Двоичное дерево

Филипп Воронов

Филипп Воронов

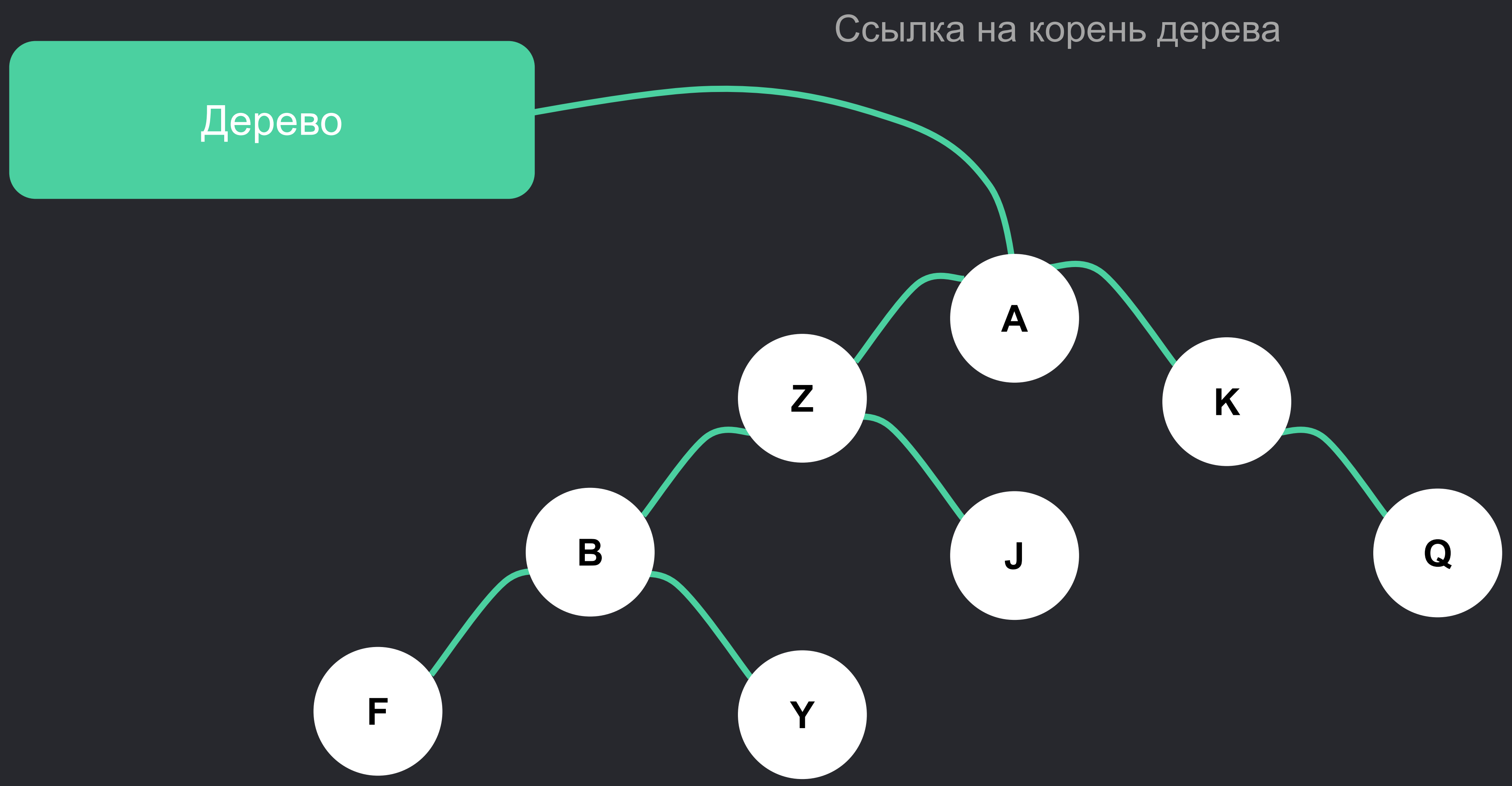
Teamlead, Поиск Mail.ru

Аккаунты в соц.сетях

@Филипп Воронов

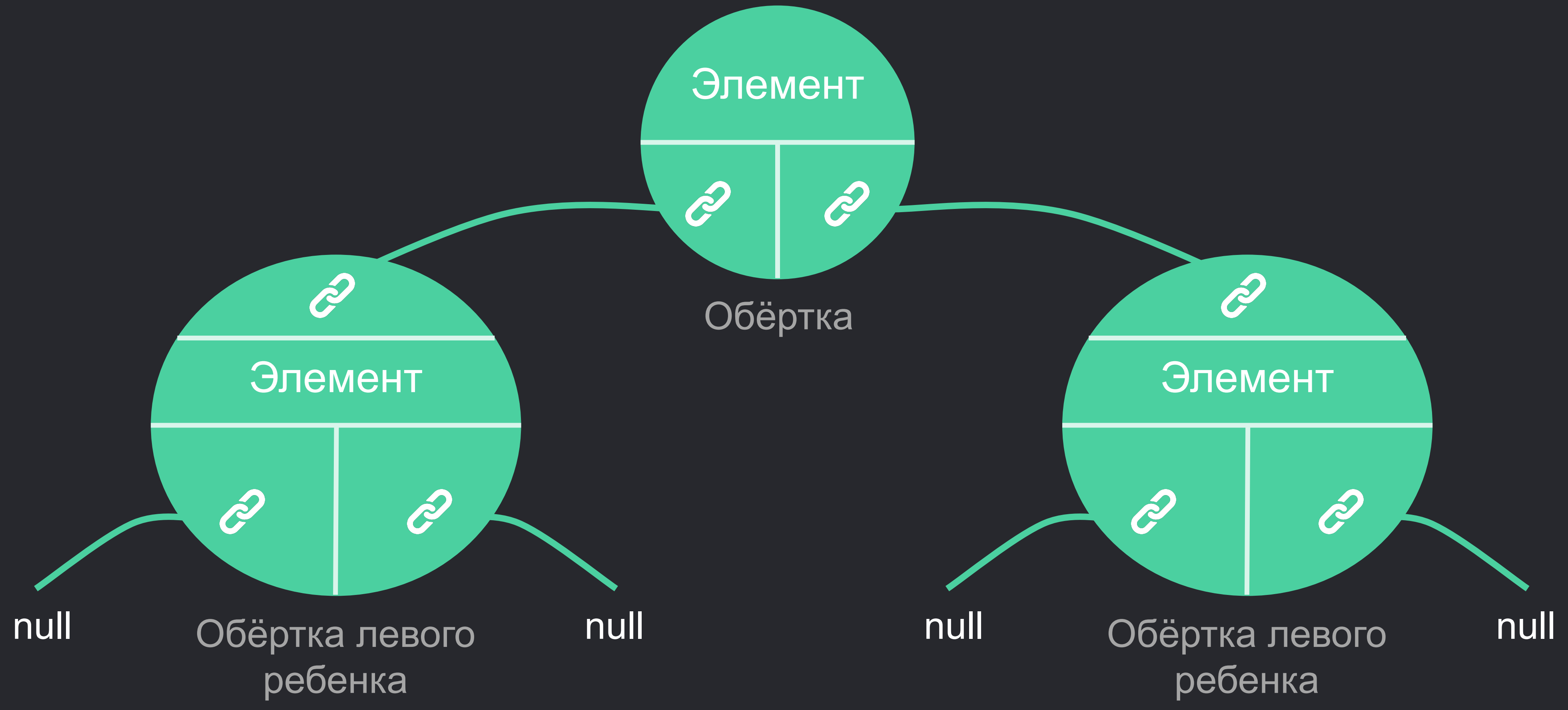
Что такое двоичное дерево?

Это структура данных, где у каждого узла есть 0–2 ребёнка.
Само дерево хранит ссылку на общего предка — на корень дерева



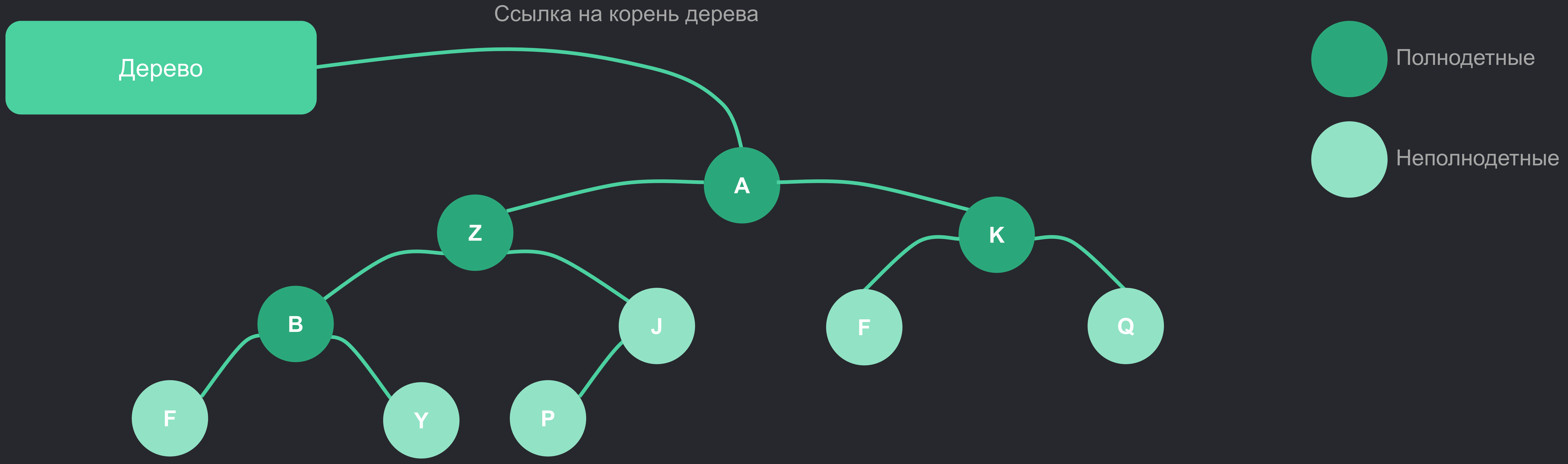
Узел двоичного дерева

Определим узел дерева аналогично узлу связного списка: обёртка, внутри которой находятся сам элемент, ссылка на левого ребёнка и ссылка на правого ребёнка



Что такое полное двоичное дерево?

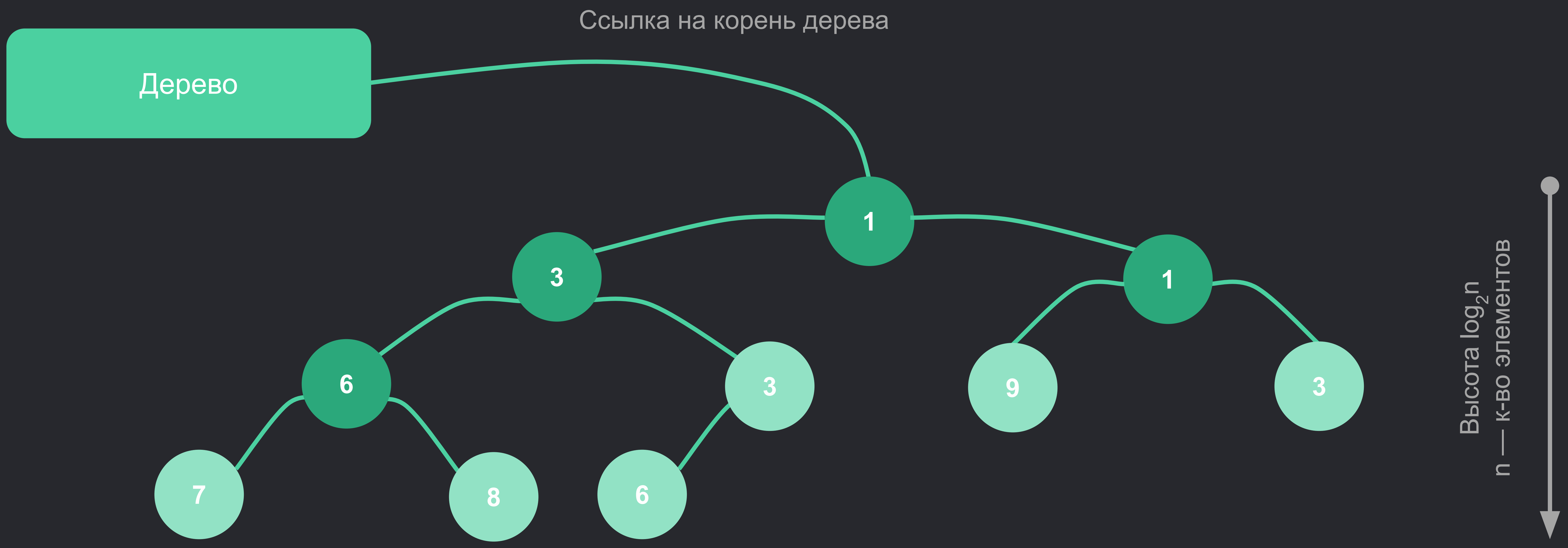
Это дерево, уровни которого мы заполняем элементами сверху вниз слева направо, причём слева идут сначала полнодетные, затем все неполнодетные



Пирамида

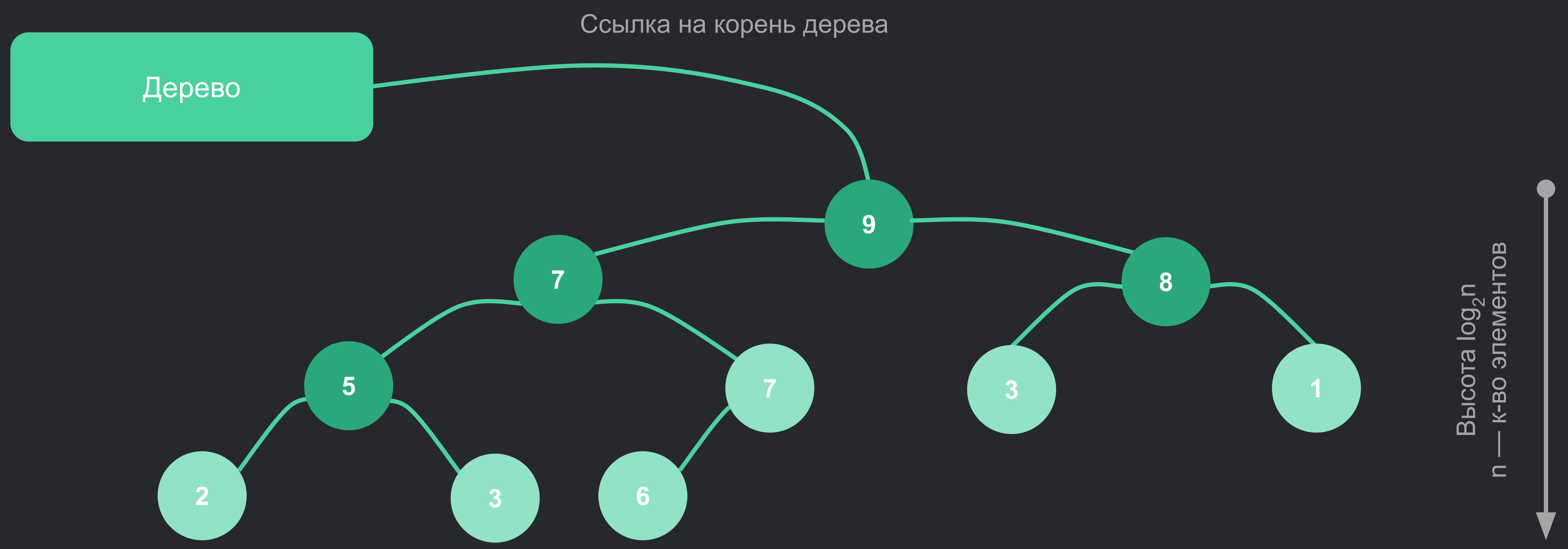
Что такое пирамида?

Пирамида (куча) — полное двоичное дерево, для которого соблюдается правило: значение в каждом родителе не больше чем у детей. Такая пирамида называется пирамидой на минимум



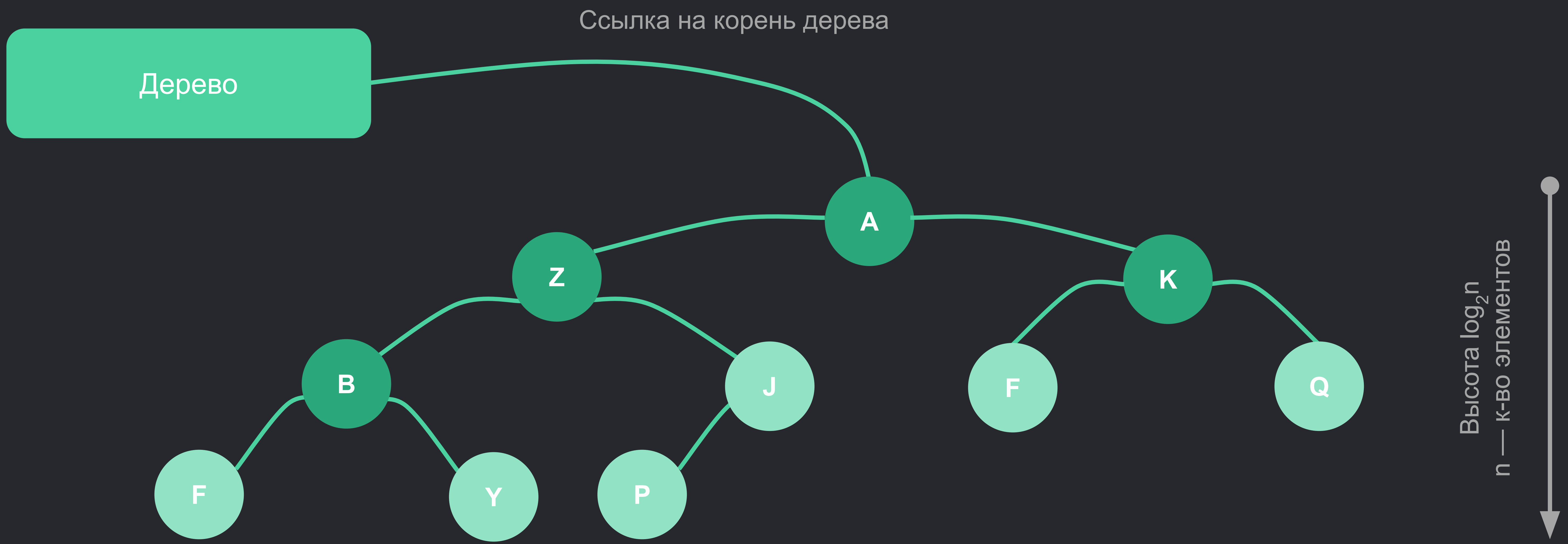
Что такое пирамида?

Аналогично определяется пирамида на максимум



Что такое пирамида?

Стоит отметить, что подойдут любые типы данных, для которых вы определите понятие сравнения. Если сравнение идёт по какой-то части элемента, эта часть называется ключом



Как реализовать пирамиду на ссылках?

Реализуем пирамиду подобно связному списку — напрямую будем держать ссылки на детей в узле

```
Node {
  e: значение,
  parent: ссылка на родителя или пусто,
  left: ссылка на левого ребёнка или
пусто,
  right: ссылка на правого ребёнка или
пусто
}

Heap {
  root: ссылка на корень пирамиды
}
```

Комментарий
Структура для узла

Как реализовать пирамиду на ссылках?

```
Node {
  e: значение,
  parent: ссылка на родителя или пусто,
  left: ссылка на левого ребёнка или
пусто,
  right: ссылка на правого ребёнка или
пусто
}

Heap {
  root: ссылка на корень пирамиды
}
```

Комментарий
Структура для самой пирамиды

Как реализовать пирамиду на ссылках?

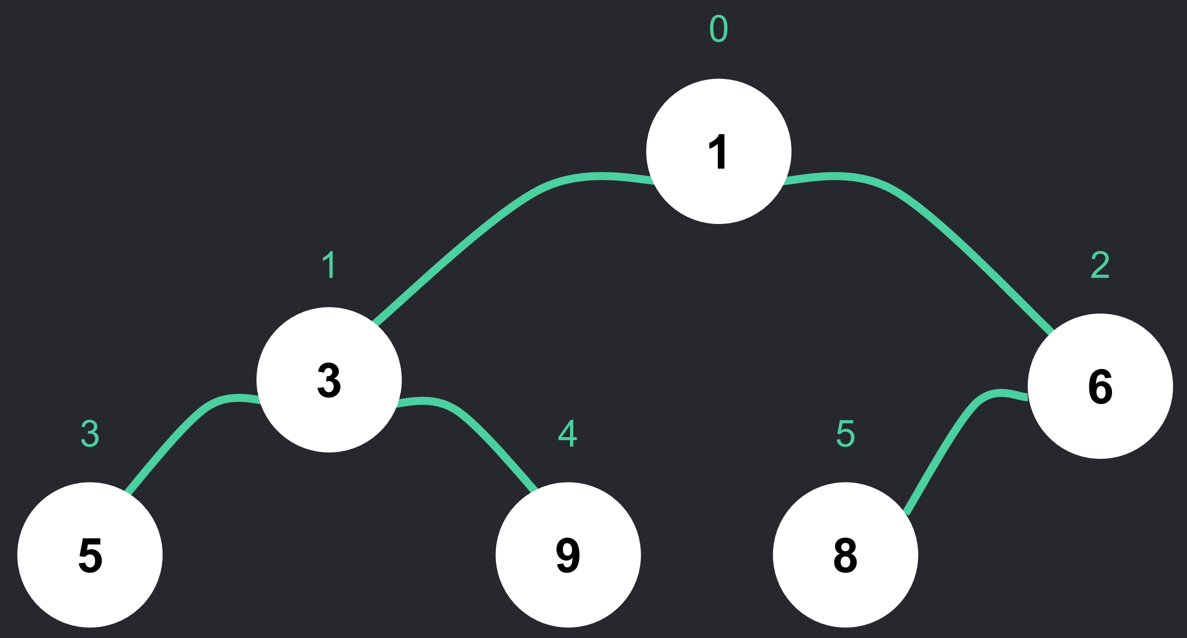
Реализуем пирамиду подобно связному списку — напрямую будем держать ссылки на детей в узле.

Но так не делают. Дело в том, что пирамида — это всегда полное двоичное дерево и те операции, которые мы над ней будем проводить, позволят нам придумать более эффективную реализацию пирамид без ухудшения асимптотик этих операций.

Как реализовать пирамиду на массиве?

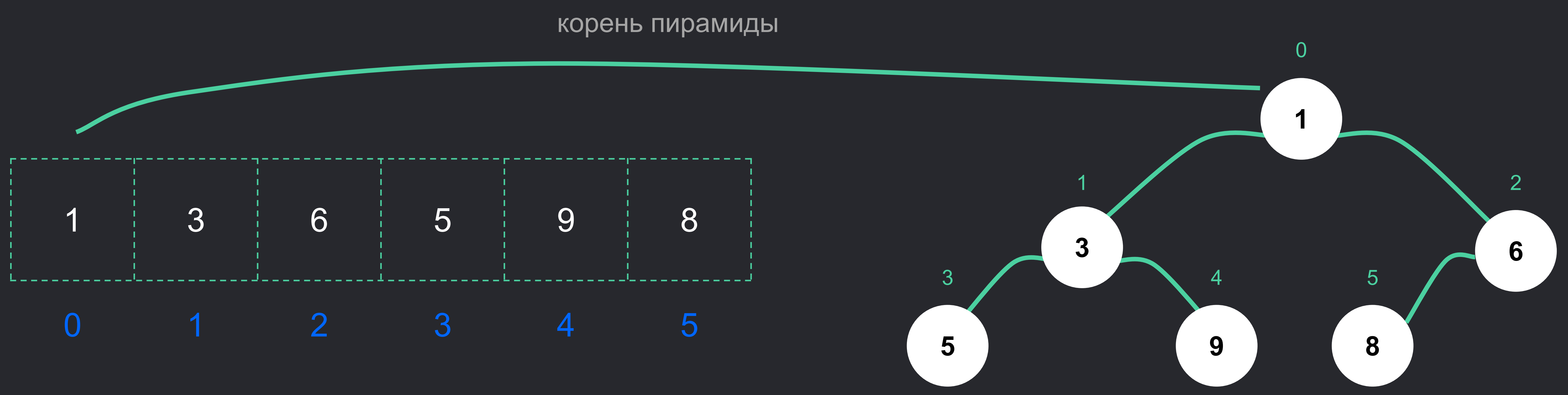
Заведём массив, в котором будем держать элементы нашей пирамиды

1	3	6	5	9	8
0	1	2	3	4	5



Как реализовать пирамиду на массиве?

Будем считать, что корень пирамиды хранится в ячейке с индексом 0



Как реализовать пирамиду на массиве?

Для любого узла пирамиды, хранящегося в ячейке под индексом i, будем считать, что его левый ребёнок хранится в 2i+1, а правый в 2i+2.

Разберём пример: необходимо узнать, где находятся левый и правый дети для узла пирамиды, хранящегося в ячейке под номером 1

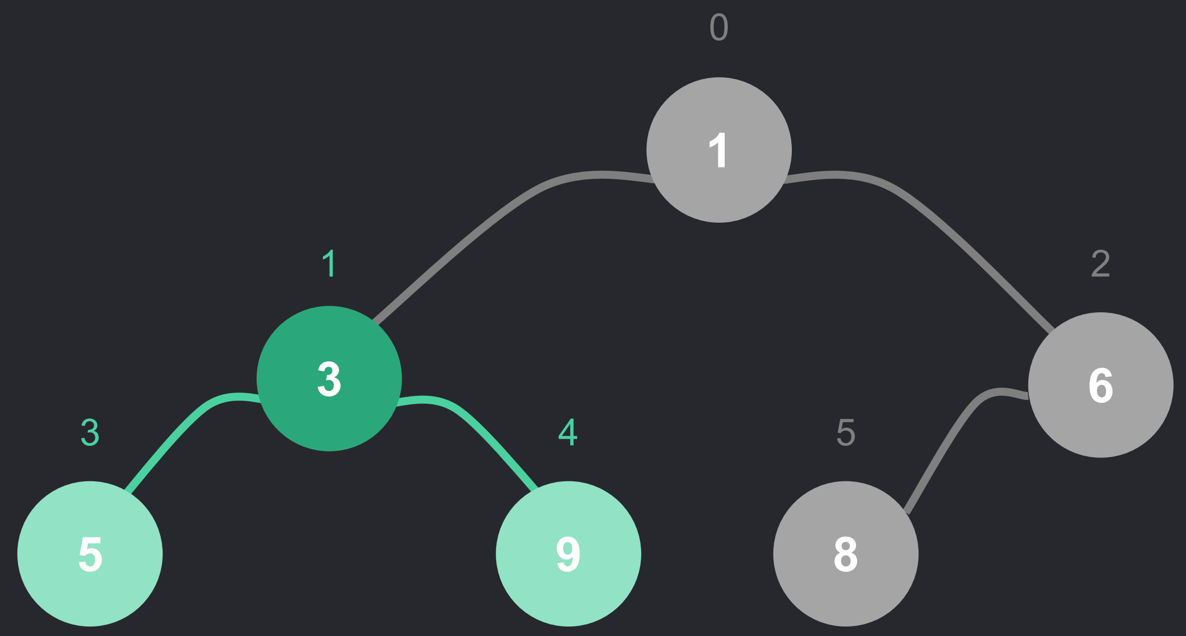
1	3	6	5	9	8
0	1	2	3	4	5

Как реализовать пирамиду на массиве?

Решение примера

- Левый ребёнок: $2 \times 1 + 1 = 3$
- Правый ребёнок: $2 \times 1 + 2 = 4$

1	3	6	5	9	8
0	1	2	3	4	5

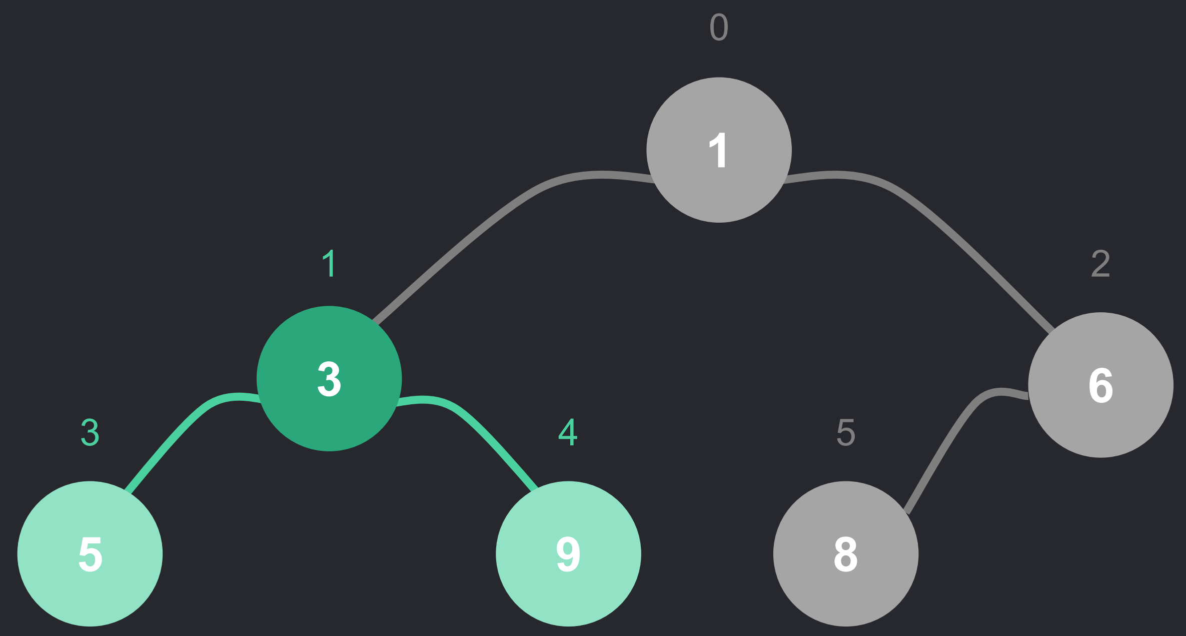


Как реализовать пирамиду на массиве?

Математически можно доказать, что конфликтов при таком заполнении не будет, как и пустых промежутков в самом массиве.

Переход к детям от родителя будет всё также за O(1)

1	3	6	5	9	8
0	1	2	3	4	5



Как реализовать пирамиду на массиве?

Пирамида на массиве:

```
Heap {
  data: [массив с элементами]

  root():
    return data[0]

  from_index(index):
    return data[index]

  left_index(parent_index):
    return 2 * parent_index + 1

  right_index(parent_index):
    return 2 * parent_index + 2

  parent_index(child_index):
    return (child_index-1) / 2
}
```

Комментарий

Массив значений из узлов пирамиды

Как реализовать пирамиду на массиве?

Пирамида на массиве:

```
Heap {
  data: [массив с элементами]

  root():
    return data[0]

  from_index(index):
    return data[index]

  left_index(parent_index):
    return 2 * parent_index + 1

  right_index(parent_index):
    return 2 * parent_index + 2

  parent_index(child_index):
    return (child_index-1) / 2
}
```

Комментарий
Корень пирамиды в 0-й ячейке

Как реализовать пирамиду на массиве?

Пирамида на массиве:

```
Heap {
  data: [массив с элементами]

  root():
    return data[0]

  from_index(index):
    return data[index]

  left_index(parent_index):
    return 2 * parent_index + 1

  right_index(parent_index):
    return 2 * parent_index + 2

  parent_index(child_index):
    return (child_index-1) / 2
}
```

Комментарий

Получение элемента по его индексу.

Сам индекс — лишь деталь реализации кучи на массиве, сама куча индексов не требует!

Как реализовать пирамиду на массиве?

Пирамида на массиве:

```
Heap {
  data: [массив с элементами]

  root():
    return data[0]

  from_index(index):
    return data[index]

  left_index(parent_index):
    return 2 * parent_index + 1

  right_index(parent_index):
    return 2 * parent_index + 2

  parent_index(child_index):
    return (child_index-1) / 2
}
```

Комментарий
Левый ребёнок

Как реализовать пирамиду на массиве?

Пирамида на массиве:

```
Heap {
  data: [массив с элементами]

  root():
    return data[0]

  from_index(index):
    return data[index]

  left_index(parent_index):
    return 2 * parent_index + 1

  right_index(parent_index):
    return 2 * parent_index + 2

  parent_index(child_index):
    return (child_index-1) / 2
}
```

Комментарий
Правый ребёнок

Как реализовать пирамиду на массиве?

Пирамида на массиве:

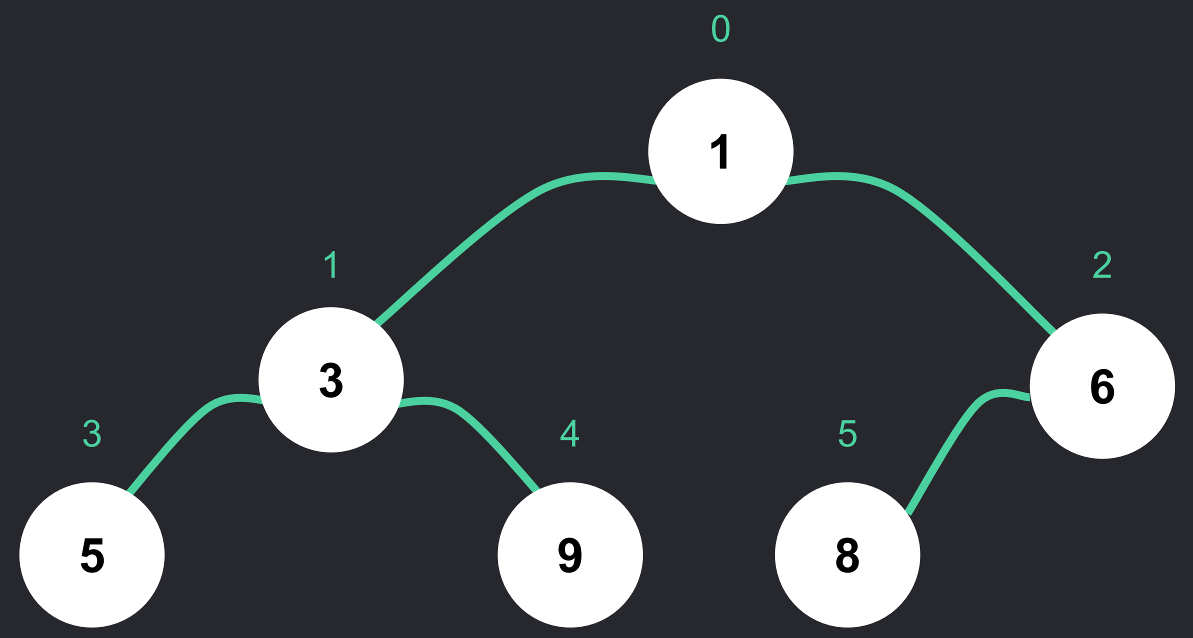
```
Heap {
  data: [массив с элементами]

  root():
    return data[0]

  from_index(index):
    return data[index]

  left_index(parent_index):
    return 2 * parent_index + 1

  right_index(parent_index):
    return 2 * parent_index + 2

  parent_index(child_index):
    return (child_index-1) / 2
}
```

Комментарий

Номер родительской ячейки получается обратным действием к получению номера дочернего узла. Там умножали на 2, тут делим на 2

Как реализовать пирамиду на массиве?

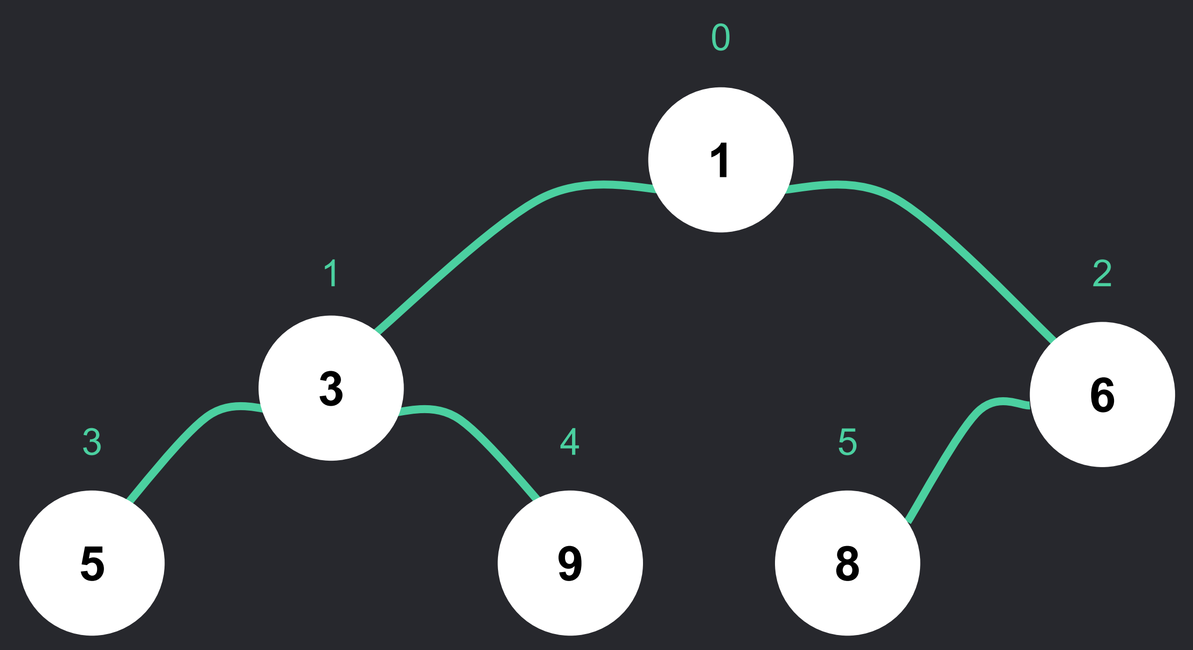
Итог: мы более компактно храним нашу пирамиду и работа на массиве зачастую быстрее, чем работа на ссылках



Добавление нового элемента в пирамиду

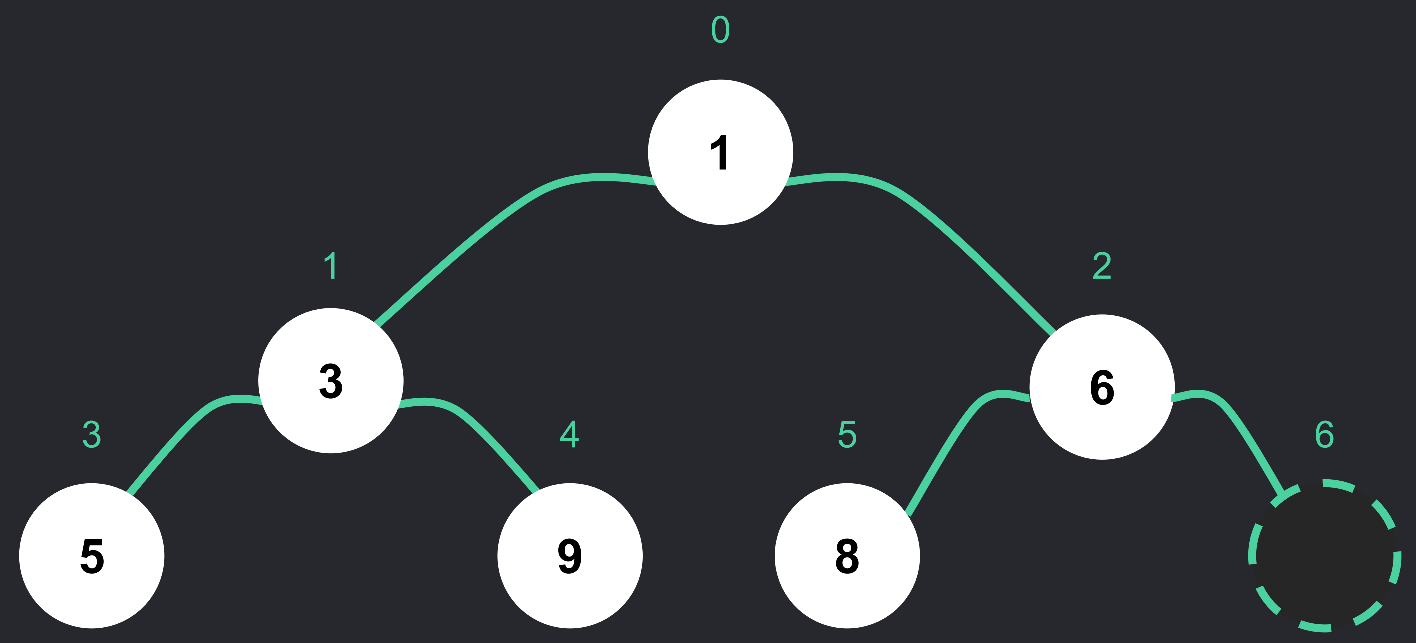
Как добавить элемент в пирамиду?

Представим, что мы имеем двоичное дерево пирамиды и мы хотим добавить туда элемент -2



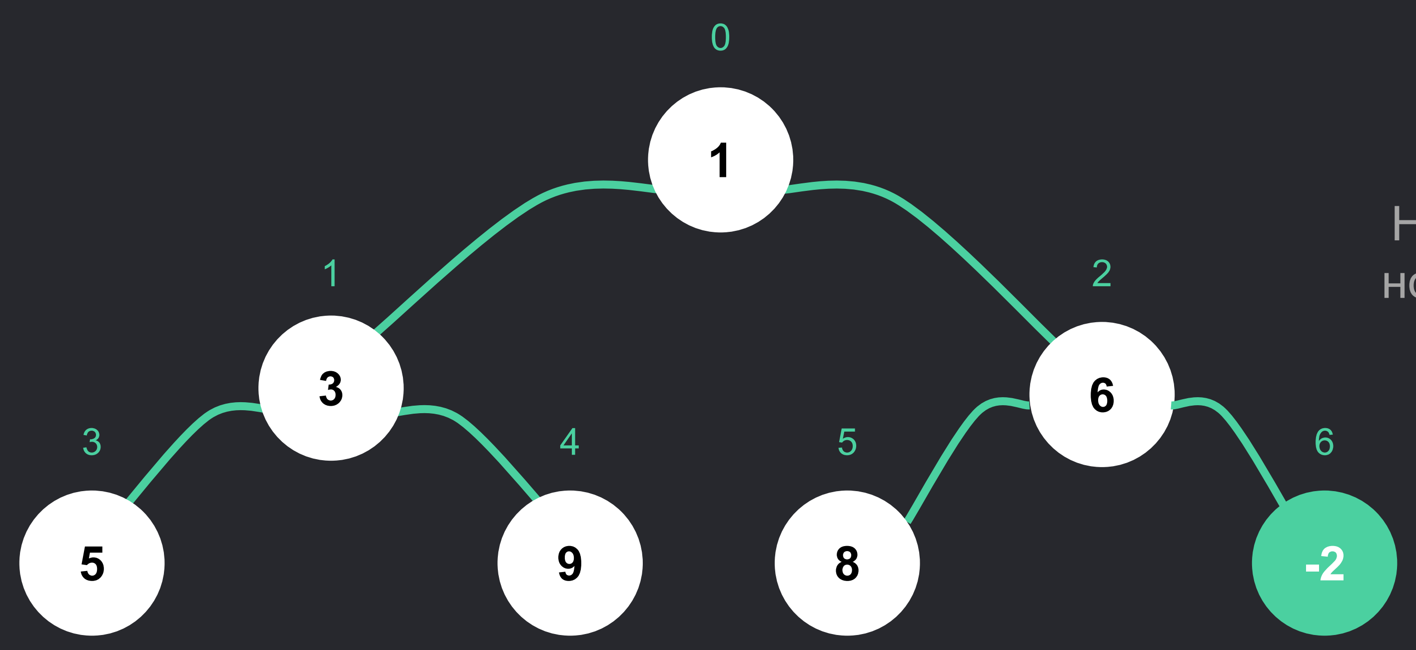
Как добавить элемент в пирамиду?

Сначала создадим следующий новый узел в нижнем ряду



Как добавить элемент в пирамиду?

Теперь заполним новый узел добавляемым значением -2



Нарушилось условие пирамиды:
новый элемент меньше родителя

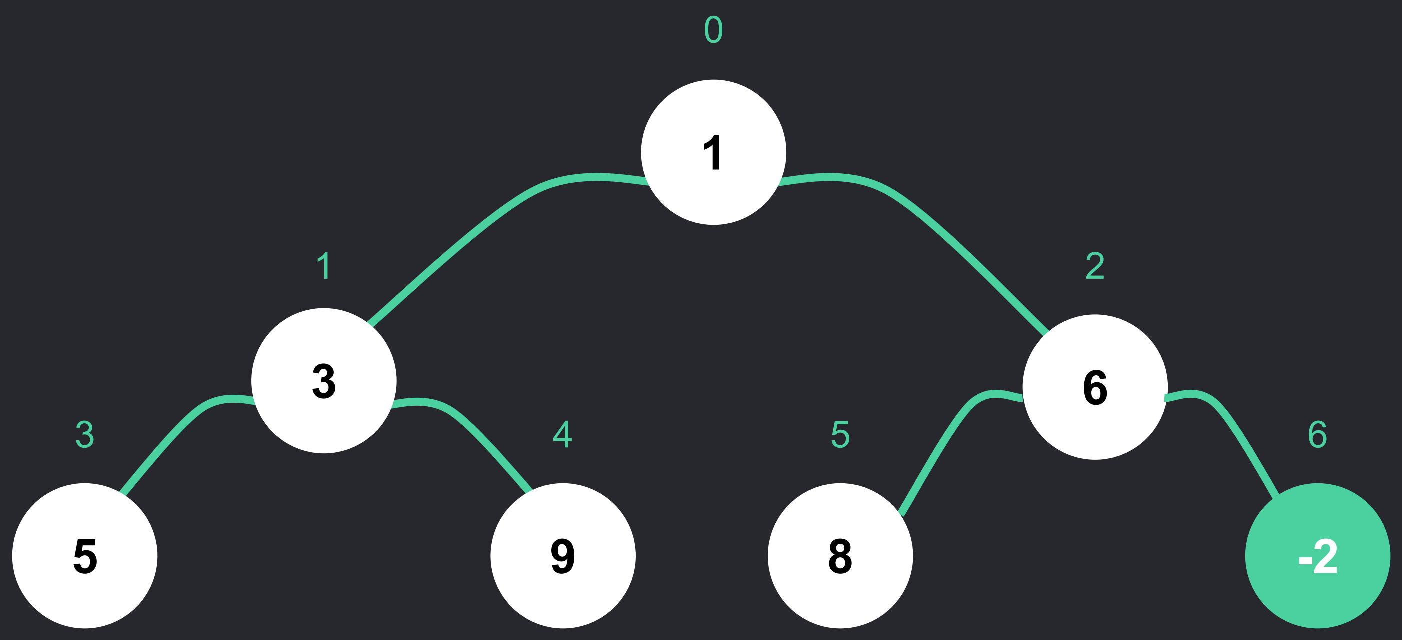
Как добавить элемент в пирамиду?

Если условие пирамиды нарушено, то запустим в этом месте операцию «всплывания» элемента

Операция «всплывание» элемента

Сравниваем значение нового узла со значением родителя, если родитель больше — повторяем операцию для позиции родителя.

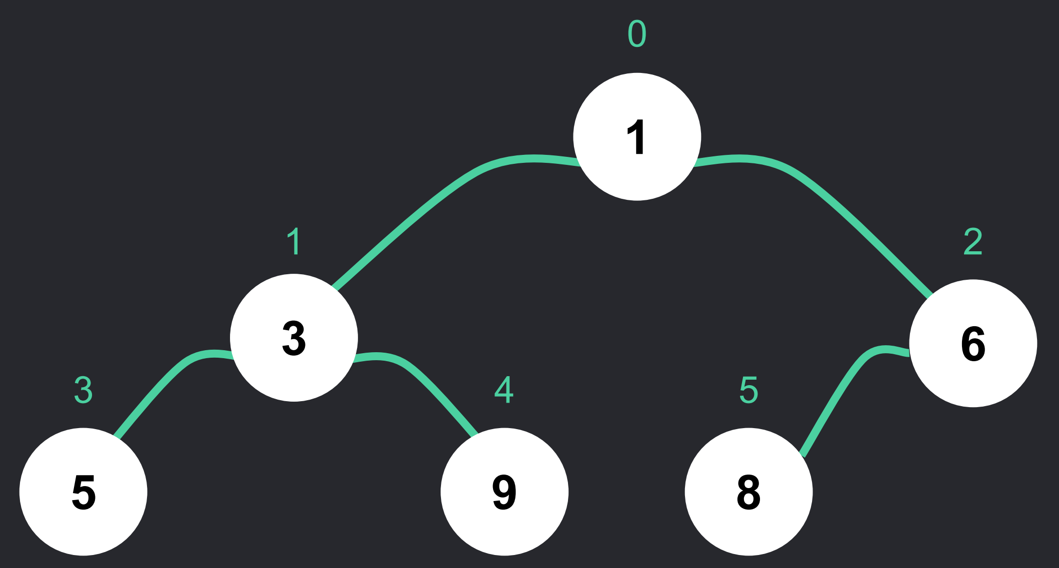
Время операции — O(высоты), а т. к. высота — это логарифм n, то $O(\log_2 n)$.



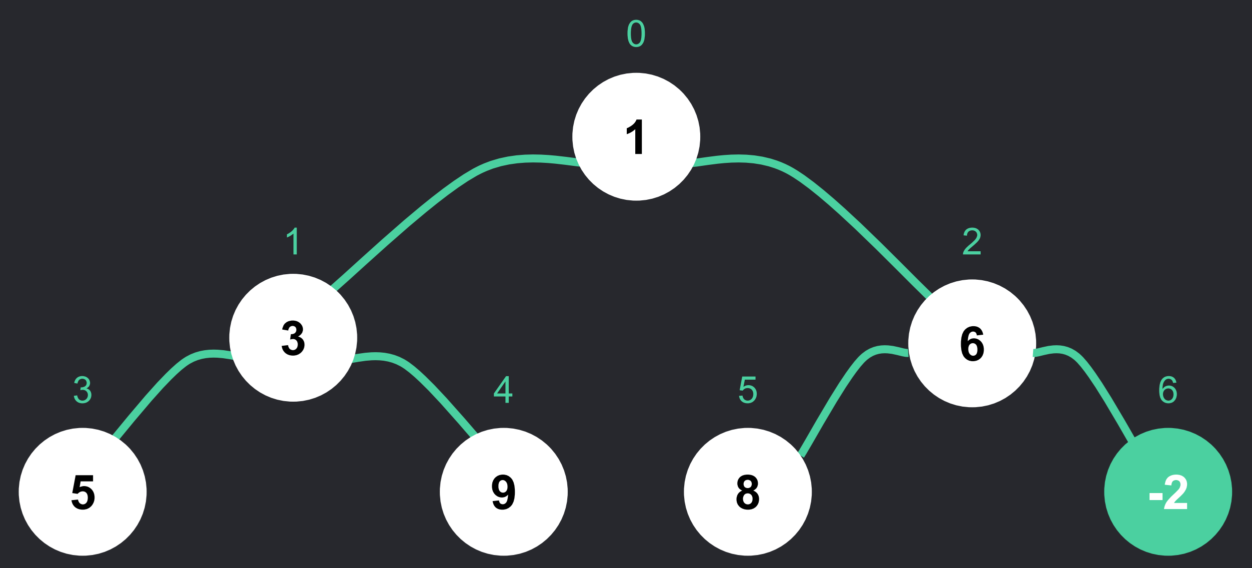
Как добавить элемент в пирамиду?

Новый узел будет добавляться в несколько шагов:

0 Начальное состояние



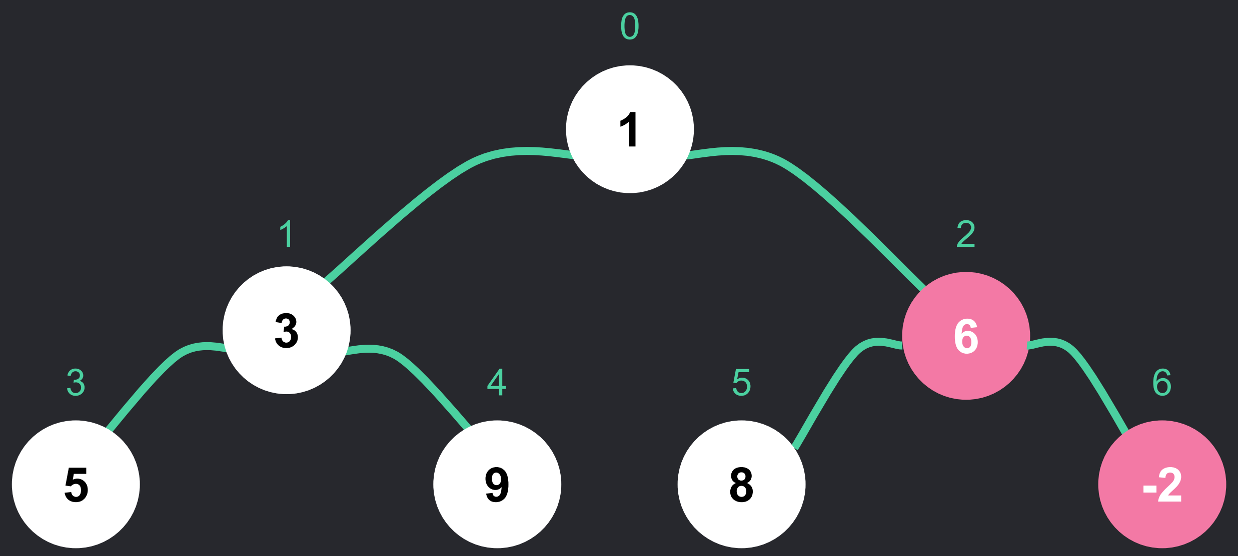
1 Создаём узел для добавляемого элемента



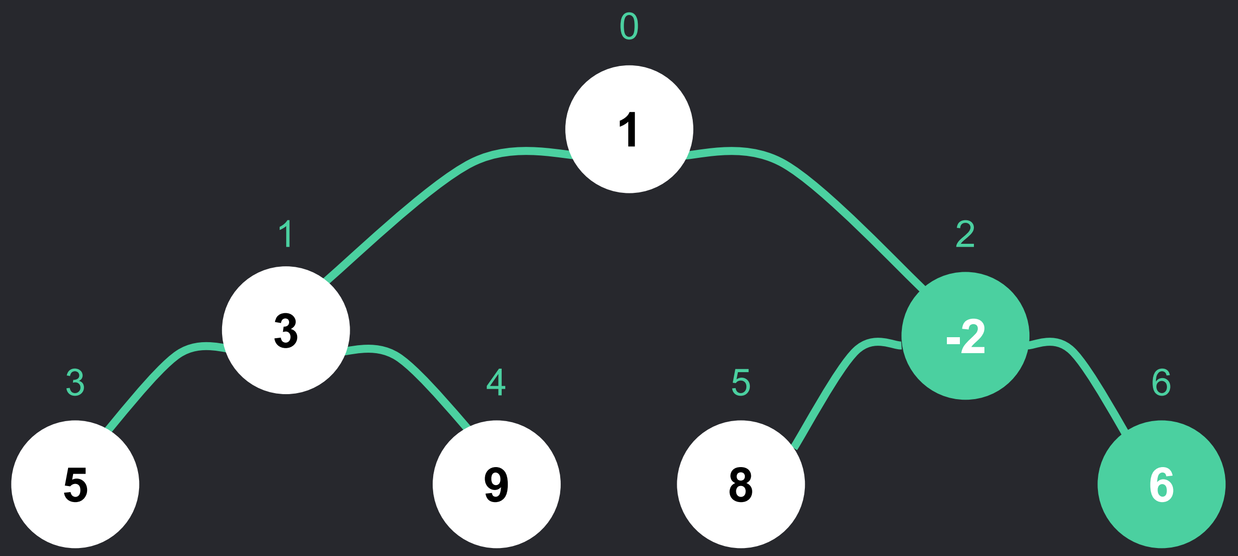
Как добавить элемент в пирамиду?

Новый узел будет добавляться в несколько шагов:

2 Проверяем пирамидальность



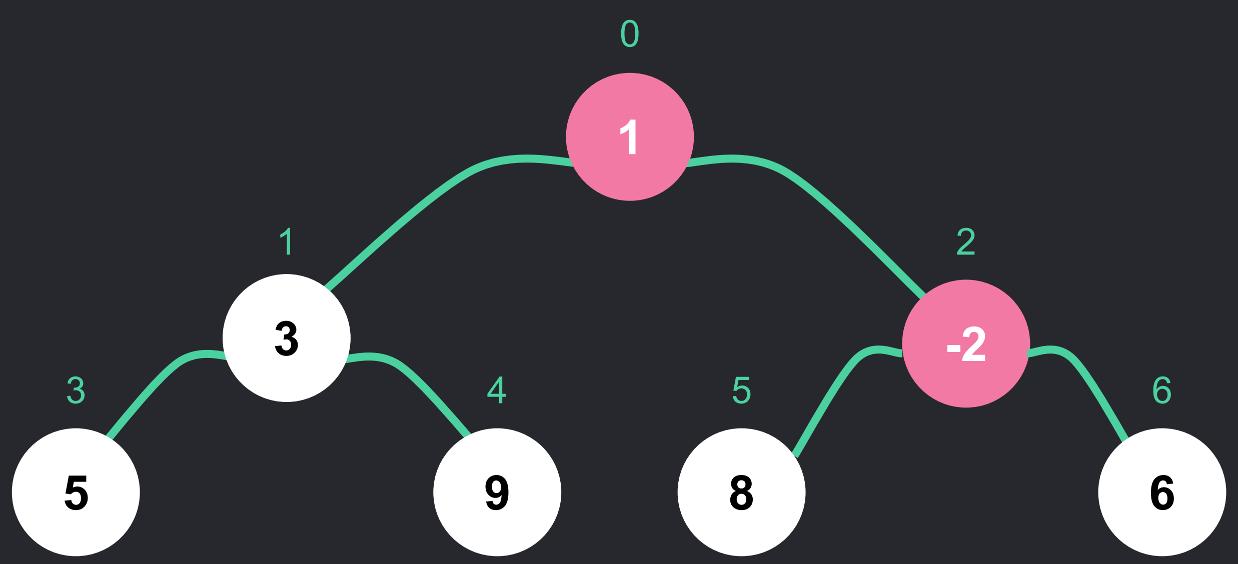
3 Всплываем



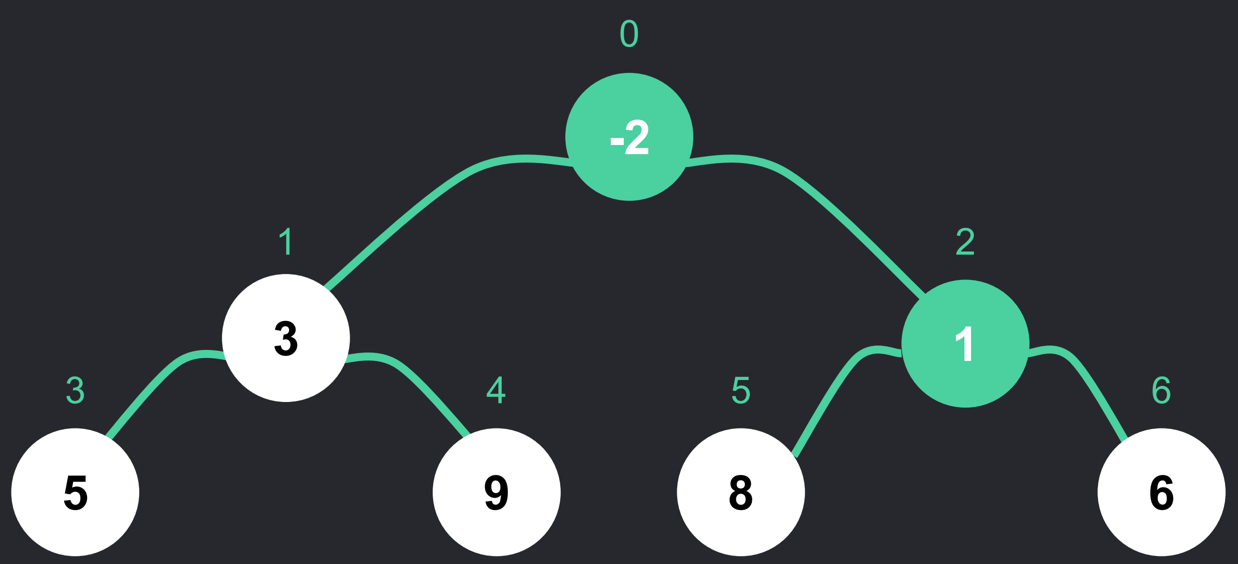
Как добавить элемент в пирамиду?

Новый узел будет добавляться в несколько шагов:

4 Проверяем пирамидальность



5 Всплываем



Как добавить элемент в пирамиду?

Псевдокод

```
Heap {
  ...

  try_up(index):
    if index = 0: выход
    parent_index = parent_index(index)
    parent = data[parent_index]
    if parent > data[index]
      swap data[index] data[parent_index]
      try_up(parent_index)

  add(e):
    добавить в конец data значение e
    try_up(длина(data) — 1)
}
```

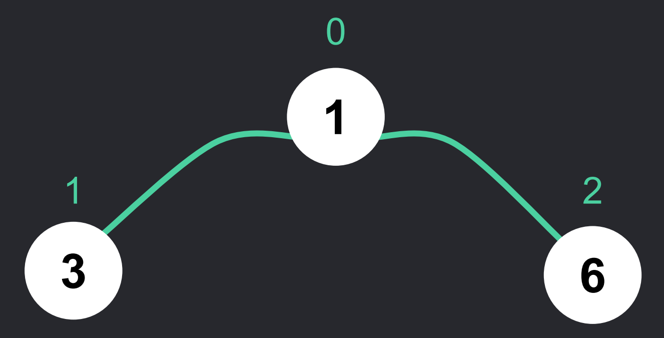
Комментарий
Вспомогательная функция всплытия элемента на один уровень выше нарушающего пирамидальность

Как добавить элемент в пирамиду?

Псевдокод

```
Heap {
  ...

  try_up(index):
    if index = 0: выход
    parent_index = parent_index(index)
    parent = data[parent_index]
    if parent > data[index]
      swap data[index] data[parent_index]
      try_up(parent_index)

  add(e):
    добавить в конец data значение e
    try_up(длина(data) — 1)
}
```

Комментарий
Мы в корне, всплывать некуда — это самая верхняя позиция



Как добавить элемент в пирамиду?

Псевдокод

```
Heap {
  ...

  try_up(index):
    if index = 0: выход
    parent_index = parent_index(index)
    parent = data[parent_index]
    if parent > data[index]
      swap data[index] data[parent_index]
      try_up(parent_index)

  add(e):
    добавить в конец data значение e
    try_up(длина(data) — 1)
}
```

Комментарий

Находим родителя этого элемента

Как добавить элемент в пирамиду?

Псевдокод

```
Heap {
  ...

  try_up(index):
    if index = 0: выход
    parent_index = parent_index(index)
    parent = data[parent_index]
    if parent > data[index]
      swap data[index] data[parent_index]
      try_up(parent_index)

  add(e):
    добавить в конец data значение e
    try_up(длина(data) — 1)
}
```

Комментарий

Проверяем, нарушается ли свойство пирамиды

Как добавить элемент в пирамиду?

Псевдокод

```
Heap {
  ...

  try_up(index):
    if index = 0: выход
    parent_index = parent_index(index)
    parent = data[parent_index]
    if parent > data[index]
      swap data[index] data[parent_index]
      try_up(parent_index)

  add(e):
    добавить в конец data значение e
    try_up(длина(data) — 1)
}
```

Комментарий
Меняем местами этот элемент с его родителем

Как добавить элемент в пирамиду?

Псевдокод

```
Heap {
  ...

  try_up(index):
    if index = 0: выход
    parent_index = parent_index(index)
    parent = data[parent_index]
    if parent > data[index]
      swap data[index] data[parent_index]
      try_up(parent_index)

  add(e):
    добавить в конец data значение e
    try_up(длина(data) — 1)
}
```

Комментарий

Пытаемся всплыть от новой позиции добавляемого элемента

Как добавить элемент в пирамиду?

Псевдокод

```
Heap {
  ...

  try_up(index):
    if index = 0: выход
    parent_index = parent_index(index)
    parent = data[parent_index]
    if parent > data[index]
      swap data[index] data[parent_index]
      try_up(parent_index)

  add(e):
    добавить в конец data значение e
    try_up(длина(data) — 1)
}
```

Комментарий
Добавляем элемент в пирамиду

Как добавить элемент в пирамиду?

Псевдокод

```
Heap {
  ...

  try_up(index):
    if index = 0: выход
    parent_index = parent_index(index)
    parent = data[parent_index]
    if parent > data[index]
      swap data[index] data[parent_index]
      try_up(parent_index)

  add(e):
    добавить в конец data значение e
    try_up(длина(data) — 1)
}
```

Комментарий

Добавляем элемент в конец массива

Как добавить элемент в пирамиду?

Псевдокод

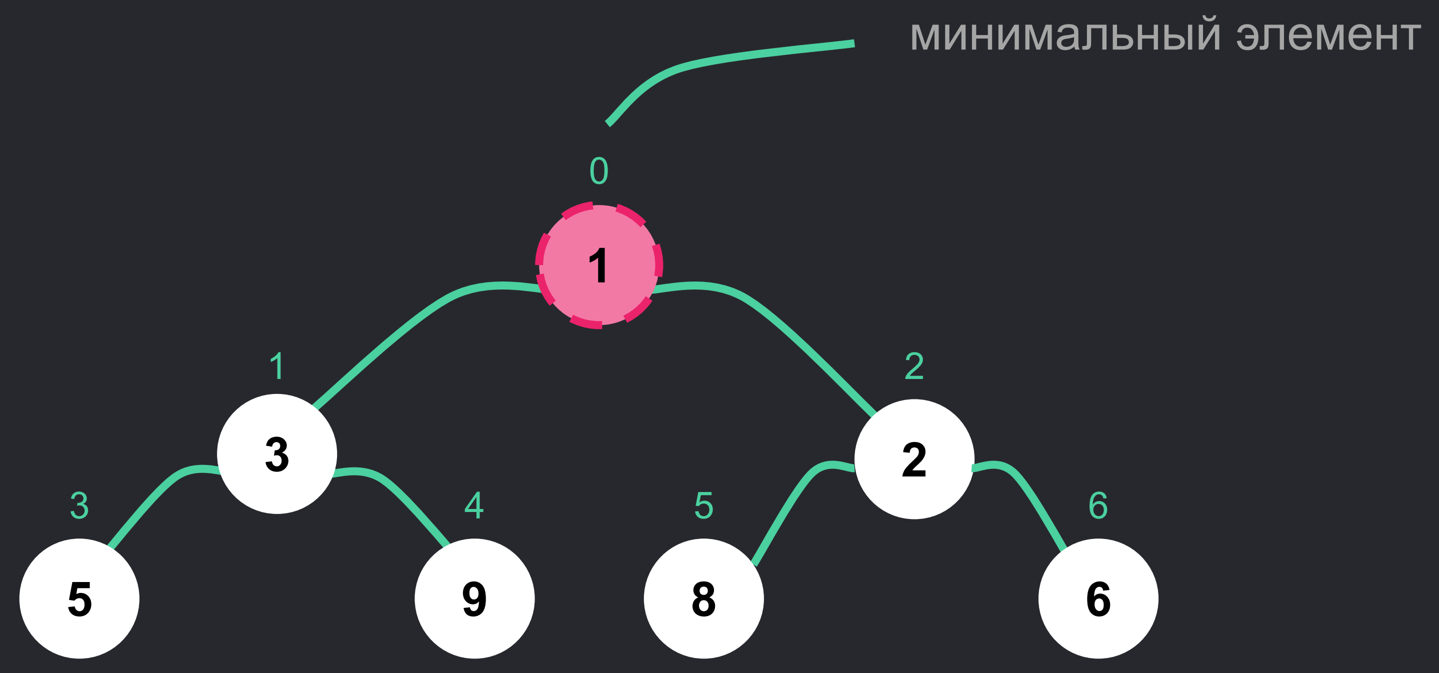
```
Heap {
  ...

  try_up(index):
    if index = 0: выход
    parent_index = parent_index(index)
    parent = data[parent_index]
    if parent > data[index]
      swap data[index] data[parent_index]
      try_up(parent_index)

  add(e):
    добавить в конец data значение e
    try_up(длина(data) — 1)
}
```

Комментарий
Детей у него нет. Он может нарушить свойство пирамиды только если слишком маленький по значению. Попробуем всплыть

Удаление минимума в пирамиде

Как удалить минимум в пирамиде?

Минимальный элемент в пирамиде на минимум лежит в корне, так что время его поиска — O(1). А вот с извлечением его из пирамиды — уже трудность.



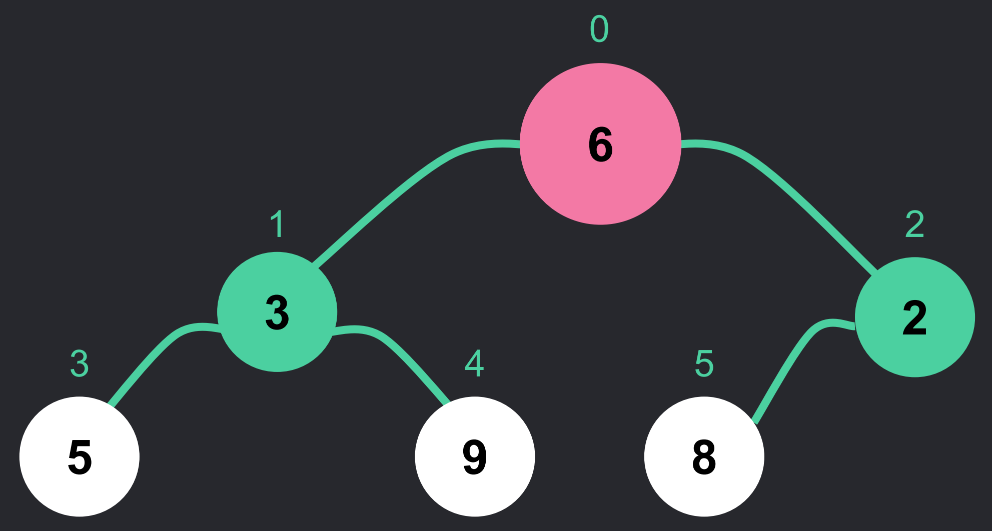
Как удалить минимум в пирамиде?

Реализация:

- вынем из корня элемент
- заполним корень значением, лежащим в самом последнем узле пирамиды
- удалим последний узел

Как удалить минимум в пирамиде?

Однако может нарушиться свойство пирамиды, если в корне будет значение большее, чем у его детей



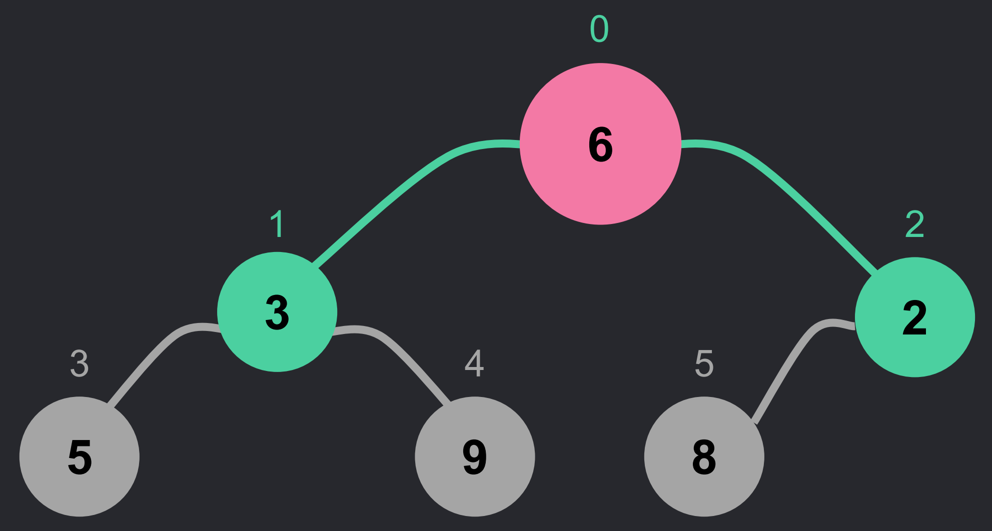
Как удалить минимум в пирамиде?

Мы видим, что свойство пирамиды нарушено.
Аналогично операции всплывания, сделаем операцию просеивания относительно нового корня пирамиды, пока свойство пирамидности не восстановится

Операция «просеивания»

Сравниванием значение родителя и минимального ребенка, если родитель больше — меняем местами.

Время операции — O(высоты), а т. к. высота — это логарифм n, то $O(\log_2 n)$.



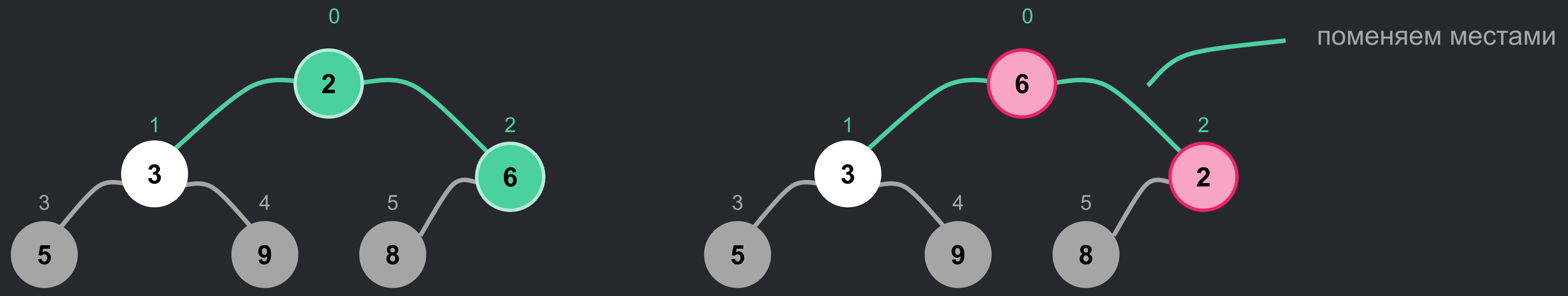
Как удалить минимум в пирамиде?

В нашем примере достаточно одной операции «просеивания».

Операция «просеивания»

Сравниванием значение родителя и минимального ребенка, если родитель больше — меняем местами.

Время операции — O(высоты), а т. к. высота — это логарифм n, то $O(\log_2 n)$.



Как удалить минимум в пирамиде?

Рассмотрим псевдокод

```
Heap {
  ...

  sift_down(index):
    if детей у index нет: выход
    min_child = минимальный ребёнок index
    if data[index] > data[min_child]
      swap data[index] data[min_child]
      sift_down(min_child)

  extract_min(e):
    ans = data[0]
    data[0] = data[длина(data)-1]
    удалить у data 1 элемент с конца
    sift_down(0)
    return ans
}
```

Комментарий

Вспомогательная операция просеивания вниз

Как удалить минимум в пирамиде?

Рассмотрим псевдокод

```
Heap {
  ...

  sift_down(index):
    if детей у index нет: выход
    min_child = минимальный ребёнок index
    if data[index] > data[min_child]
      swap data[index] data[min_child]
      sift_down(min_child)

  extract_min(e):
    ans = data[0]
    data[0] = data[длина(data)-1]
    удалить у data 1 элемент с конца
    sift_down(0)
    return ans
}
```

Комментарий

Проверяем, есть ли дети у этого элемента. Если детей нет, просеивать некуда

Как удалить минимум в пирамиде?

Рассмотрим псевдокод

```
Heap {
  ...

  sift_down(index):
    if детей у index нет: выход
    min_child = минимальный ребёнок index
    if data[index] > data[min_child]
      swap data[index] data[min_child]
      sift_down(min_child)

  extract_min(e):
    ans = data[0]
    data[0] = data[длина(data)-1]
    удалить у data 1 элемент с конца
    sift_down(0)
    return ans
}
```

Комментарий

Это индекс самого маленького ребёнка

Как удалить минимум в пирамиде?

Рассмотрим псевдокод

```
Heap {
  ...

  sift_down(index):
    if детей у index нет: выход
    min_child = минимальный ребёнок index
    if data[index] > data[min_child]
      swap data[index] data[min_child]
      sift_down(min_child)

  extract_min(e):
    ans = data[0]
    data[0] = data[длина(data)-1]
    удалить у data 1 элемент с конца
    sift_down(0)
    return ans
}
```

Комментарий
Проверяем, нарушается ли свойство пирамиды, сравнивая с минимальным значением из детей этого элемента

Как удалить минимум в пирамиде?

Рассмотрим псевдокод

```
Heap {
  ...

  sift_down(index):
    if детей у index нет: выход
    min_child = минимальный ребёнок index
    if data[index] > data[min_child]
      swap data[index] data[min_child]
      sift_down(min_child)

  extract_min(e):
    ans = data[0]
    data[0] = data[длина(data)-1]
    удалить у data 1 элемент с конца
    sift_down(0)
    return ans
}
```

Комментарий
Поменяемся местами с этим ребёнком

Как удалить минимум в пирамиде?

Рассмотрим псевдокод

```
Heap {
  ...

  sift_down(index):
    if детей у index нет: выход
    min_child = минимальный ребёнок index
    if data[index] > data[min_child]
      swap data[index] data[min_child]
      sift_down(min_child)

  extract_min(e):
    ans = data[0]
    data[0] = data[длина(data)-1]
    удалить у data 1 элемент с конца
    sift_down(0)
    return ans
}
```

Комментарий
Попытаемся просеиваться дальше

Как удалить минимум в пирамиде?

Рассмотрим псевдокод

```
Heap {
  ...

  sift_down(index):
    if детей у index нет: выход
    min_child = минимальный ребёнок index
    if data[index] > data[min_child]
      swap data[index] data[min_child]
      sift_down(min_child)

  extract_min(e):
    ans = data[0]
    data[0] = data[длина(data)-1]
    удалить у data 1 элемент с конца
    sift_down(0)
    return ans
}
```

Комментарий

Проводим операцию извлечения минимума из пирамиды

Как удалить минимум в пирамиде?

Рассмотрим псевдокод

```
Heap {
  ...

  sift_down(index):
    if детей у index нет: выход
    min_child = минимальный ребёнок index
    if data[index] > data[min_child]
      swap data[index] data[min_child]
      sift_down(min_child)

  extract_min(e):
    ans = data[0]
    data[0] = data[длина(data)-1]
    удалить у data 1 элемент с конца
    sift_down(0)
    return ans
}
```

Комментарий

Минимум нам известен — он в корне

Как удалить минимум в пирамиде?

Рассмотрим псевдокод

```
Heap {
  ...

  sift_down(index):
    if детей у index нет: выход
    min_child = минимальный ребёнок index
    if data[index] > data[min_child]
      swap data[index] data[min_child]
      sift_down(min_child)

  extract_min(e):
    ans = data[0]
    data[0] = data[длина(data)-1]
    удалить у data 1 элемент с конца
    sift_down(0)
    return ans
}
```

Комментарий

Помещаем в корень последний элемент массива

Как удалить минимум в пирамиде?

Рассмотрим псевдокод

```
Heap {
  ...

  sift_down(index):
    if детей у index нет: выход
    min_child = минимальный ребёнок index
    if data[index] > data[min_child]
      swap data[index] data[min_child]
      sift_down(min_child)

  extract_min(e):
    ans = data[0]
    data[0] = data[длина(data)-1]
    удалить у data 1 элемент с конца
    sift_down(0)
    return ans
}
```

Комментарий
Помечаем ячейку, откуда переместили элемент в корень, как свободную

Как удалить минимум в пирамиде?

Рассмотрим псевдокод

```
Heap {
  ...

  sift_down(index):
    if детей у index нет: выход
    min_child = минимальный ребёнок index
    if data[index] > data[min_child]
      swap data[index] data[min_child]
      sift_down(min_child)

  extract_min(e):
    ans = data[0]
    data[0] = data[длина(data)-1]
    удалить у data 1 элемент с конца
    sift_down(0)
    return ans
}
```

Комментарий
Пытаемся просеять новый корень

Как удалить минимум в пирамиде?

Рассмотрим псевдокод

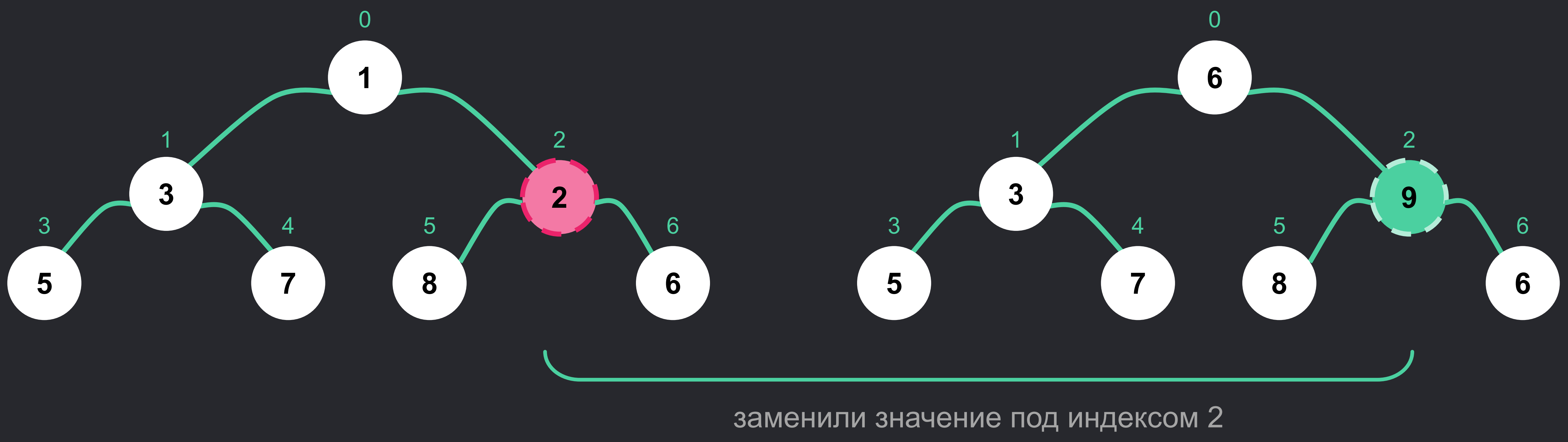
```
Heap {
  ...

  sift_down(index):
    if детей у index нет: выход
    min_child = минимальный ребёнок index
    if data[index] > data[min_child]
      swap data[index] data[min_child]
      sift_down(min_child)

  extract_min(e):
    ans = data[0]
    data[0] = data[длина(data)-1]
    удалить у data 1 элемент с конца
    sift_down(0)
    return ans
}
```

Комментарий
Теперь с пирамидой всё в порядке — возвращаем наш минимум

Другие операции с пирамидой

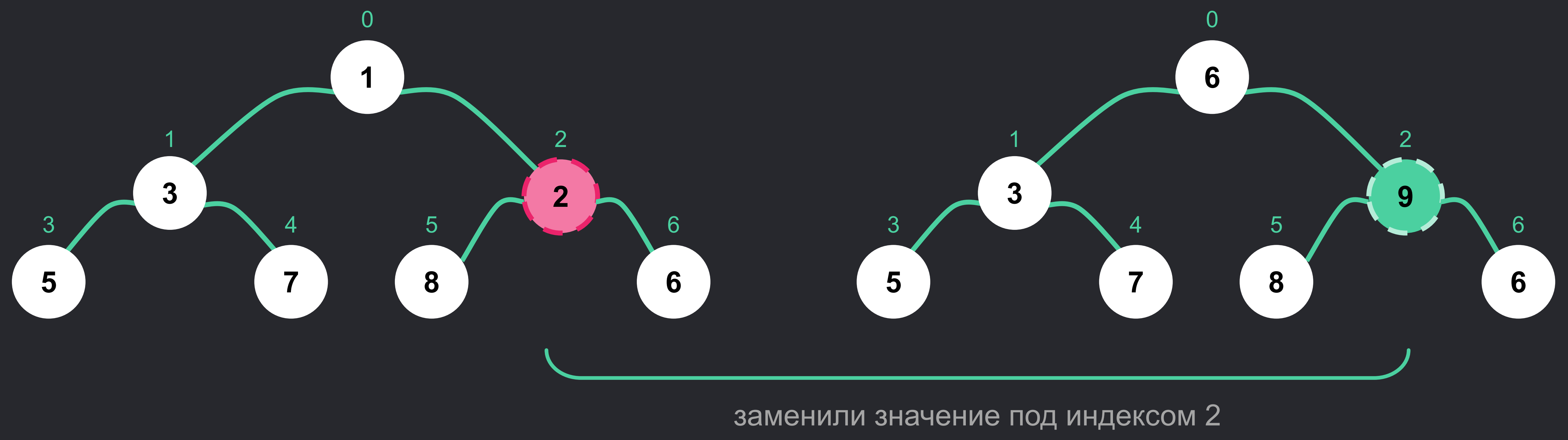
Другие операции с пирамидой

Изменение значения. В узле может быть нарушено свойство пирамиды, поэтому если мы увеличиваем значение элемента, после увеличения необходимо провести просеивание.
Это займёт $O(\log_2 n)$



Другие операции с пирамидой

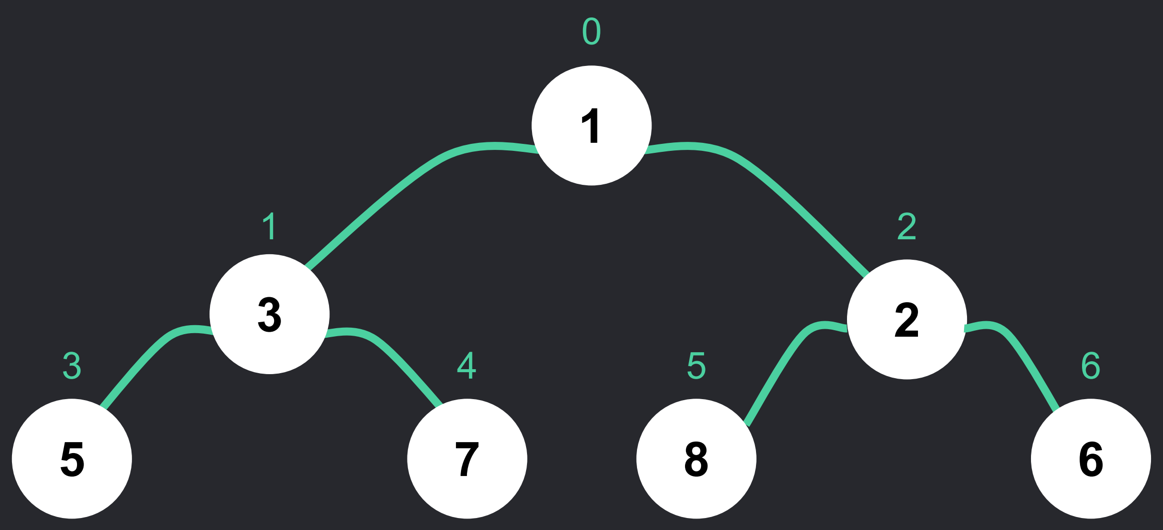
Изменение значения. В узле может быть нарушено свойство пирамиды, поэтому если мы уменьшаем значение элемента, после уменьшения необходимо провести всплывание.
Это займёт $O(\log_2 n)$



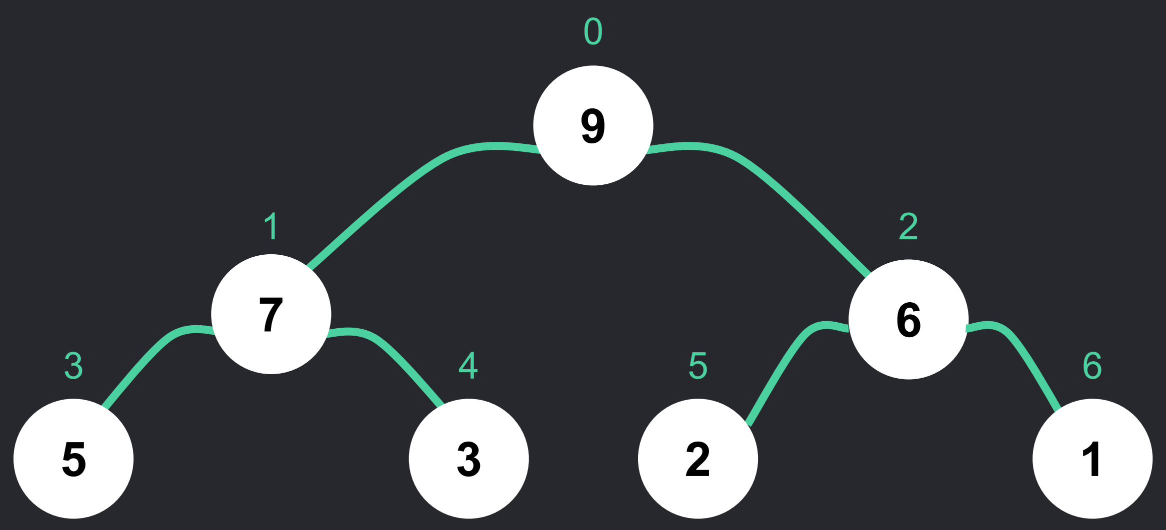
Другие операции с пирамидой

Поиск максимума. Пирамида на минимум не облегчает поиск максимума, для которого придется перебрать все элементы, т. е. O(n).

Если нам нужен только максимум, мы можем изначально строить пирамиду на максимум вместо пирамиды на минимум



пирамида на минимум

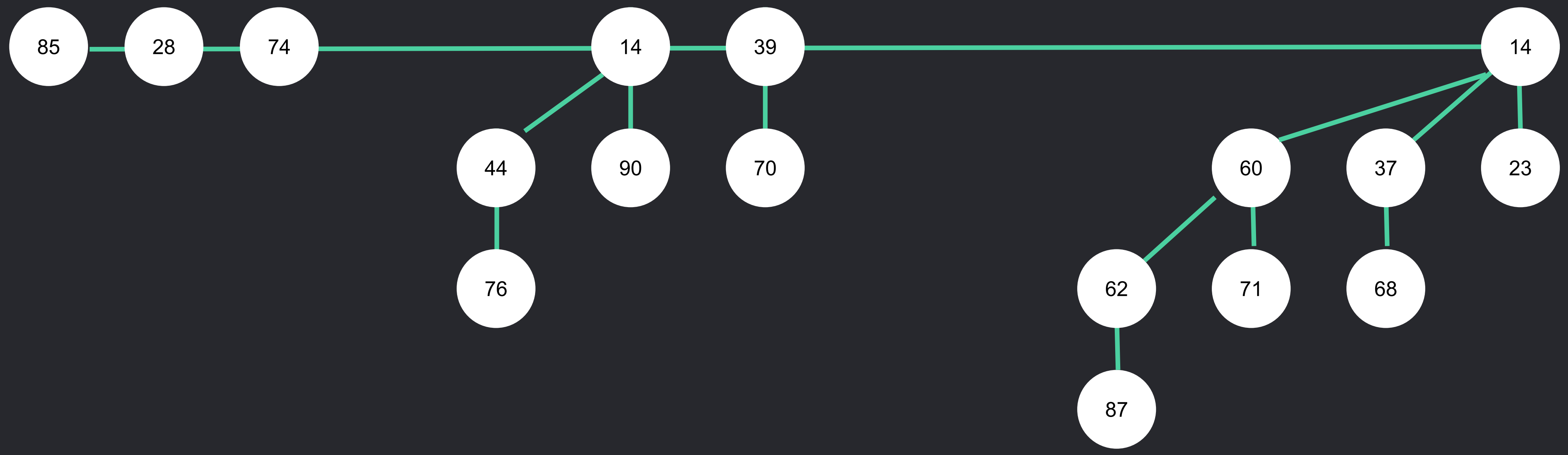


пирамида на максимум

Другие операции с пирамидой

Построение кучи. Мы можем легко построить кучу из n элементов за $O(n \log_2 n)$ — просто создав пустую кучу и поочерёдно за $O(\log_2 n)$ добавить туда каждый элемент. Но существует и алгоритм линейного построения кучи.

Существует множество модификаций пирамид, в том числе те, которые ускоряют операции вставки до амортизационной $O(1)$
и добавляют такие новые операции как объединение двух куч, причём иногда тоже за $O(1)$, например Фибоначчиевы кучи



Пирамидальная сортировка

Пирамидальная сортировка

Задача: необходимо отсортировать данный массив через использование пирамид

5	4	1	2	3	8	6
0	1	2	3	4	5	6

Пирамидальная сортировка

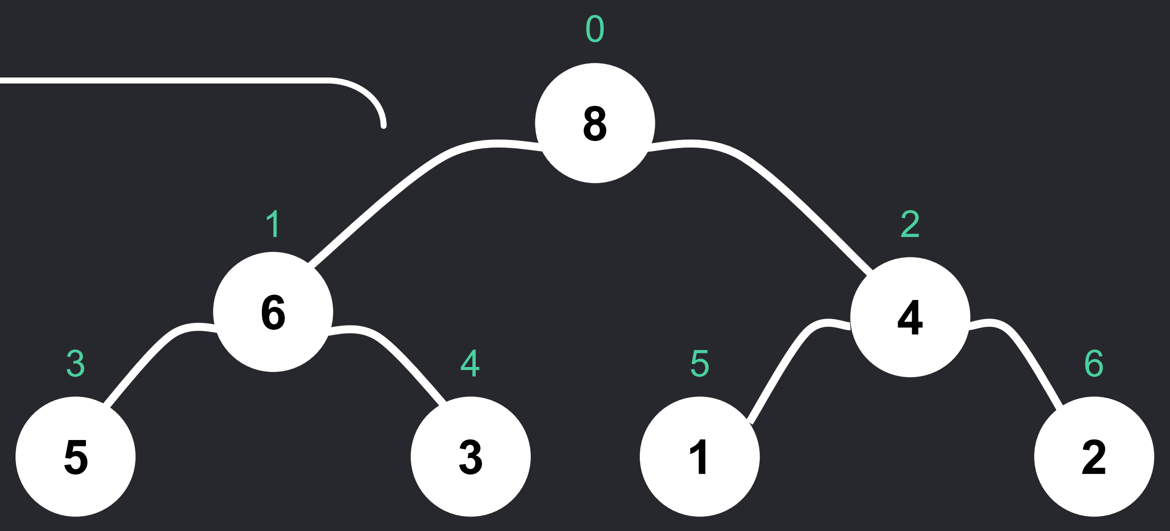
Реализация:

- преобразовать исходный массив в пирамиду на максимум на массиве

сделать за $O(n \log_2 n)$

линейно более сложным алгоритмом

5	4	1	2	3	8	6
0	1	2	3	4	5	6



Пирамидальная сортировка

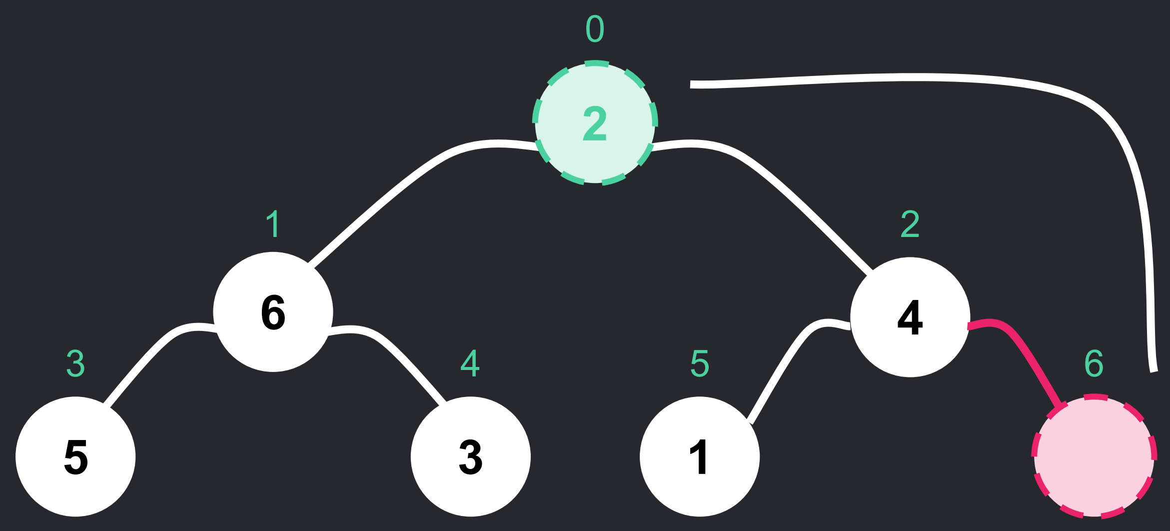
Реализация:

- поочерёдно извлекать максимум из пирамиды и вставлять в конец массива, пока она не кончится
  Кол-во извлечений — n, время каждого извлечения — $O(\log_2 n)$

Рассмотрим на этом же примере:

- извлекаем 8, перемещая его на последнее место

2	6	4	5	3	1	8
0	1	2	3	4	5	6



Пирамидальная сортировка

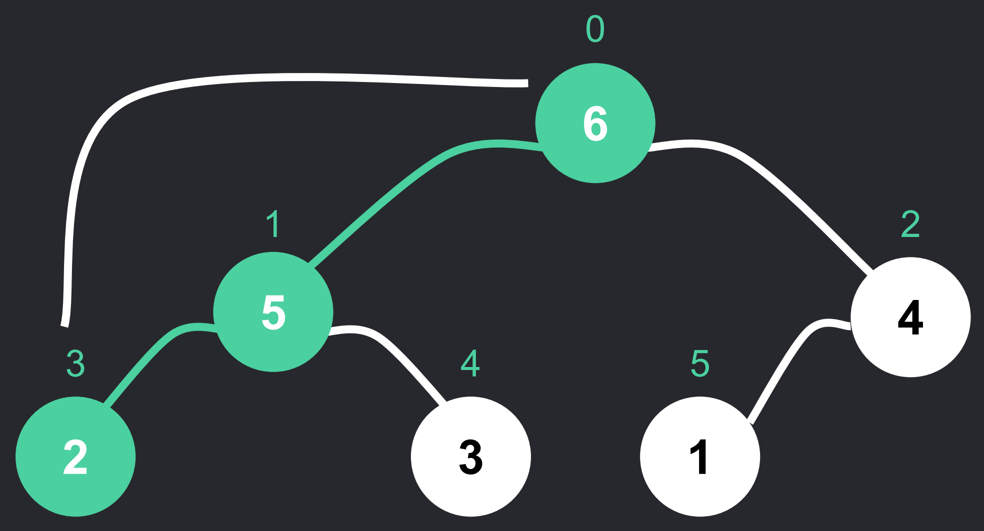
Реализация:

- поочерёдно извлекать максимум из пирамиды и вставлять в конец массива, пока она не кончится
  Кол-во извлечений — n, время каждого извлечения — $O(\log_2 n)$

Рассмотрим на этом же примере:

- извлекаем 8, перемещая его на последнее место → просеиваем

6	5	4	2	3	1	8
0	1	2	3	4	5	6



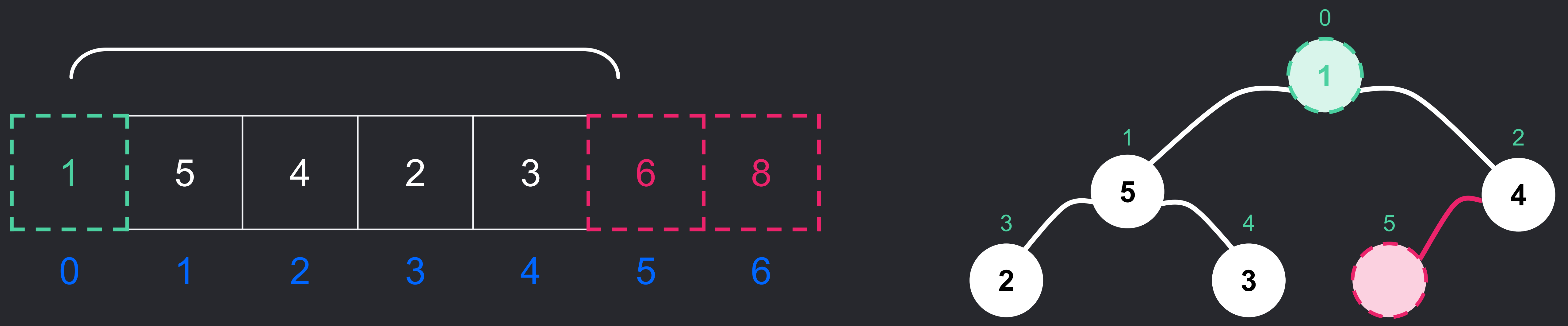
Пирамидальная сортировка

Реализация:

- поочерёдно извлекать максимум из пирамиды и вставлять в конец массива, пока она не кончится
  Кол-во извлечений — n, время каждого извлечения — $O(\log_2 n)$

Рассмотрим на этом же примере:

- извлекаем 8, перемещая его на последнее место → просеиваем
- извлекаем 6, перемещая его на последнее место



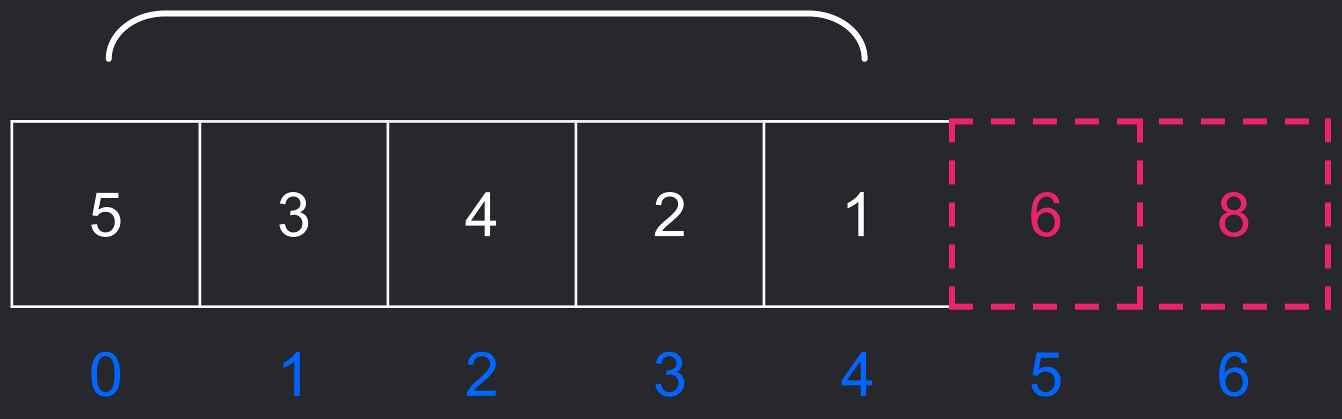
Пирамидальная сортировка

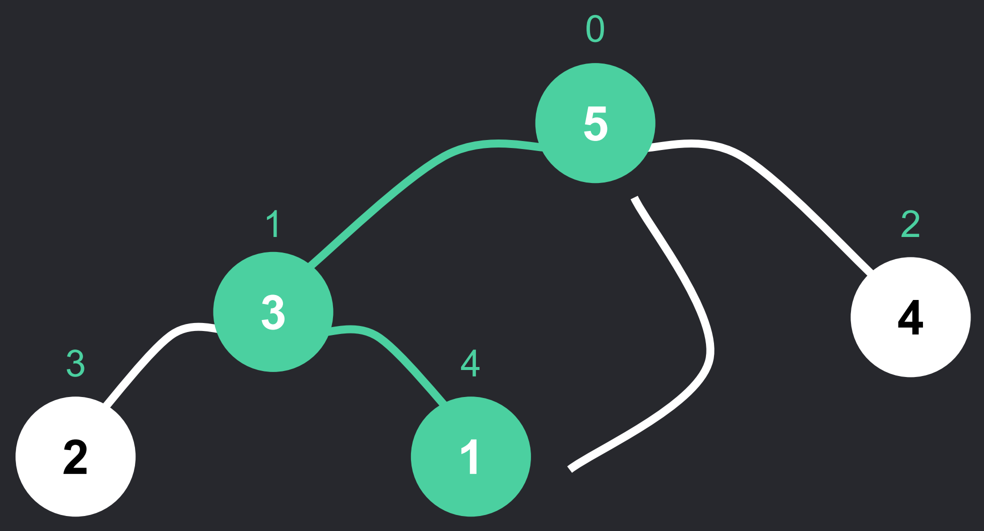
Реализация:

- поочерёдно извлекать максимум из пирамиды и вставлять в конец массива, пока она не кончится
  Кол-во извлечений — n, время каждого извлечения — $O(\log_2 n)$

Рассмотрим на этом же примере:

- извлекаем 8, перемещая его на последнее место → просеиваем
- извлекаем 6, перемещая его на последнее место → просеиваем и т. д.





Пирамидальная сортировка

Задача: необходимо отсортировать данный массив через использование пирамид.

5	4	1	2	3	8	5
0	1	2	3	4	5	6

Мы получили отсортированный массив:

1	2	3	4	5	6	8
0	1	2	3	4	5	6

Пирамидальная сортировка

Выводы:

- занимает времени $O(n \log_2 n)$
- неустойчивая — не сохраняет порядок на одинаковых для сравнения элементов
- на месте — не требует дополнительной памяти свыше $O(1)$

Визуализация: https://visualgo.net/en/sorting → MER

Очередь с приоритетами

Очередь с приоритетами

Очередь с приоритетами — это очередь, где у каждого элемента есть приоритет и операция next, которая реализована так, что из очереди извлекается элемент с максимальным приоритетом.

Такую очередь легко написать на пирамиде на максимум, где элементы сравниваются через их приоритеты

?

Что такое приоритет?

Чем больше какое-то число, тем оно приоритетнее

Очередь с приоритетами

Операция очереди с приоритетами	Операция пирамиды на максимум	Асимптотика
add	add	$O(\log_2 n)$
next	extractMax	$O(\log_2 n)$

В отличие от обычной очереди асимптотика уже не $O(1)$, но для практических целей $O(\log_2 n)$ — это тоже очень быстро, поэтому нет смысла в создавать Фибоначчиевы кучи для ускорения add до $O(1)$.

Поиск k максимумов

Поиск k максимумов: наивный алгоритм

Задача: есть массив из n элементов. Необходимо в нём найти k максимумов.

Рассмотрим наивную реализацию:

```
maxs = [k мин. допустимых значений]
for im от 0 до k
  for i от 0 до n
    if i нет в maxs[0..im-1]
      if arr[maxs[im]] < arr[i]
        maxs[im] = i
```

Комментарий
В этом массиве будем хранить ответ

Будем заполнять этот массив по порядку: сначала найдем первый максимум, затем второй и т. д.

Поиск k максимумов: наивный алгоритм

Задача: есть массив из n элементов. Необходимо в нём найти k максимумов.

Рассмотрим наивную реализацию:

```
maxs = [k мин. допустимых значений]
for im от 0 до k
  for i от 0 до n
    if i нет в maxs[0..im-1]
      if arr[maxs[im]] < arr[i]
        maxs[im] = i
```

Комментарий
Какой по счёту максимум ищем?

Поиск k максимумов: наивный алгоритм

Задача: есть массив из n элементов. Необходимо в нём найти k максимумов.

Рассмотрим наивную реализацию:

```
maxs = [k мин. допустимых значений]
for im от 0 до k
 for i от 0 до n
   if i нет в maxs[0..im-1]
     if arr[maxs[im]] < arr[i]
       maxs[im] = i
```

Комментарий
Пробегаемся по массиву

Поиск k максимумов: наивный алгоритм

Задача: есть массив из n элементов. Необходимо в нём найти k максимумов.

Рассмотрим наивную реализацию:

```
maxs = [k мин. допустимых значений]
for im от 0 до k
  for i от 0 до n
    if i нет в maxs[0..im-1]
      if arr[maxs[im]] < arr[i]
        maxs[im] = i
```

Комментарий

Если этот элемент мы еще не помещали в наш массив — то добавить его туда

Важно проверять, что мы не берём один и тот же элемент в качестве очередного максимума

Поиск k максимумов: наивный алгоритм

Задача: есть массив из n элементов. Необходимо в нём найти k максимумов.

Рассмотрим наивную реализацию:

```
maxs = [k мин. допустимых значений]
for im от 0 до k
  for i от 0 до n
    if i нет в maxs[0..im-1]
      if arr[maxs[im]] < arr[i]
        maxs[im] = i
```

Комментарий
Применяем обычный поиск максимума на массиве

Поиск k максимумов: наивный алгоритм

Задача: есть массив из n элементов. Необходимо в нём найти k максимумов.

Рассмотрим наивную реализацию:

```
maxs = [k мин. допустимых значений]
for im от 0 до k
  for i от 0 до n
    if i нет в maxs[0..im-1]
      if arr[maxs[im]] < arr[i]
        maxs[im] = i
```

Комментарий

Сохраняем индекс i как текущую версию индекса im-ного максимума

Поиск k максимумов: наивный алгоритм

Задача: есть массив из n элементов. Необходимо в нём найти k максимумов.

Алгосложность:

- количество итераций внешнего цикла — k
- количество итераций внутреннего цикла — n
- проверка на каждой вложенной итерации отсутствия текущего индекса в предыдущих максимумах — k

Итог: будет затрачено $O(n*k^2)$ времени, что при n = 100 млрд и k = 100 тыс.
Займёт 3 тысячелетия!

Поиск k максимумов: улучшенный алгоритм

Поиск k максимумов: улучшенный алгоритм

Задача: есть массив из n элементов. Необходимо в нём найти k максимумов.

Рассмотрим улучшенный подход:

```
maxs = []
for i от 0 до n
  if длина(maxs) < k
    добавить arr[i] в конец maxs
  else
    min = 0
    for im от 0 до k
      if maxs[im] < maxs[min]
        min = im
    if arr[i] > maxs[min]
      maxs[min] = arr[i]
```

Комментарий

Вспомогательный массив, в котором теперь будем хранить k максимумов среди всех просмотренных элементов

Поиск k максимумов: улучшенный алгоритм

Задача: есть массив из n элементов. Необходимо в нём найти k максимумов.

Рассмотрим улучшенный подход:

```
maxs = []
for i от 0 до n
  if длина(maxs) < k
    добавить arr[i] в конец maxs
  else
    min = 0
    for im от 0 до k
      if maxs[im] < maxs[min]
        min = im
    if arr[i] > maxs[min]
      maxs[min] = arr[i]
```

Комментарий
Всего один раз пробегаемся по массиву

Поиск k максимумов: улучшенный алгоритм

Задача: есть массив из n элементов. Необходимо в нём найти k максимумов.

Рассмотрим улучшенный подход:

```
maxs = []
for i от 0 до n
  if длина(maxs) < k
    добавить arr[i] в конец maxs
  else
    min = 0
    for im от 0 до k
      if maxs[im] < maxs[min]
        min = im
    if arr[i] > maxs[min]
      maxs[min] = arr[i]
```

Комментарий

Пока просмотрели меньше k элементов: просто добавляем каждый элемент в массив maxs

Поиск k максимумов: улучшенный алгоритм

Задача: есть массив из n элементов. Необходимо в нём найти k максимумов.

Рассмотрим улучшенный подход:

```
maxs = []
for i от 0 до n
  if длина(maxs) < k
    добавить arr[i] в конец maxs
  else
    min = 0
    for im от 0 до k
      if maxs[im] < maxs[min]
        min = im
    if arr[i] > maxs[min]
      maxs[min] = arr[i]
```

Комментарий
Иначе находим самый маленький из максимумов

Поиск k максимумов: улучшенный алгоритм

Задача: есть массив из n элементов. Необходимо в нём найти k максимумов.

Рассмотрим улучшенный подход:

```
maxs = []
for i от 0 до n
  if длина(maxs) < k
    добавить arr[i] в конец maxs
  else
    min = 0
    for im от 0 до k
      if maxs[im] < maxs[min]
        min = im
    if arr[i] > maxs[min]
      maxs[min] = arr[i]
```

Комментарий
Если рассматриваемый элемент больше этого минимума, то заменяем им его

Поиск k максимумов: улучшенный алгоритм

Задача: есть массив из n элементов. Необходимо в нём найти k максимумов.

В рамках улучшенного подхода по большому массиву мы пробегаемся только один раз. Если массив maxs ещё не дошел до размера k, просто добавляем новый элемент в maxs. Иначе ищем минимальный элемент в массиве максимумов и если он больше этого минимального, то заменяем его новым.

Поиск k максимумов: улучшенный алгоритм

Имея элемент с индексом i и k максимумов для первых i элементов, мы получаем k максимумов для первых (i + 1) элементов.

К примеру, у нас есть массив, k = 4, i = 5:

Массив элементов

11	43	23	66	44	10	63	45	25	26	31
0	1	2	3	4	5	6	7	8	9	10

Массив максимумов

44	43	63	66
0	1	2	3

Поиск k максимумов: улучшенный алгоритм

Мы получаем k максимумов для i = 5 + 1 = 6 элементов:

11	43	23	66	44	10	63	45	25	26	31
0	1	2	3	4	5	6	7	8	9	10

Массив элементов

44	43	63	66
0	1	2	3

Массив максимумов

Элемент со значением 23 выпадает

Поиск k максимумов: улучшенный алгоритм

Задача: есть массив из n элементов. Необходимо в нём найти k максимумов.

Алгосложность:

- количество итераций внешнего цикла — n
- поиск минимума в массиве максимумов — k итераций
- замена минимума на новый элемент — O(1)

Итог: будет затрачено O(n*k), что при n = 100 млрд и k = 100 тыс. займёт около 11 дней

Поиск k максимумов: улучшенный алгоритм

Задача: есть массив из n элементов. Необходимо в нём найти k максимумов.

Еще одно важное отличие: улучшенный подход — однопроходный, т. е. ему достаточно один раз увидеть каждый элемент, а не проходить по данным несколько раз.

Однопроходность

Зачем нужна однопроходность?
В некоторых случаях вы не хотите где-либо хранить исходные данные, ограничившись принципом «обработал элемент и забыл». В такой ситуации однопроходные алгоритмы помогают вам это сделать, а многопроходные — нет.

Многопроходные алгоритмы

Однопроходные алгоритмы

Поиск k максимумов: быстрый алгоритм

Поиск k максимумов: быстрый алгоритм

Псевдокод

```
maxs = new Heap
for i от 0 до n
  if длина(maxs) < k
    добавить arr[i] в maxs
  else
    if arr[i] > min(maxs)
      извлечь минимум из maxs
      добавить arr[i] в maxs
```

Комментарий

Храним k текущих максимумов в пирамиде на минимум вместо массива

Поиск k максимумов: быстрый алгоритм

Псевдокод

```
maxs = new Heap
for i от 0 до n
  if длина(maxs) < k
    добавить arr[i] в maxs
  else
    if arr[i] > min(maxs)
      извлечь минимум из maxs
      добавить arr[i] в maxs
```

Комментарий
Добавляем теперь в пирамиду, а не в массив

Поиск k максимумов: быстрый алгоритм

Псевдокод

```
maxs = new Heap
for i от 0 до n
  if длина(maxs) < k
    добавить arr[i] в maxs
  else
    if arr[i] > min(maxs)
      извлечь минимум из maxs
      добавить arr[i] в maxs
```

Комментарий

Логика осталась, но поиск и замена минимума среди набора k чисел теперь алгоритмически быстрее

Поиск k максимумов: быстрый алгоритм

Алгосложность		
Операция	Асимптотика	
	в массиве максимумов	в пирамиде
Добавление в конец	O(1)	$O(\log_2 k)$
Поиск минимума	O(k)	$O(\log_2 k)$
Замена минимума на новый элемент	O(1)	$O(\log_2 k)$

Итог: временная сложность алгоритма превратилась в $O(n \log_2 k)$, что для нашего алгоритма даёт меньше 3-х минут. Свойство однопроходности алгоритма сохраняется

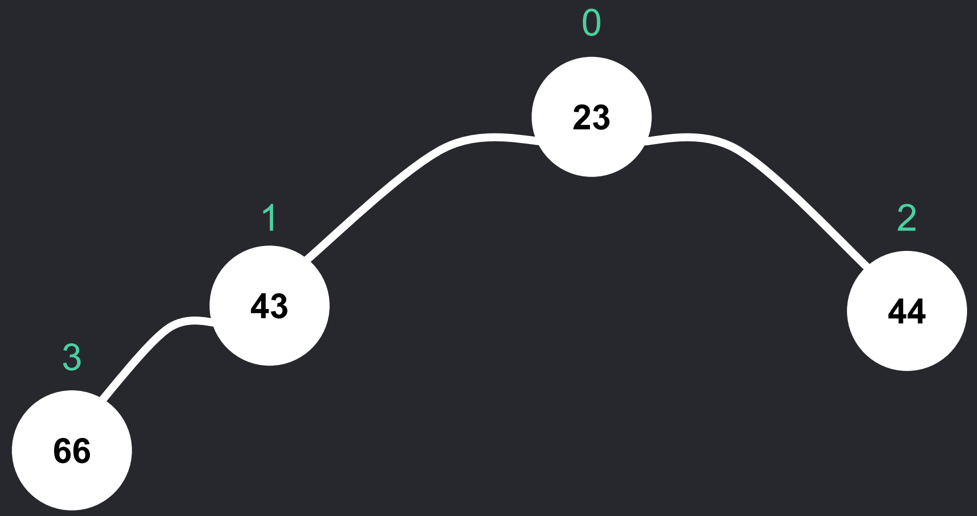
Поиск k максимумов: быстрый алгоритм

Имея элемент с индексом i и k максимумов для первых i элементов, мы получаем k максимумов для первых (i + 1) элементов.

К примеру, у нас есть массив, k = 4, i = 5:

11	43	23	66	44	10	63	45	25	26	31
0	1	2	3	4	5	6	7	8	9	10

Массив элементов



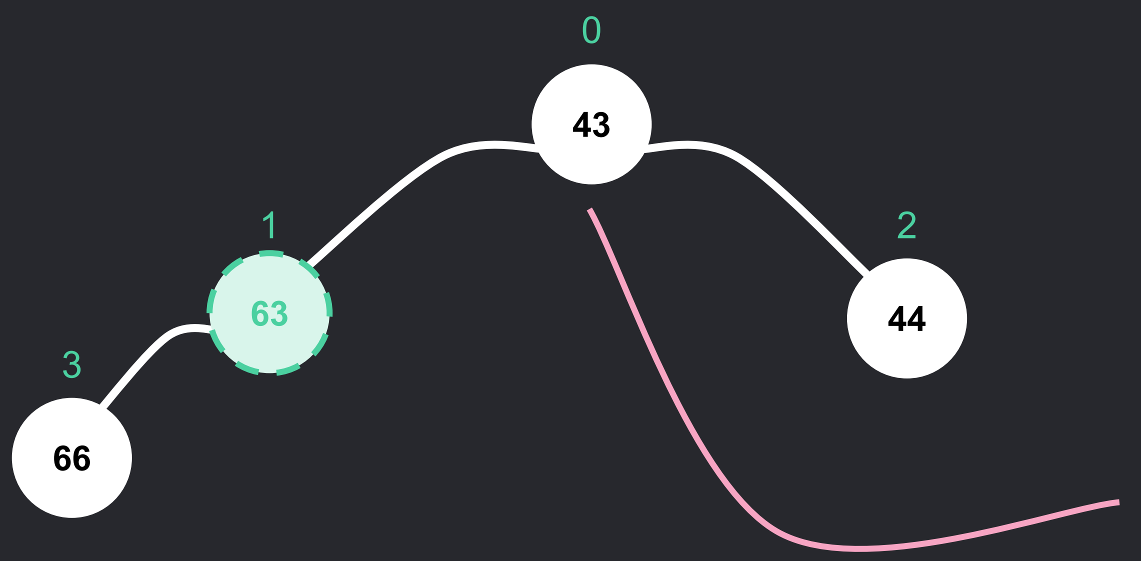
Пирамида максимумов

Поиск k максимумов: быстрый алгоритм

Мы получаем k максимумов для i = 5 + 1 = 6 элементов:

0	1	2	3	4	5	6	7	8	9	10
11	43	23	66	44	10	63	45	25	26	31

Массив элементов



Пирамида максимумов

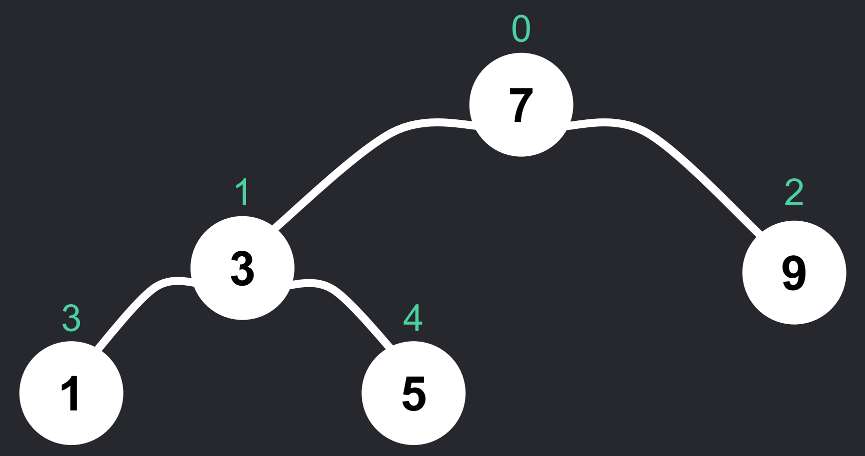
Элемент со значением 23 выпадает из пирамиды

Деревья поиска

Двоичные деревья поиска

Дерево поиска — бинарное дерево, где для каждого узла соблюдается условие:

- левый ребёнок и его потомки меньше него
- правый ребёнок и его потомки больше него

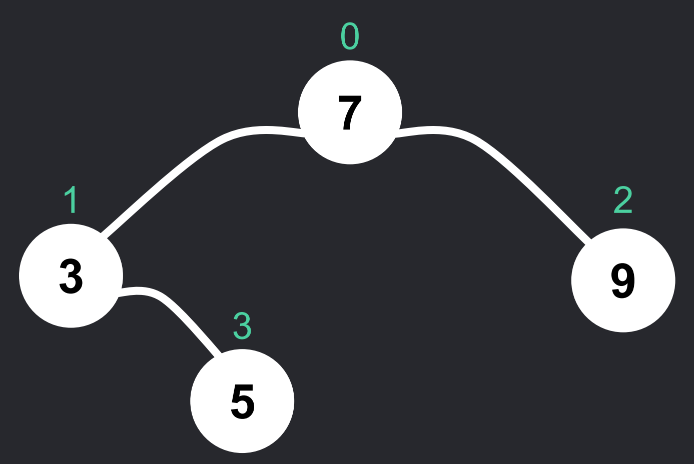


Примечание: в нашем дереве поиска нет повторов, но не сложно его обобщить и на такой случай

Двоичные деревья поиска

Дерево поиска — бинарное дерево, где для каждого узла соблюдается условие:

- левый ребёнок и его потомки меньше него
- правый ребёнок и его потомки больше него

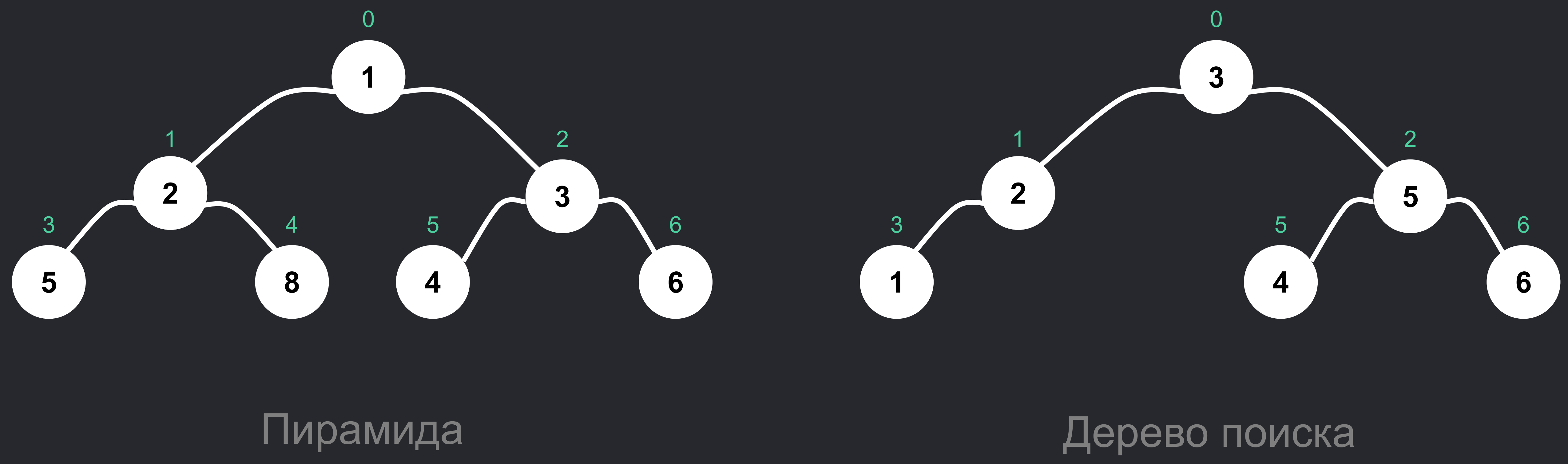


Заметим, что дерево может быть неполным двоичным, и реализовать
на динамическом массиве как с пирамидами уже не получится

Двоичные деревья поиска

В чём отличие деревьев поиска от пирамид?

Простые пирамиды — всегда полные двоичные деревья, а деревья поиска — не всегда. У пирамид каждый узел больше или меньше своих детей,
а в деревьях поиска каждый узел больше левого ребенка и меньше правого



Интерактив

Покажем разные деревья, а они пусть выбирают что это: дерево поиска, пирамида или ничего из этого

Поиск и вставка в двоичные деревья поиска

Поиск

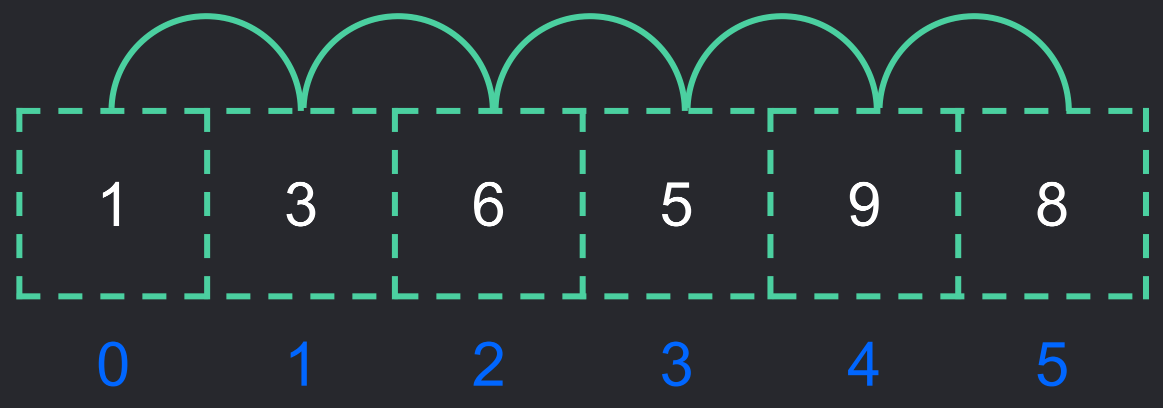
У каждого элемента есть ключ — то, по чему дерево поиска сравнивает элементы на меньше, равно или больше.

Давайте реализуем операцию поиска элемента по заданному ключу.

Поиск и вставка в двоичные деревья поиска

Реализуем операцию поиска элемента по заданному ключу:

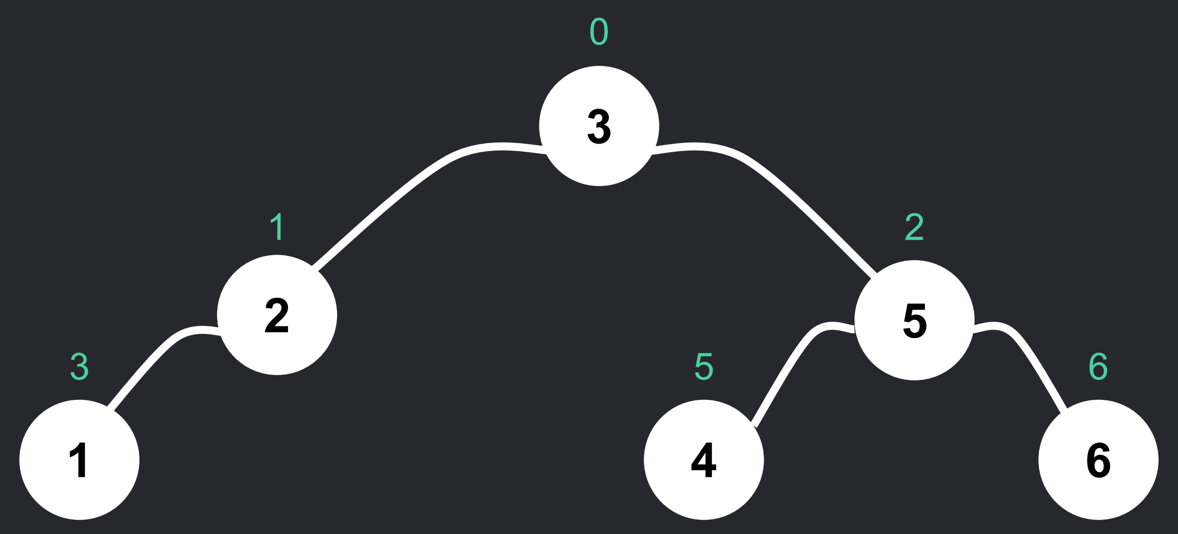
- если бы мы искали в обычном массиве, нам пришлось бы пройтись по всем элементам, сравнивая ключи



Поиск и вставка в двоичные деревья поиска

Реализуем операцию поиска элемента по заданному ключу:

- если мы ищем в двоичных деревьях поиска, мы можем начать с корня и в зависимости от того больше или меньше ключ — пойти вправо или влево

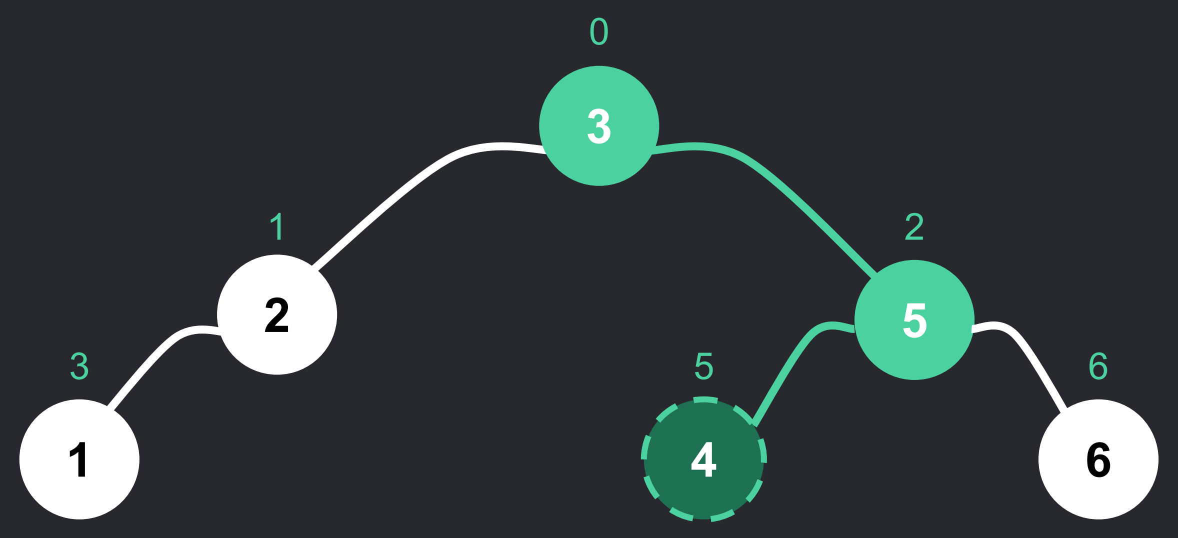


Поиск и вставка в двоичные деревья поиска

Рассмотрим пример: необходимо найти элемент со значением 4.

Реализация:

- в корне 3, значит искомый элемент находится справа (4 больше 3)
- попадаем на элемент 5, значит искомый элемент левее (5 больше 4)
- нашли искомый элемент

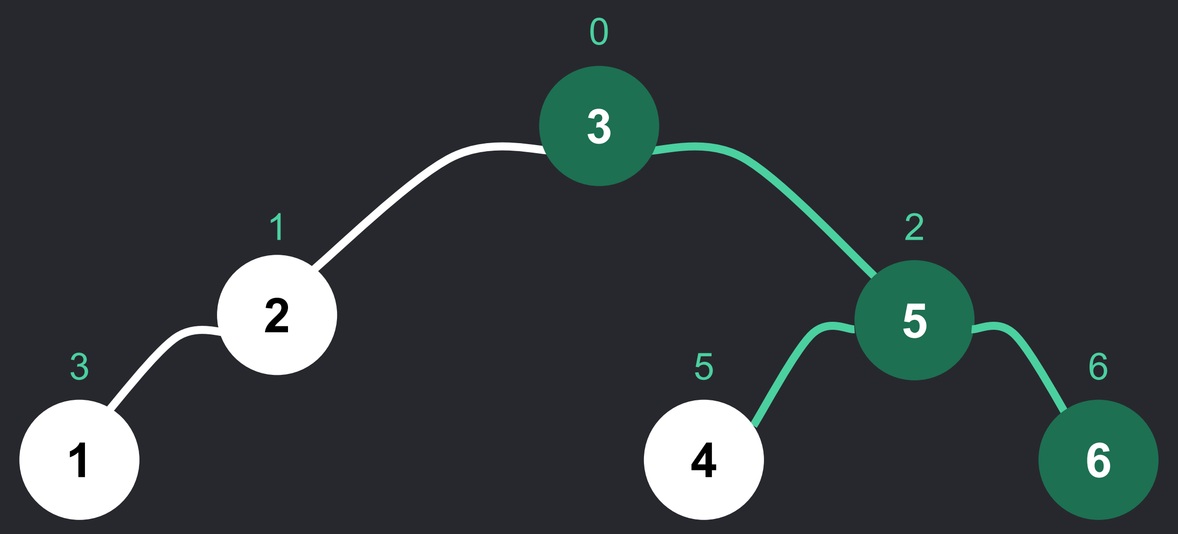


Поиск и вставка в двоичные деревья поиска

Добавление

Для этого повторим операцию поиска, только в случае, когда нам некуда пойти (нет такого элемента) просто вставляем в это место новый элемент как ребёнка.

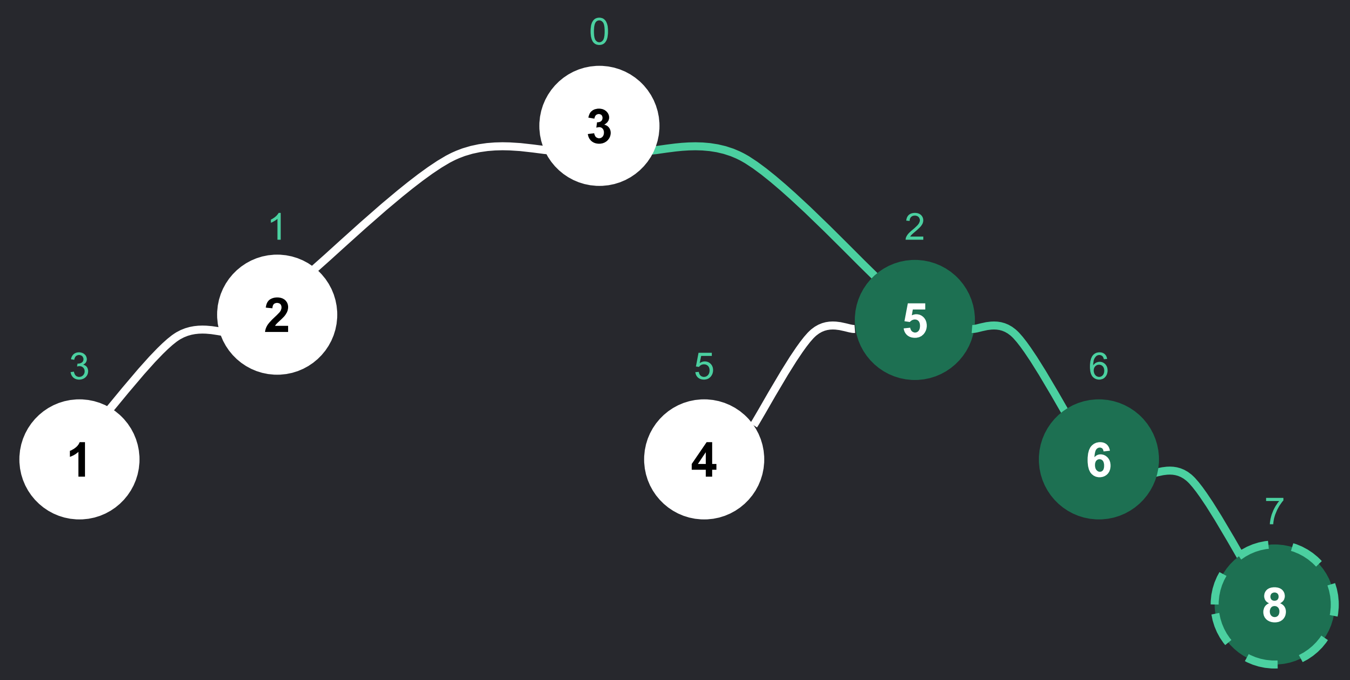
Предположим, мы хотим добавить элемент со значением 8.
Для этого мы запускаем операцию поиска:



Ищем элемент со значением 8

Поиск и вставка в двоичные деревья поиска

Вставляем новый элемент как ребёнка:



Вставляем элемент со значением 8

Поиск и вставка в двоичные деревья поиска

Рассмотрим псевдокод поиска:

```
BinTree {
  …

  search(key):
    search(key, root)

  search(key, node):
    if key = node
      return node
    else if key < node
      if есть левый сын у node
        return search(key, node.left)
      else return нет такого элемента
    else if key > node
      if есть правый сын у node
        return search(key, node.right)
      else return нет такого элемента
}
```

Комментарий
Для поиска элемента по ключу начнем поиск с корня

Визуализация двоичных деревьев поиска:
https://visualgo.net/en/bst?slide=1

Поиск и вставка в двоичные деревья поиска

Рассмотрим псевдокод поиска:

```
BinTree {
  …

  search(key):
    search(key, root)

  search(key, node):
    if key = node
      return node
    else if key < node
      if есть левый сын у node
        return search(key, node.left)
      else return нет такого элемента
    else if key > node
      if есть правый сын у node
        return search(key, node.right)
      else return нет такого элемента
}
```

Комментарий

Если мы ищем элемент, находясь в узле node

Поиск и вставка в двоичные деревья поиска

Рассмотрим псевдокод поиска:

```
BinTree {
  …

  search(key):
    search(key, root)

  search(key, node):
    if key = node
      return node
    else if key < node
      if есть левый сын у node
        return search(key, node.left)
      else return нет такого элемента
    else if key > node
      if есть правый сын у node
        return search(key, node.right)
      else return нет такого элемента
}
```

Комментарий

Если ключ узла совпадает с тем, который мы ищем, то мы его нашли

Поиск и вставка в двоичные деревья поиска

Рассмотрим псевдокод поиска:

```
BinTree {
  …

  search(key):
    search(key, root)

  search(key, node):
    if key = node
      return node
    else if key < node
      if есть левый сын у node
        return search(key, node.left)
      else return нет такого элемента
    else if key > node
      if есть правый сын у node
        return search(key, node.right)
      else return нет такого элемента
}
```

Комментарий

Если искомый ключ меньше, то идём искать у левого ребёнка текущего узла, если он у него есть

Поиск и вставка в двоичные деревья поиска

Рассмотрим псевдокод поиска:

```
BinTree {
  …

  search(key):
    search(key, root)

  search(key, node):
    if key = node
      return node
    else if key < node
      if есть левый сын у node
        return search(key, node.left)
      else return нет такого элемента
    else if key > node
      if есть правый сын у node
        return search(key, node.right)
      else return нет такого элемента
}
```

Комментарий

Если искомый ключ больше, то идём искать у правого ребёнка текущего узла, если он у него есть

Поиск и вставка в двоичные деревья поиска

Асимптотика: и добавление, и поиск работают за $O(h)$, где h — это глубина.

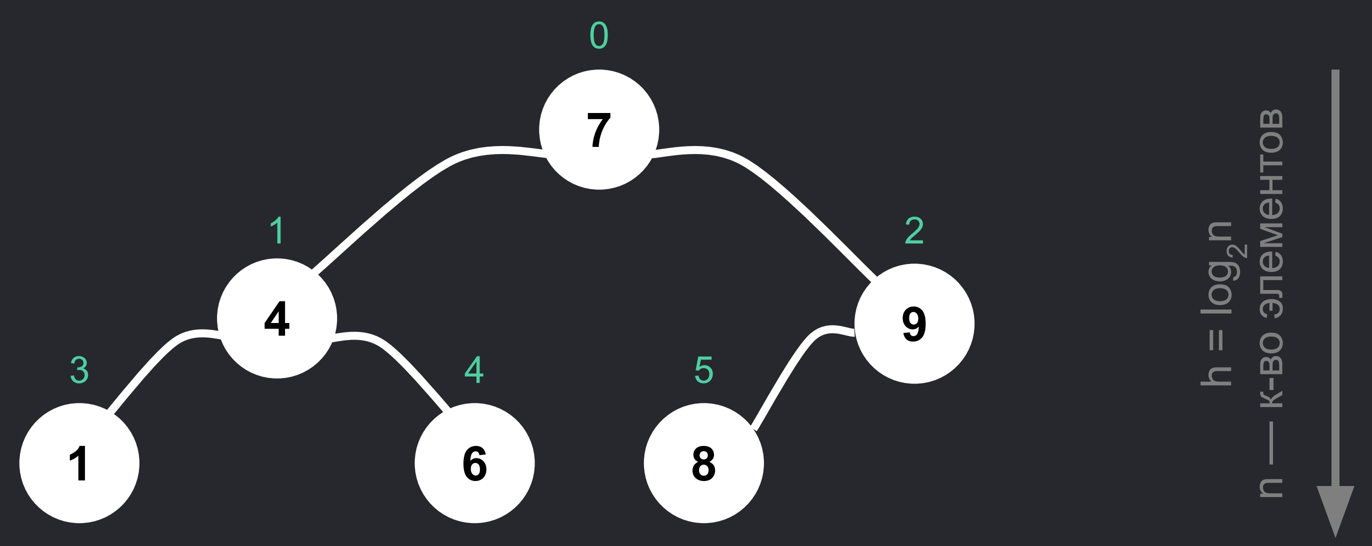
Глубина необязательно логарифм из-за неполноты двоичного дерева поиска. Она может быть чем угодно, но не больше $O(n)$

Несбалансированность и красно-чёрные деревья

Несбалансированность

Сбалансированное дерево — такое дерево, у которого глубина $O(\log_2 n)$.
В идеале $\sim\log_2 n$.

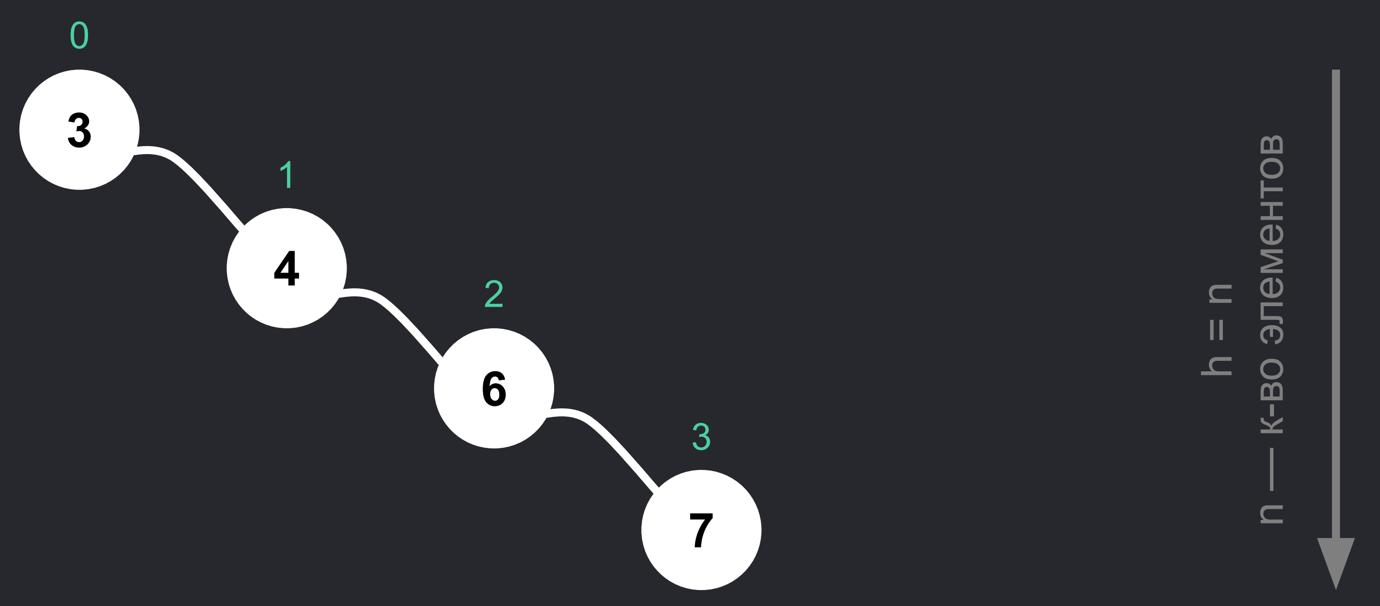
В таком случае основные операции над деревом поиска: $O(h) = O(\log_2 n)$.



Несбалансированность

Несбалансированное дерево — такое дерево, у которого высота асимптотически хуже чем $O(\log_2 n)$. В худшем случае равно n.

Ниже представлено корректное дерево поиска, которое называется «бамбук». Основные операции над деревом поиска: $O(h) = O(\log_2 n)$, что не лучше, чем обычный массив



Красно-чёрные деревья

Red-Black Tree

Самым популярным сбалансированным деревом поиска является красно-чёрное дерево. Оно присваивает каждому узлу цвет и задаёт инвариант на цвета в дереве, который гарантирует высоту $O(\log_2 n)$.

С помощью операций-поворотов реализуются все базовые операции над деревом, которые сохраняют инвариант и работают за $O(h)$