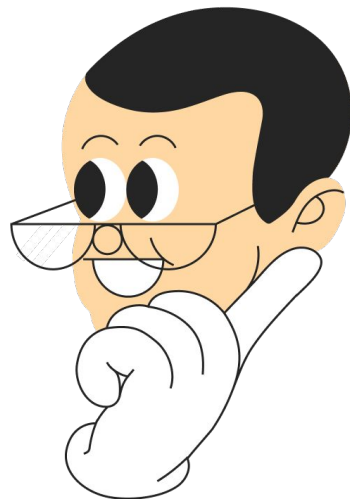
ИТОГИ





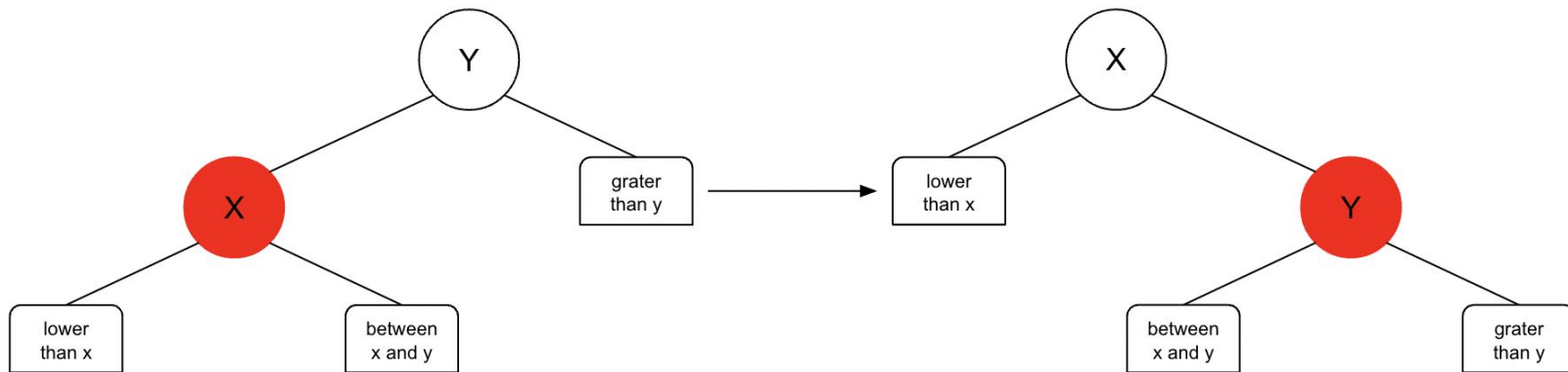
## Домашнее задание

1. Необходимо превратить собранное на семинаре дерево поиска в полноценное левостороннее красно-черное дерево. И реализовать в нем метод добавления новых элементов с балансировкой.
2. Красно-черное дерево имеет следующие критерии:
  - Каждая нода имеет цвет (красный или черный)
  - Корень дерева всегда черный
  - Новая нода всегда красная
  - Красные ноды могут быть только левым ребенком
  - У красной ноды все дети черного цвета
3. Соответственно, чтобы данные условия выполнялись, после добавления элемента в дерево необходимо произвести балансировку, благодаря которой все критерии выше станут валидными.
4. Для балансировки существует 3 операции – левый малый поворот, правый малый поворот и смена цвета.



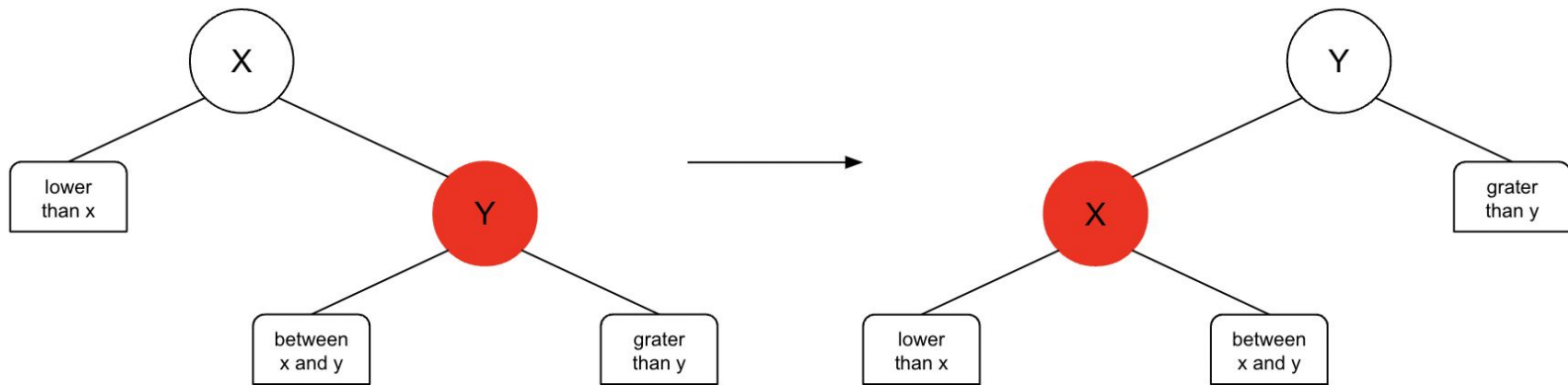


## Левосторонний поворот (малый левый поворот)





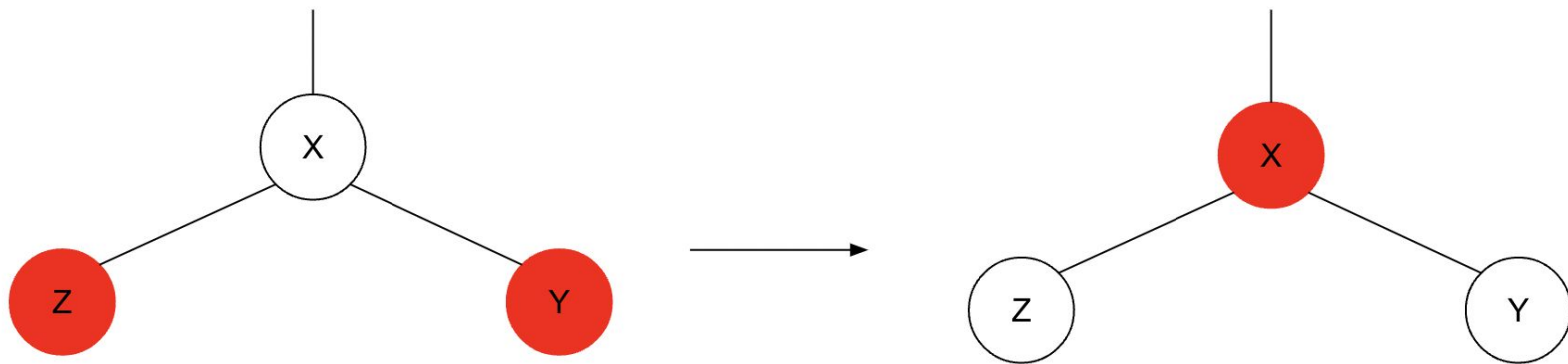
## Правосторонний поворот (малый правый поворот)







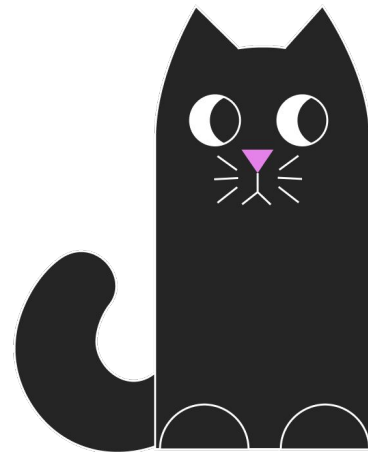
## Смена цвета





## Критерии применения этих операций следующие:

- Если правый ребенок – красный, а левый - черный, то применяем малый правый поворот
- Если левый ребенок красный и его левый ребенок тоже красный – применяем малый левый поворот
- Если оба ребенка красные – делаем смену цвета
- Если корень стал красным – просто перекрашиваем его в черный





Что было  
сложного на  
семинаре?





Напишите 3 вещи в  
комментариях, которым  
вы научились сегодня.



## Ответьте на несколько вопросов сообщением в чат




**Насколько курс был вам  
полезен?  
Оцените от 0 до 10**



**Насколько понятно и  
доступно было объяснение  
материала?  
Оцените от 0 до 10**



**Что вы бы улучшили или  
добавили?**

**Спасибо**   
**за внимание**

