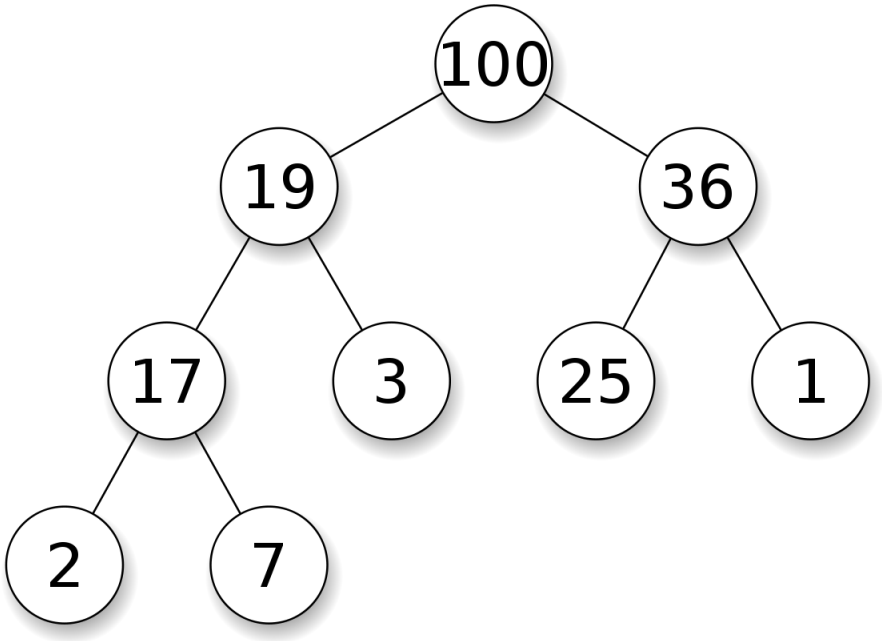
Декартово дерево

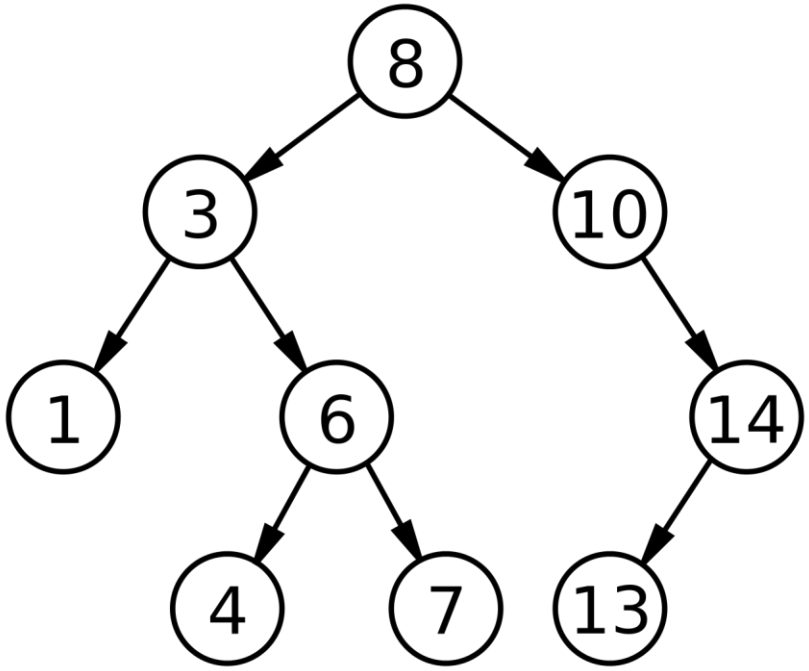
Двоичное дерево поиска (англ. *binary search tree*, BST) — это двоичное дерево, для которого выполняются следующие дополнительные условия (*свойства дерева поиска*):

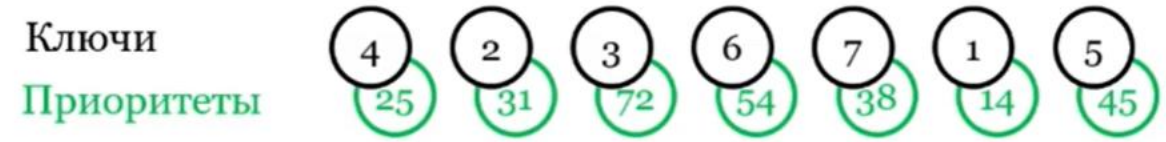
- Оба поддерева — левое и правое — являются двоичными деревьями поиска.
- У всех узлов *левого* поддерева произвольного узла X значения ключей данных *меньше либо равны*, нежели значение ключа данных самого узла X.
- У всех узлов *правого* поддерева произвольного узла X значения ключей данных *больше либо равны*, нежели значение ключа данных самого узла X.

Двои́чная ку́ча, **пирами́да**[1], или **сортиру́ющее де́рево** — такое двоичное дерево, для которого выполнены три условия:

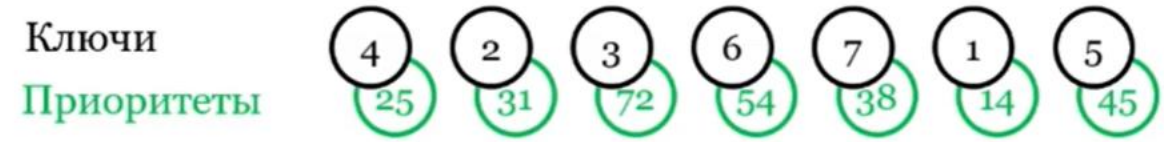
1. Значение в любой вершине не меньше, чем значения её потомков[K 1].
2. Глубина всех листьев (расстояние до корня) отличается не более чем на 1 слой.
3. Последний слой заполняется слева направо без «дырок».



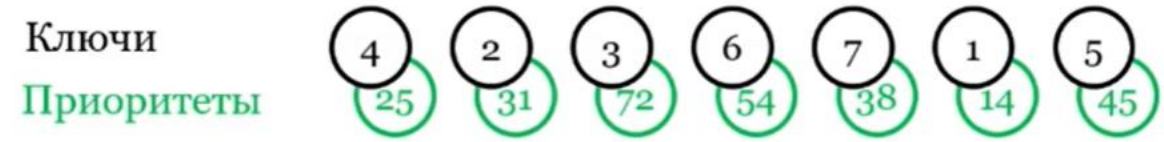




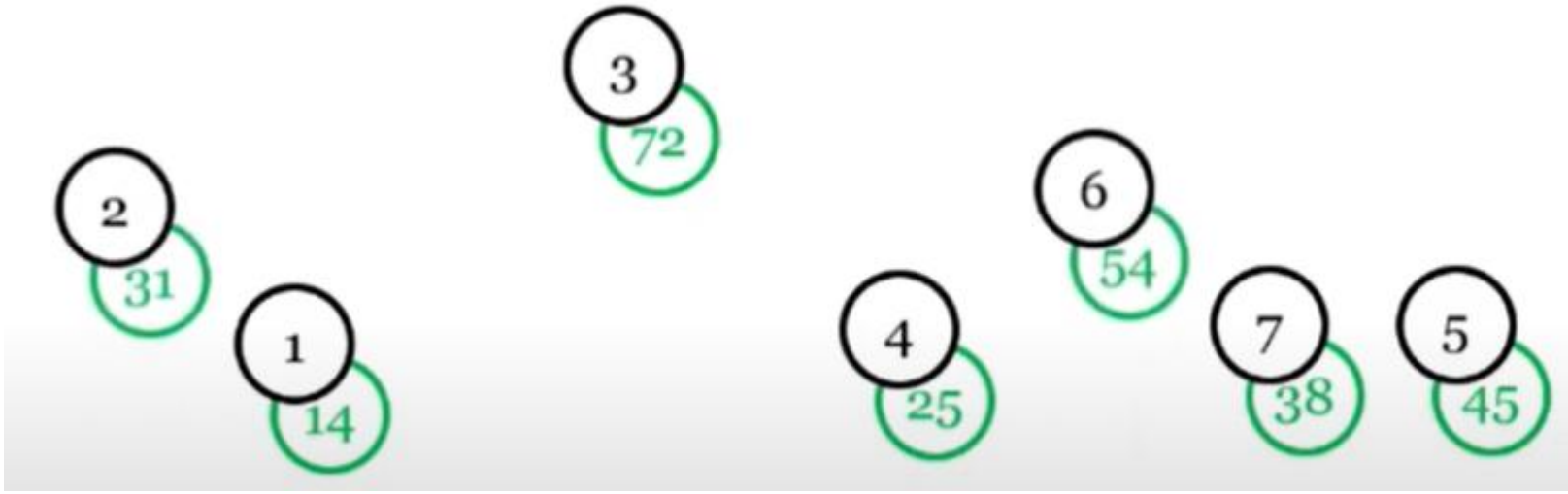
Декартово дерево:
по ключам — дерево поиска (меньше — левее, больше — правее);
по приоритетам — куча (меньше — ниже, больше — выше).

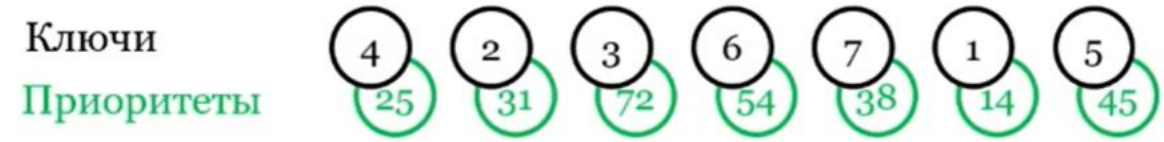


Декартово дерево:
по ключам — дерево поиска (меньше — левее, больше — правее);
по приоритетам — куча (меньше — ниже, больше — выше).

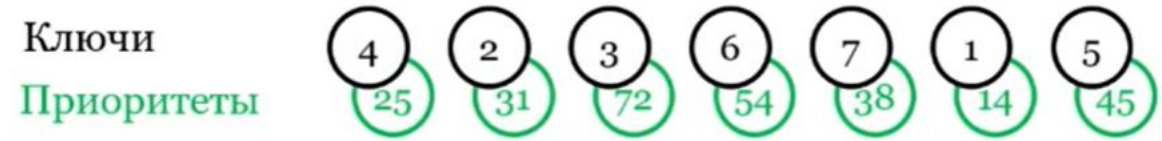


Декартово дерево:
по ключам — дерево поиска (меньше — левее, больше — правее);
по приоритетам — куча (меньше — ниже, больше — выше).

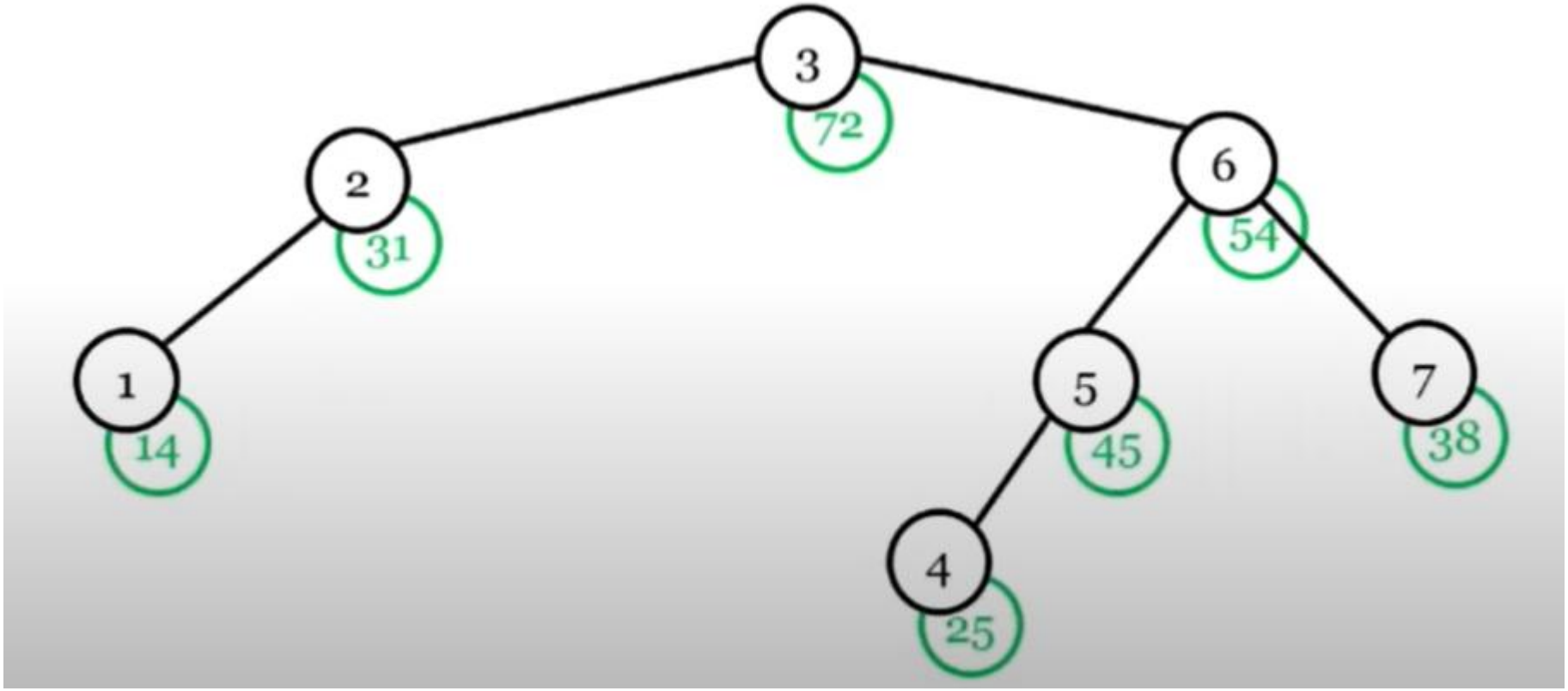




Декартово дерево:
по ключам — дерево поиска (меньше — левее, больше — правее);
по приоритетам — куча (меньше — ниже, больше — выше).

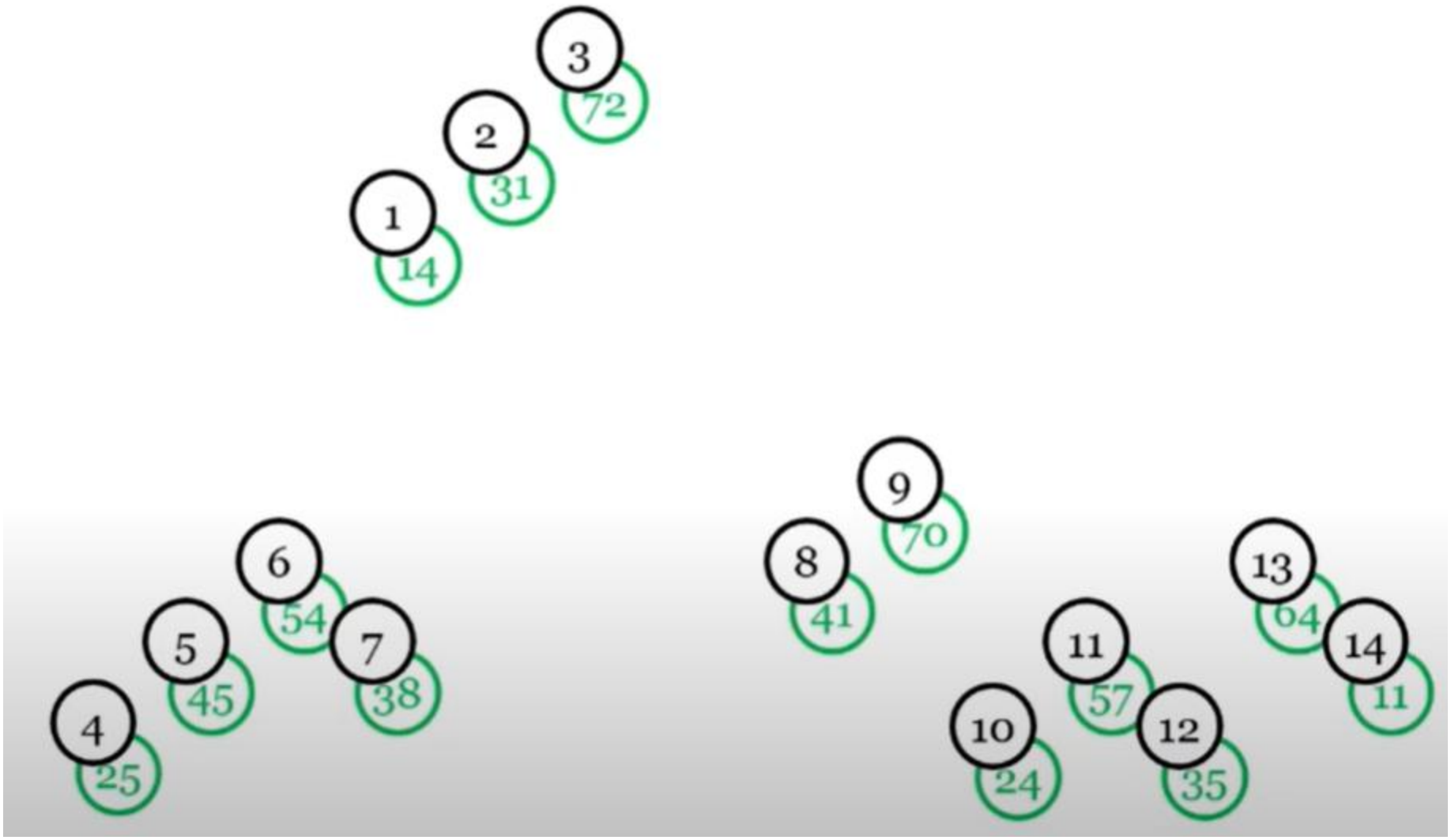


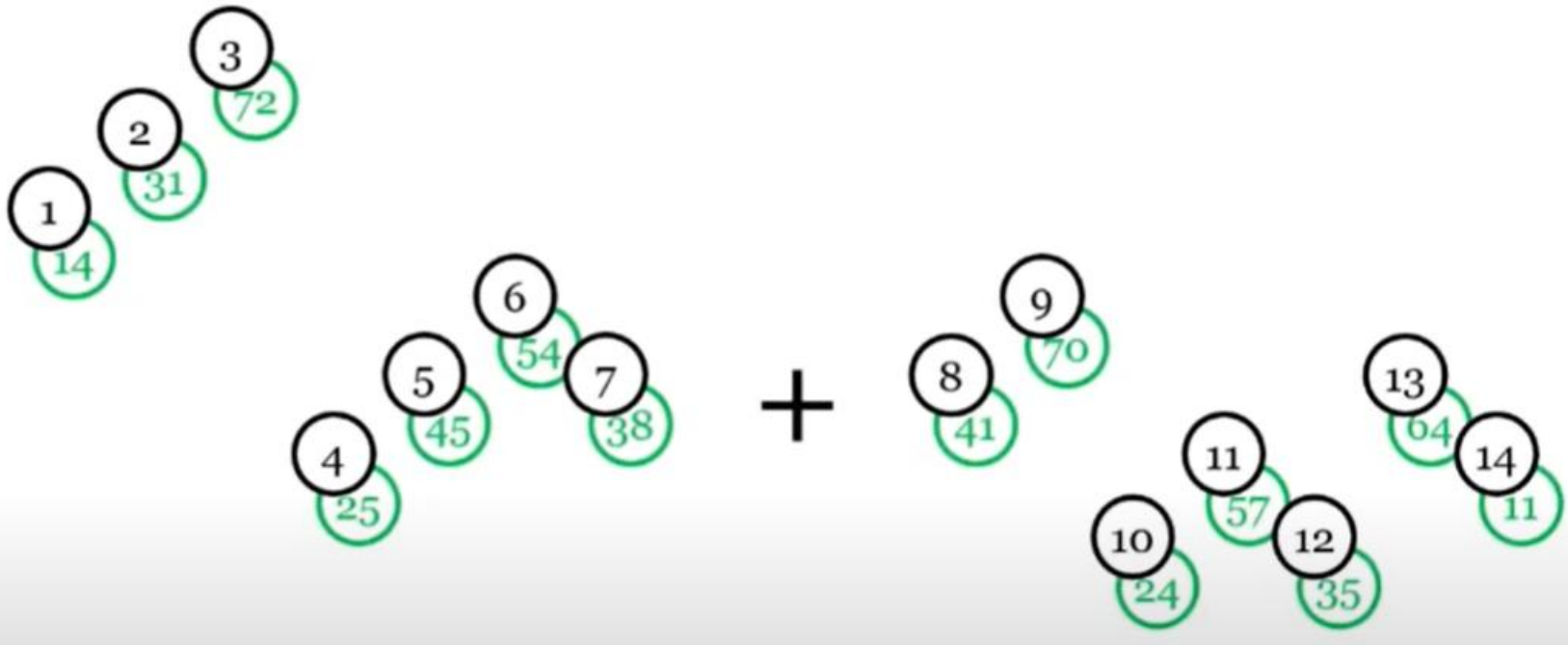
Декартово дерево:
по ключам — дерево поиска (меньше — левее, больше — правее);
по приоритетам — куча (меньше — ниже, больше — выше).

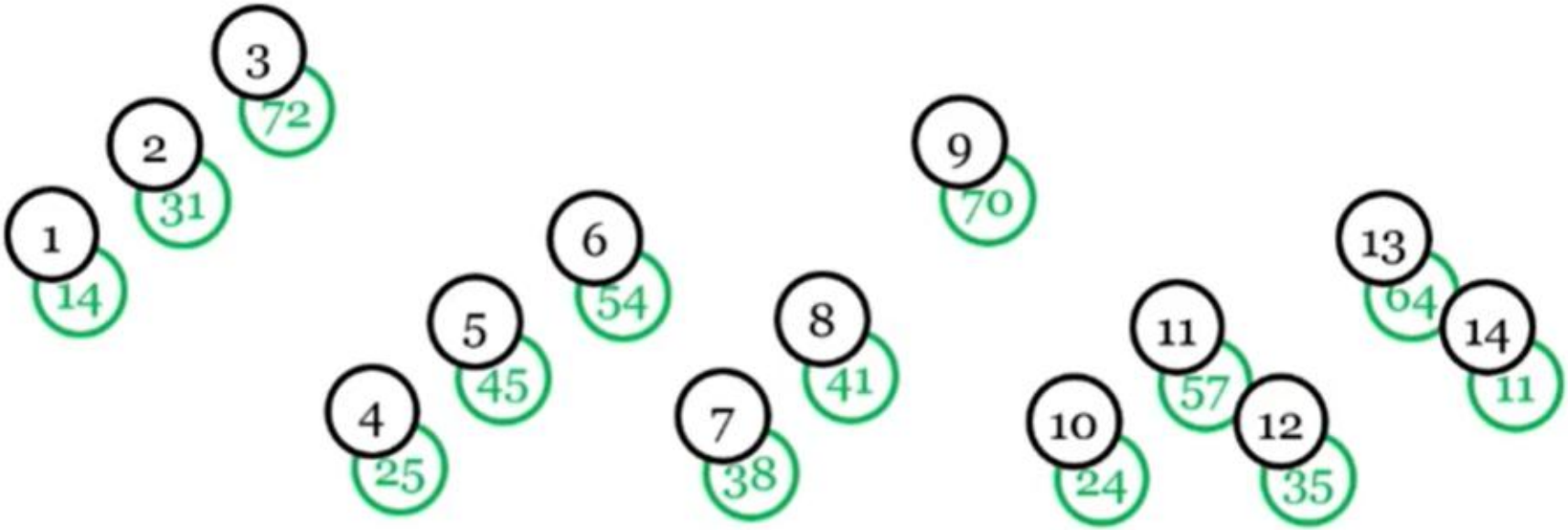


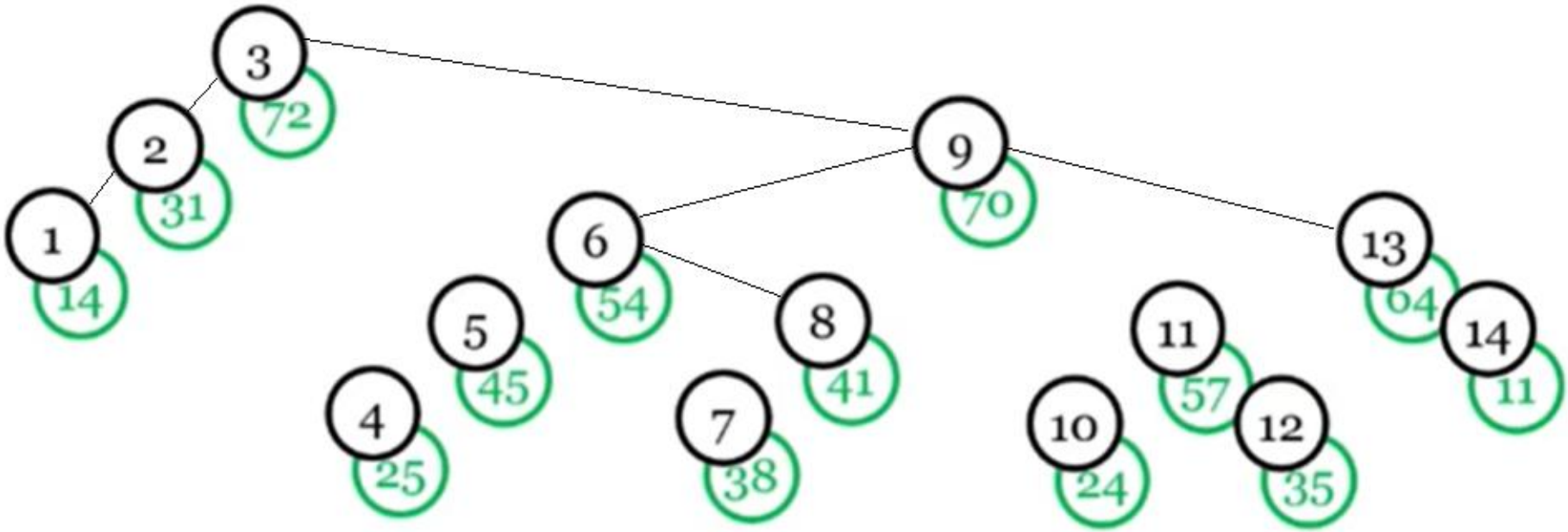
```cpp
#include <random>

using namespace std;

class Treap {
    static minstd_rand generator;

    struct Node {
        int key, priority;
        Node *l = nullptr, *r = nullptr;
        Node (int key): key(key), priority(generator()) {}
    } *root = nullptr;

};

minstd_rand Treap::generator;
```

Операция merge:
Объединение двух ДД в одно, ключи в первом меньше ключей во втором


3
72
2
31
1
14
9
70
8
41
6
54
5
45
7
38
4
25
13
64
14
11
11
57
10
24
12
35


3
72
2
31
1
14
6
54
5
45
7
38
4
25
+
9
70
8
41
13
64
11
14
57
11
10
12
24
35


3
72
2
31
1
14
6
54
5
45
4
25
9
70
8
41
7
38
13
64
14
11
11
57
12
35
10
24


3
72
2
31
1
14
9
70
6
54
8
41
5
45
4
25
7
38
13
64
14
11
11
57
12
35
10
24

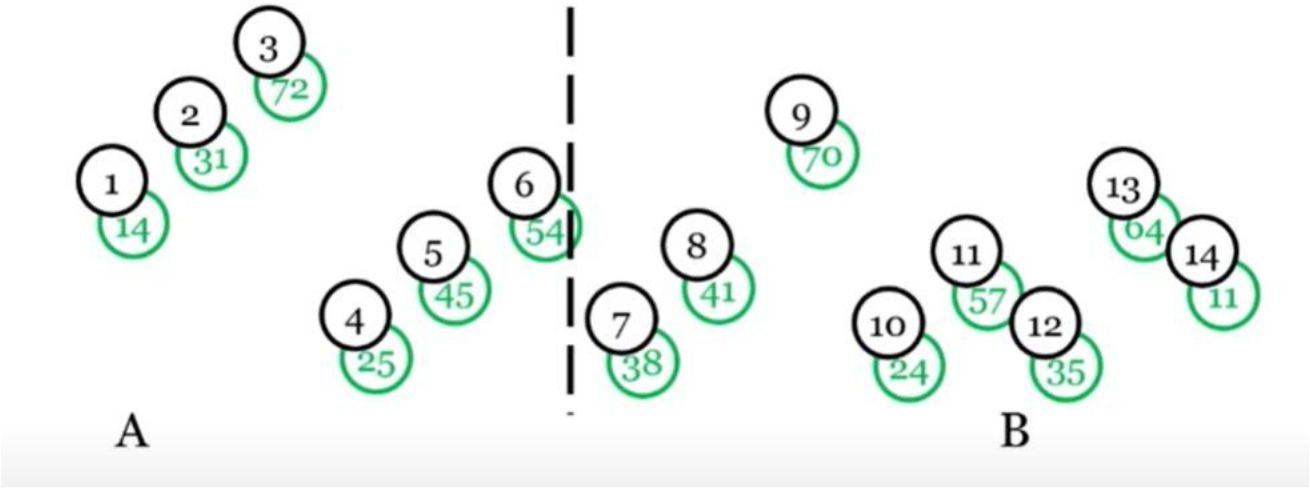
```
static Node *merge(Node *a, Node *b){
    if (!a || !b){
        return a ? a : b;
    }
    if (a -> priority > b -> priority){
        a -> r = merge(a -> r, b);
        return a;
    }
    else {
        b -> l = merge(a, b -> l);
        return b;
    }
}
```

Операция split:
Разделить ДД на два по ключу k, в первом все ключи меньше k, во втором – все остальные значения

Операция split:

Разделить ДД на два по ключу k, в первом все ключи меньше k, во втором – все остальные значения

Пример разделения дерева по ключу 7



Операция split:

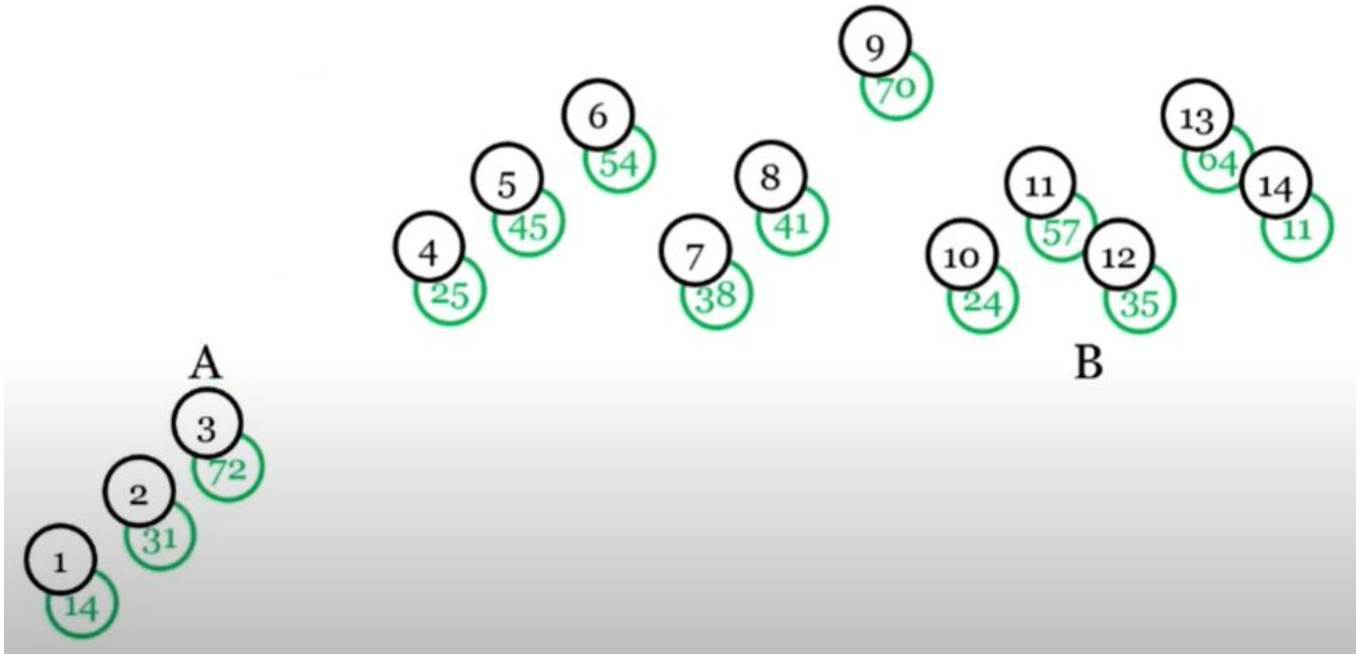
Разделить ДД на два по ключу k, в первом все ключи меньше k, во втором – все остальные значения

Пример разделения дерева по ключу 7

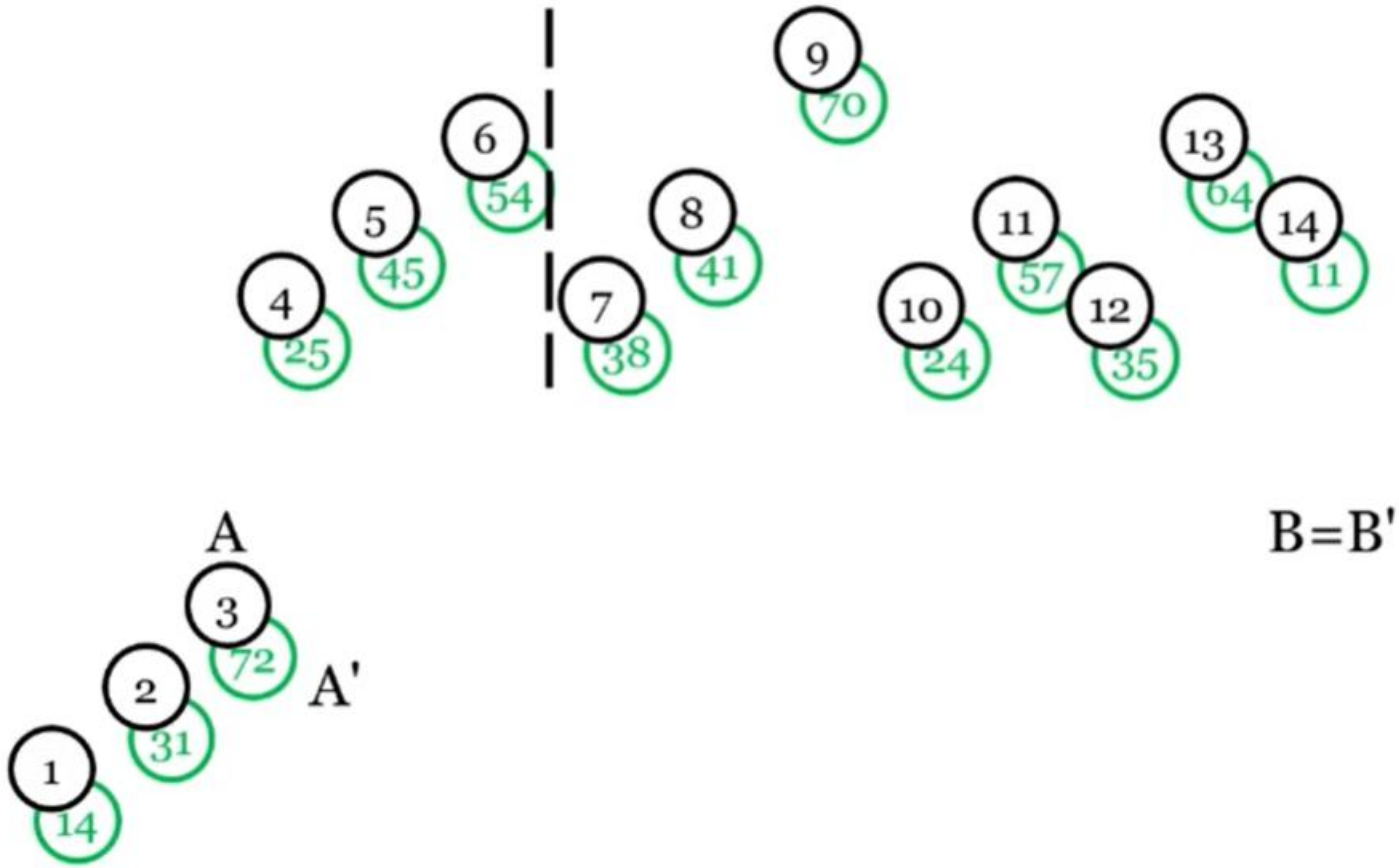
Операция split:

Разделить ДД на два по ключу k, в первом все ключи меньше k, во втором – все остальные значения

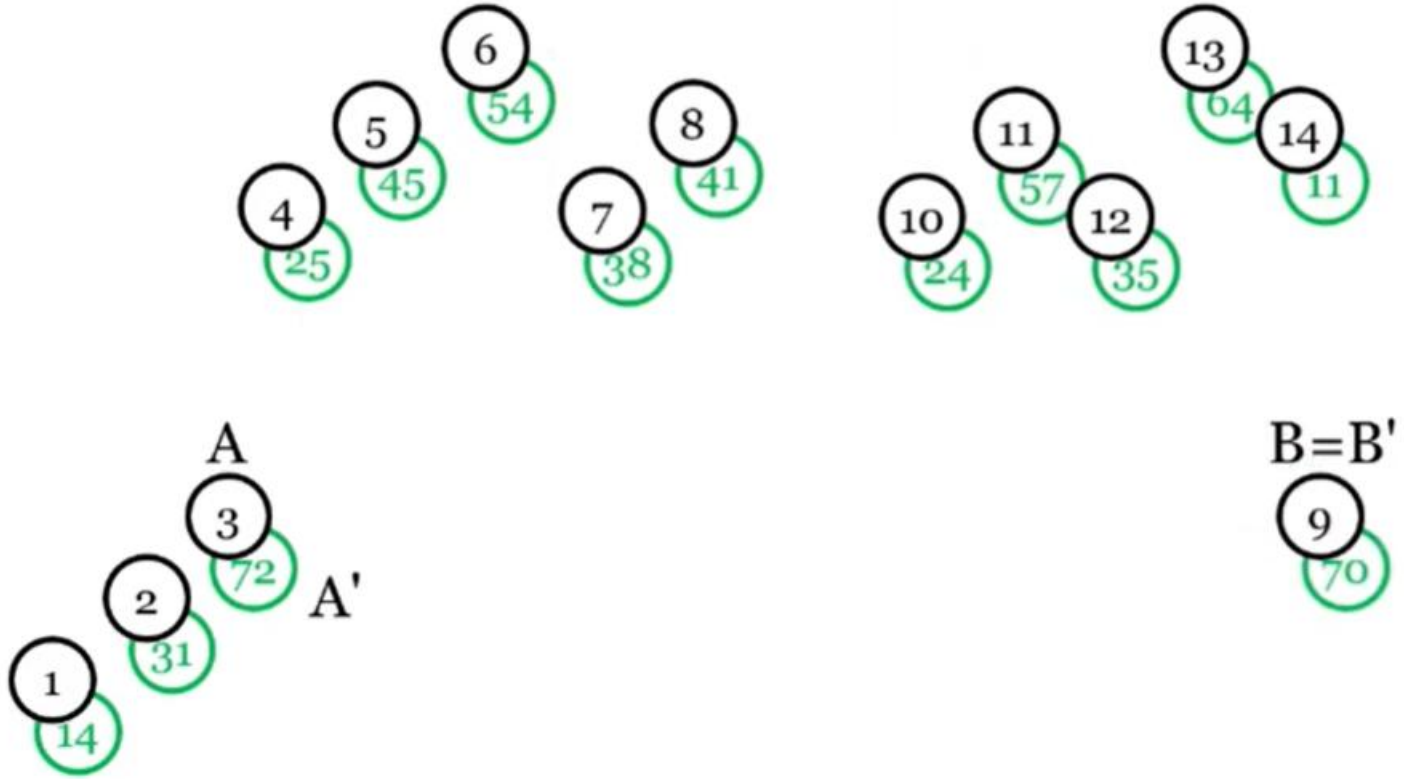
Пример разделения дерева по ключу 7



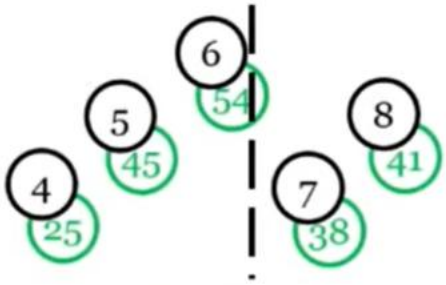
Операция split:
Разделить ДД на два по ключу k, в первом все ключи меньше k, во втором – все остальные значения

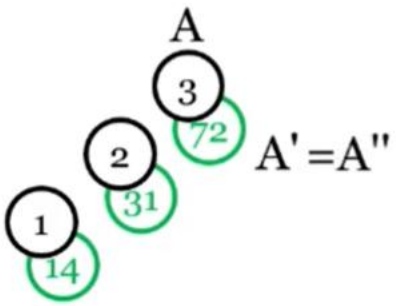


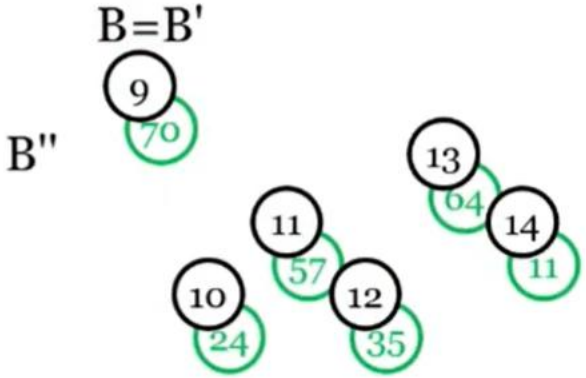
Операция split:
Разделить ДД на два по ключу k, в первом все ключи меньше k, во втором – все остальные значения



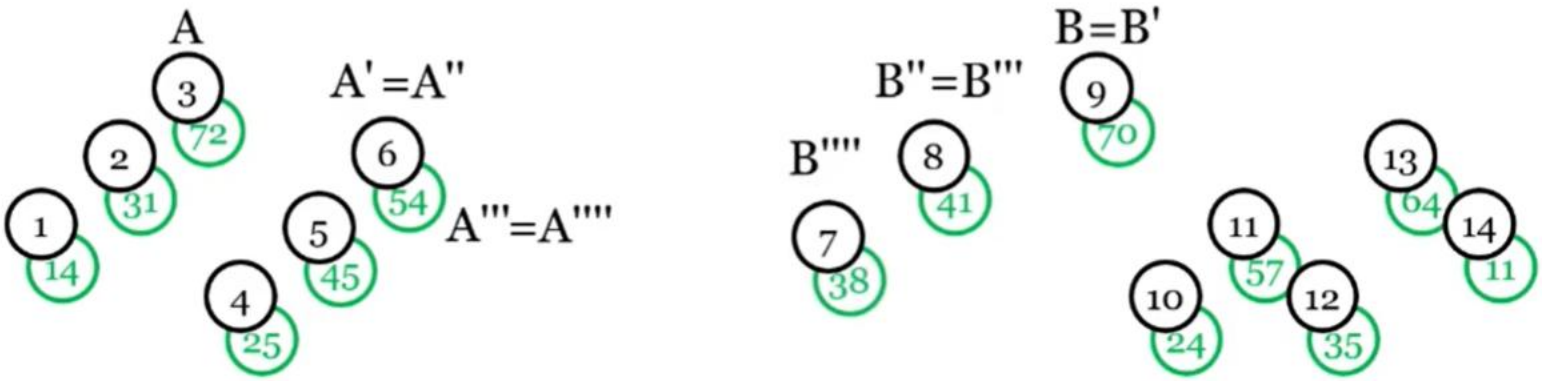
Операция split:
Разделить ДД на два по ключу k, в первом все ключи меньше k, во втором – все остальные значения







Операция split:
Разделить ДД на два по ключу k, в первом все ключи меньше k, во втором – все остальные значения



```
void split(Node *n, int key, Node *&a, Node *&b){
    if (!n){
        a = b = nullptr;
        return ;
    }
    if (n -> key < key){
        //       a = n
        // n->l          a' b'
        split(n -> r, key, n -> r, b);
        a = n;
    }
    else {
        split(n -> l, key, a, n -> l);
        b = n;
    }
}
```

```
public:

    bool find(int key){
        Node *greater, *equal, *less;
        split(root, key, less, greater);
        split(greater, key + 1, equal, greater);
        bool result = equal;
        root = merge(merge(less, equal), greater);
        return result;
    }

    void insert(int key){
        Node *greater, *less;
        split(root, key, less, greater);
        less = merge(less, new Node(key));
        root = merge(less, greater);
    }

    void erase(int key){
        Node *greater, *equal, *less;
        split(root, key, less, greater);
        split(greater, key + 1, equal, greater);
        root = merge(less, greater);
    }
```

Реализуйте структуру данных, которая поддерживает множество S целых чисел, с котором разрешается производить следующие операции:

- $add(i)$ — добавить в множество S число i (если он там уже есть, то множество не меняется);
- $next(i)$ — вывести минимальный элемент множества, не меньший i. Если искомый элемент в структуре отсутствует, необходимо вывести -1.

Входные данные

Исходно множество S пусто. Первая строка входного файла содержит n — количество операций ($1 \le n \le 300\,000$). Следующие n строк содержат операции. Каждая операция имеет вид либо «+ i», либо «? i». Операция «? i» задает запрос $next(i)$.

Если операция «+ i» идет во входном файле в начале или после другой операции «+», то она задает операцию $add(i)$. Если же она идет после запроса «?», и результат этого запроса был y, то выполняется операция $add((i + y) \bmod 10^9)$.

Во всех запросах и операциях добавления параметры лежат в интервале от 0 до 10^9.

Выходные данные

Для каждого запроса выведите одно число — ответ на запрос.

Примеры

входные данные
6 + 1 + 3 + 3 ? 2 + 1 ? 4
выходные данные
3 4

```
int min(Node *n) const{
    if (!n)
        return -1;
    while (n->l){
        n = n->l;
    }
    return n->key;
}

int next(int key){
    Node *greater, *less;
    split(root, key, less, greater);
    int ans = min(greater);
    root = merge(less, greater);
    return ans;
}
```

Добавим возможность находить количество ключей в диапазоне [l; r]

Добавим возможность находить количество ключей в диапазоне [l; r]

```
vector <int> a;

...

sort(a.begin(), a.end());

...

from = lower_bound(a.begin(), a.end(), l);
to = upper_bound(a.begin(), a.end(), r);
cout << to - from;
```

Добавим возможность находить количество ключей в диапазоне [l; r]

```
vector <int> a;

...

sort(a.begin(), a.end());

...

from = lower_bound(a.begin(), a.end(), l);
to = upper_bound(a.begin(), a.end(), r);
cout << to - from;
```

```
set <int> s;

...

from = s.lower_bound(l);
to = s.upper_bound(r);
cout << distance(from, to);
```

Добавим возможность находить количество ключей в диапазоне [l; r]

```
vector <int> a;

...

sort(a.begin(), a.end());

...

from = lower_bound(a.begin(), a.end(), l);
to = upper_bound(a.begin(), a.end(), r);
cout << to - from;
```

```
set <int> s;

...

from = s.lower_bound(l);
to = s.upper_bound(r);
cout << distance(from, to);
```

```
int keys_count(int l, int r){
    Node *left, *middle, *right;
    split(root, l, left, middle);
    split(middle, r + 1, middle, right);
    int ans = get_size(middle);
    root = merge(merge(left, middle), right);
    return ans;
}
```

Добавим возможность находить количество ключей в диапазоне [l; r]

```
vector <int> a;

...

sort(a.begin(), a.end());

...

from = lower_bound(a.begin(), a.end(), l);
to = upper_bound(a.begin(), a.end(), r);
cout << to - from;
```

```
set <int> s;

...

from = s.lower_bound(l);
to = s.upper_bound(r);
cout << distance(from, to);
```

```
int get_size(Node *n){
    if (!n){
        return 0;
    }
    return 1 + get_size(n -> l) + get_size(n -> r);
}
```

```
int keys_count(int l, int r){
    Node *left, *middle, *right;
    split(root, l, left, middle);
    split(middle, r + 1, middle, right);
    int ans = get_size(middle);
    root = merge(merge(left, middle), right);
    return ans;
}
```

Добавим возможность находить количество ключей в диапазоне [l; r]

```
vector <int> a;

...

sort(a.begin(), a.end());

...

from = lower_bound(a.begin(), a.end(), l);
to = upper_bound(a.begin(), a.end(), r);
cout << to - from;
```

```
set <int> s;

...

from = s.lower_bound(l);
to = s.upper_bound(r);
cout << distance(from, to);
```

```
int keys_count(int l, int r){
    Node *left, *middle, *right;
    split(root, l, left, middle);
    split(middle, r + 1, middle, right);
    int ans = get_size(middle);
    root = merge(merge(left, middle), right);
    return ans;
}

int get_size(Node *n){
    return n ? n -> size: 0;
}
```

Рассмотрим 2 задачи:

1. Найти количество ключей < k (найти «номер» ключа k)
2. Найти i-й по возрастанию ключ (узнать ключ по «номеру» i)

Рассмотрим 2 задачи:

1. Найти количество ключей < k (найти «номер» ключа k)
2. Найти i-й по возрастанию ключ (узнать ключ по «номеру» i)

Задача 2 – задача о поиске порядковой статистики

Рассмотрим 2 задачи:

1. Найти количество ключей < k (найти «номер» ключа k)
2. Найти i-й по возрастанию ключ (узнать ключ по «номеру» i)

Задача 2 – задача о поиске порядковой статистики

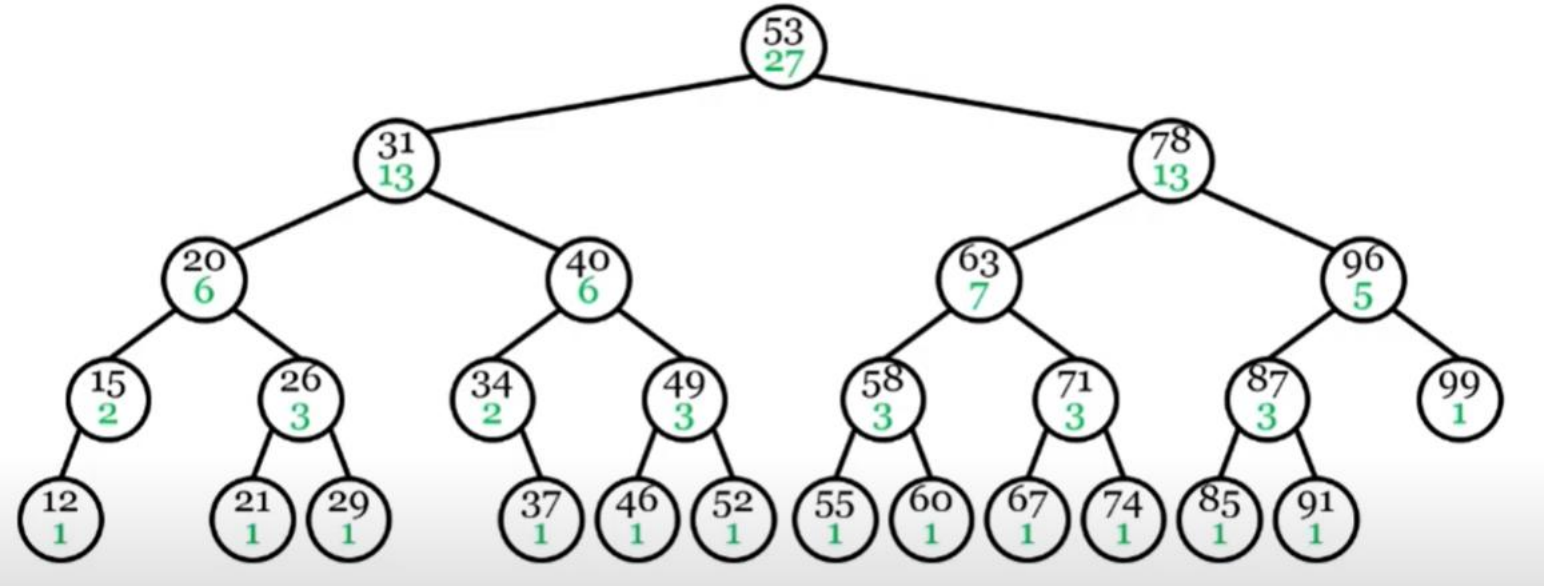
Операция есть в STL для vector (nth_element)

Рассмотрим 2 задачи:

1. Найти количество ключей < k (найти «номер» ключа k)
2. Найти i-й по возрастанию ключ (узнать ключ по «номеру» i)

Задача 2 – задача о поиске порядковой статистики

Операция есть в STL для vector (nth_element)

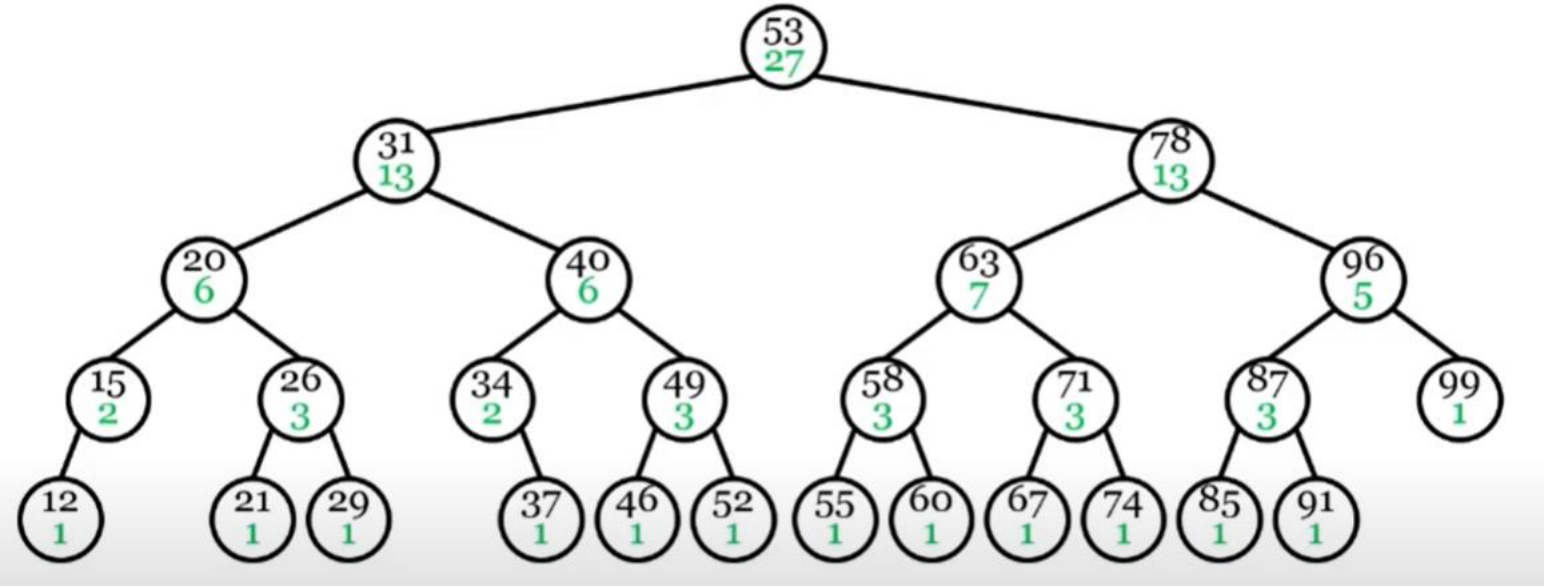


Рассмотрим 2 задачи:

1. Найти количество ключей < k (найти «номер» ключа k)
2. Найти i-й по возрастанию ключ (узнать ключ по «номеру» i)

Задача 2 – задача о поиске порядковой статистики

Операция есть в STL для vector (nth_element)



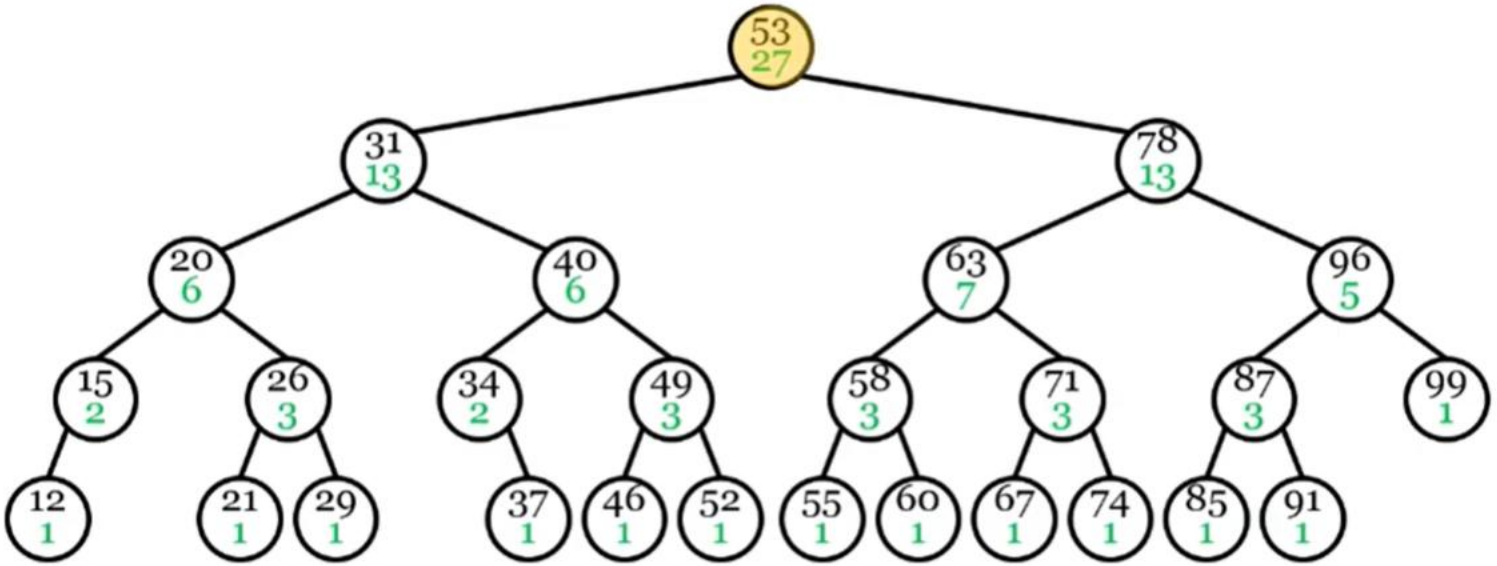
Найдем индекс ключа 49

Рассмотрим 2 задачи:

1. Найти количество ключей < k (найти «номер» ключа k)
2. Найти i-й по возрастанию ключ (узнать ключ по «номеру» i)

Задача 2 – задача о поиске порядковой статистики

Операция есть в STL для vector (nth_element)



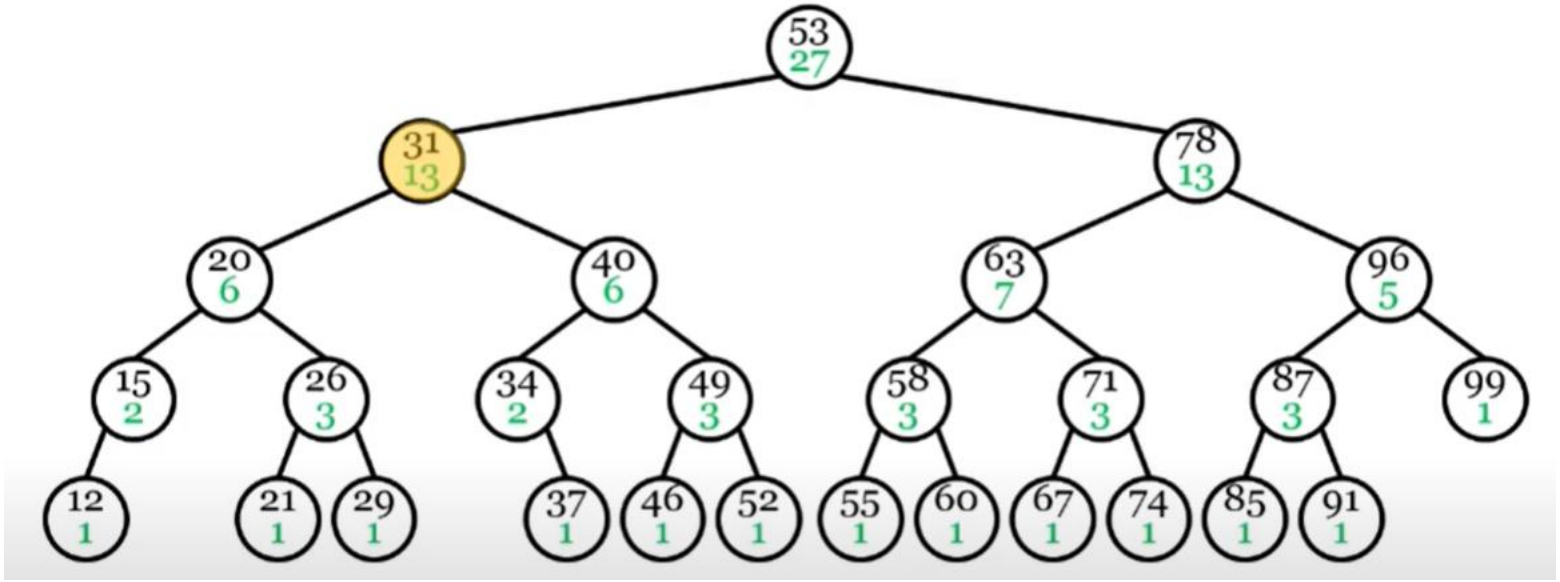
Найдем индекс ключа 49

Рассмотрим 2 задачи:

1. Найти количество ключей < k (найти «номер» ключа k)
2. Найти i-й по возрастанию ключ (узнать ключ по «номеру» i)

Задача 2 – задача о поиске порядковой статистики

Операция есть в STL для vector (nth_element)



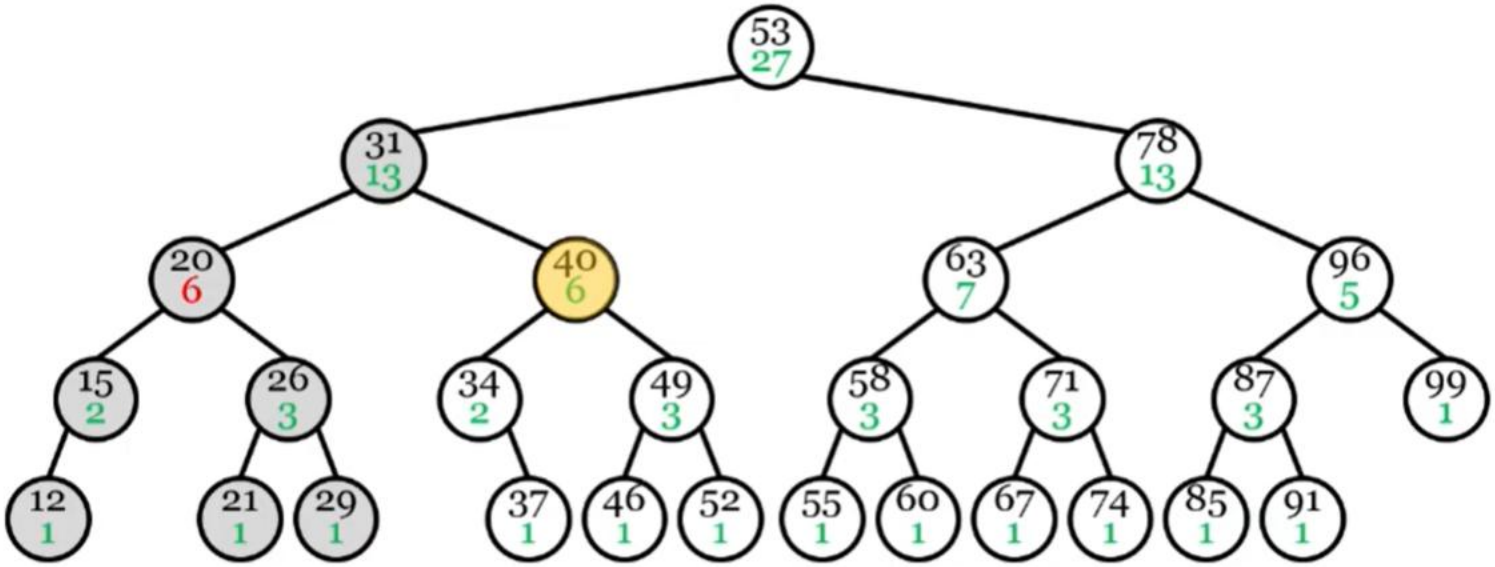
Найдем индекс ключа 49

Рассмотрим 2 задачи:

1. Найти количество ключей < k (найти «номер» ключа k)
2. Найти i-й по возрастанию ключ (узнать ключ по «номеру» i)

Задача 2 – задача о поиске порядковой статистики

Операция есть в STL для vector (nth_element)



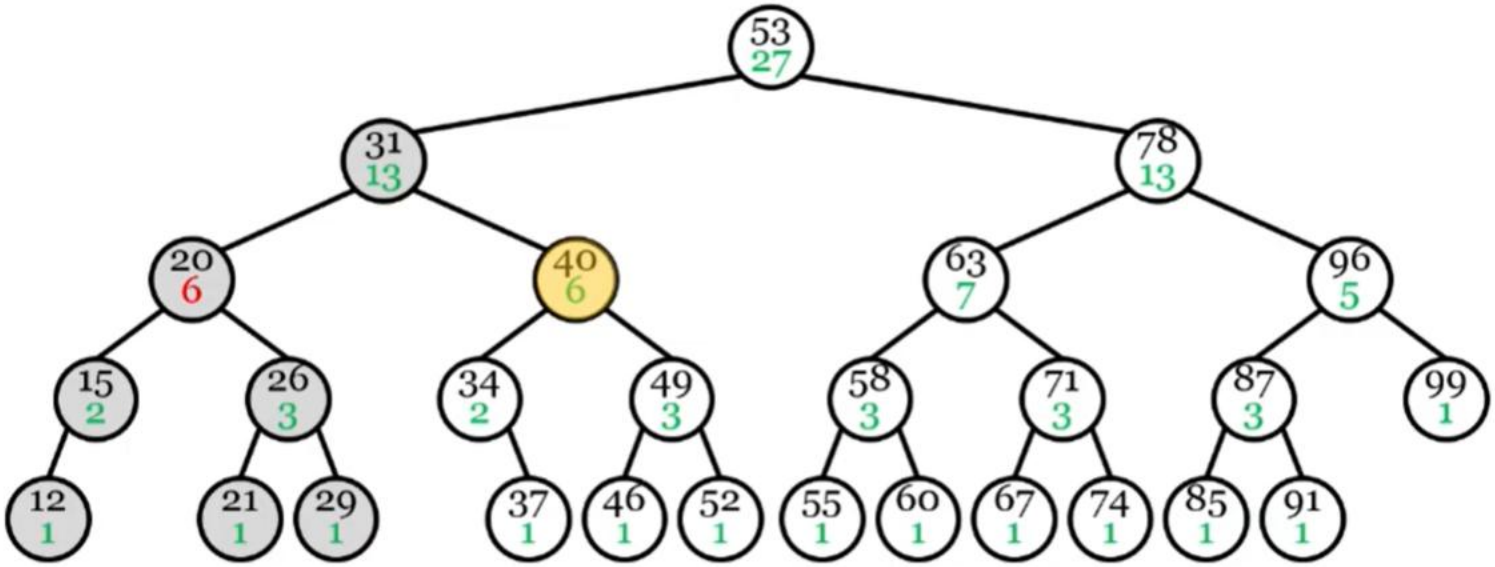
Ответ: 6 + 1

Рассмотрим 2 задачи:

1. Найти количество ключей < k (найти «номер» ключа k)
2. Найти i-й по возрастанию ключ (узнать ключ по «номеру» i)

Задача 2 – задача о поиске порядковой статистики

Операция есть в STL для vector (nth_element)



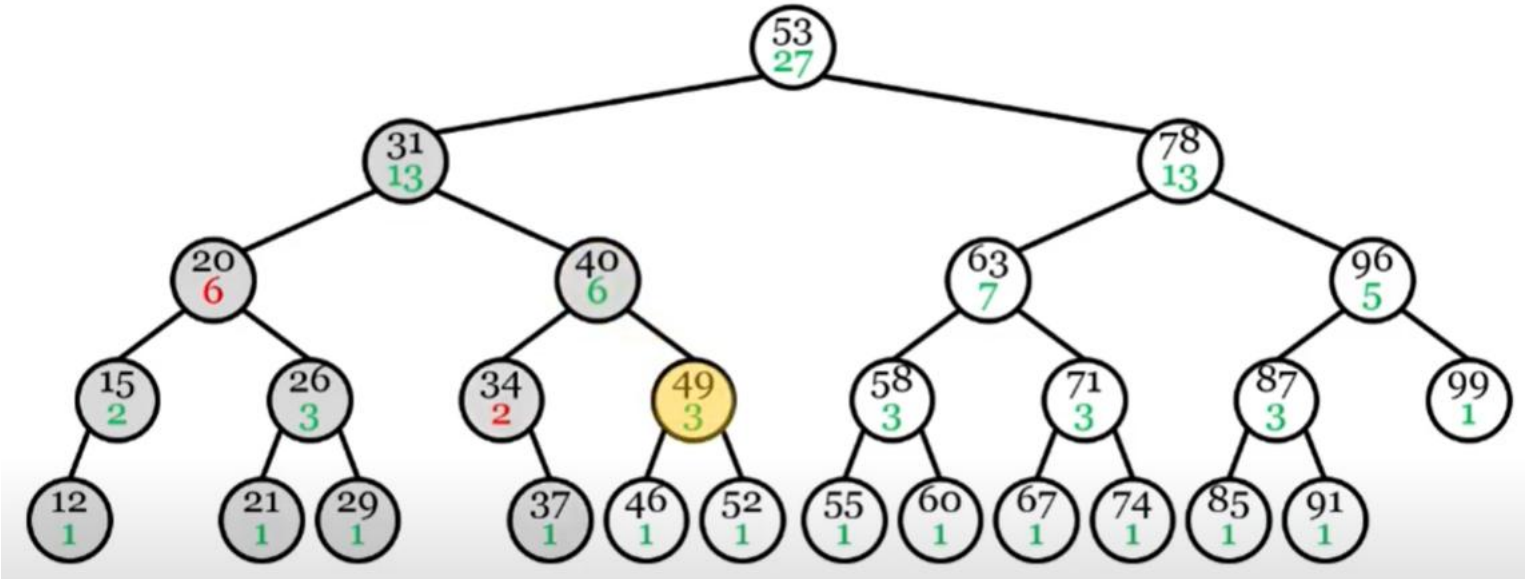
Ответ: 6 + 1

Рассмотрим 2 задачи:

1. Найти количество ключей < k (найти «номер» ключа k)
2. Найти i-й по возрастанию ключ (узнать ключ по «номеру» i)

Задача 2 – задача о поиске порядковой статистики

Операция есть в STL для vector (nth_element)



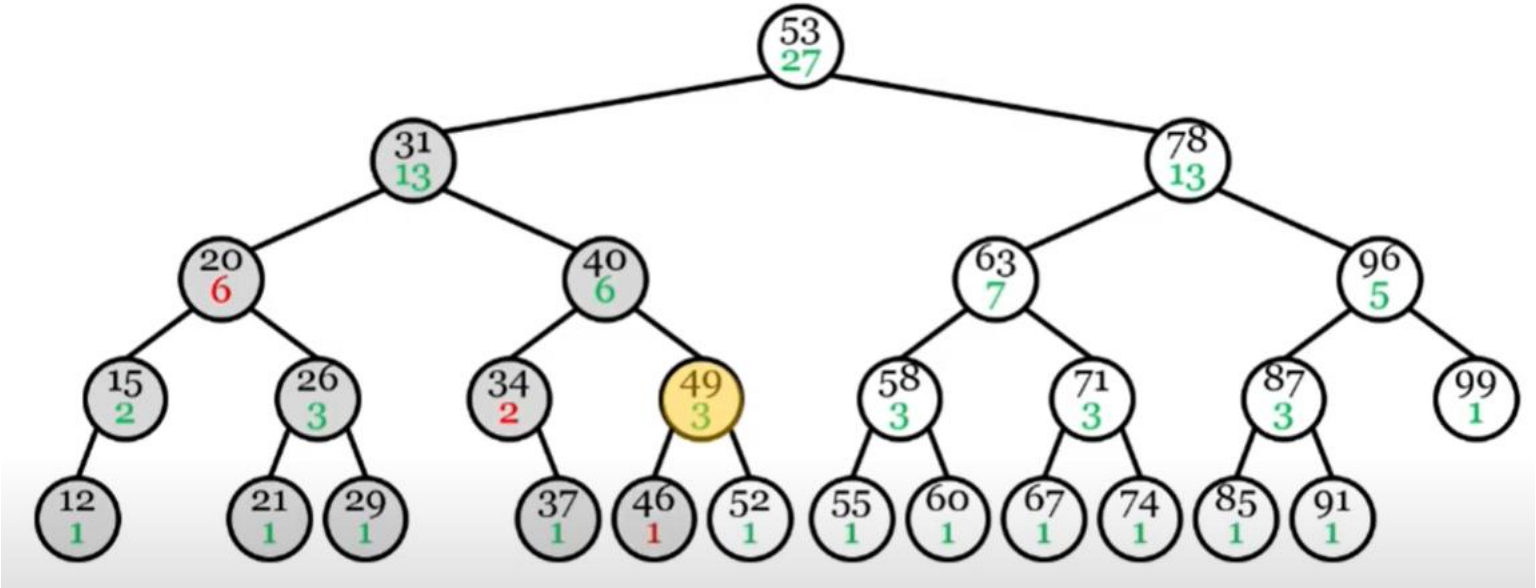
Ответ: (6 + 1) + (2 + 1)

Рассмотрим 2 задачи:

1. Найти количество ключей < k (найти «номер» ключа k)
2. Найти i-й по возрастанию ключ (узнать ключ по «номеру» i)

Задача 2 – задача о поиске порядковой статистики

Операция есть в STL для vector (nth_element)



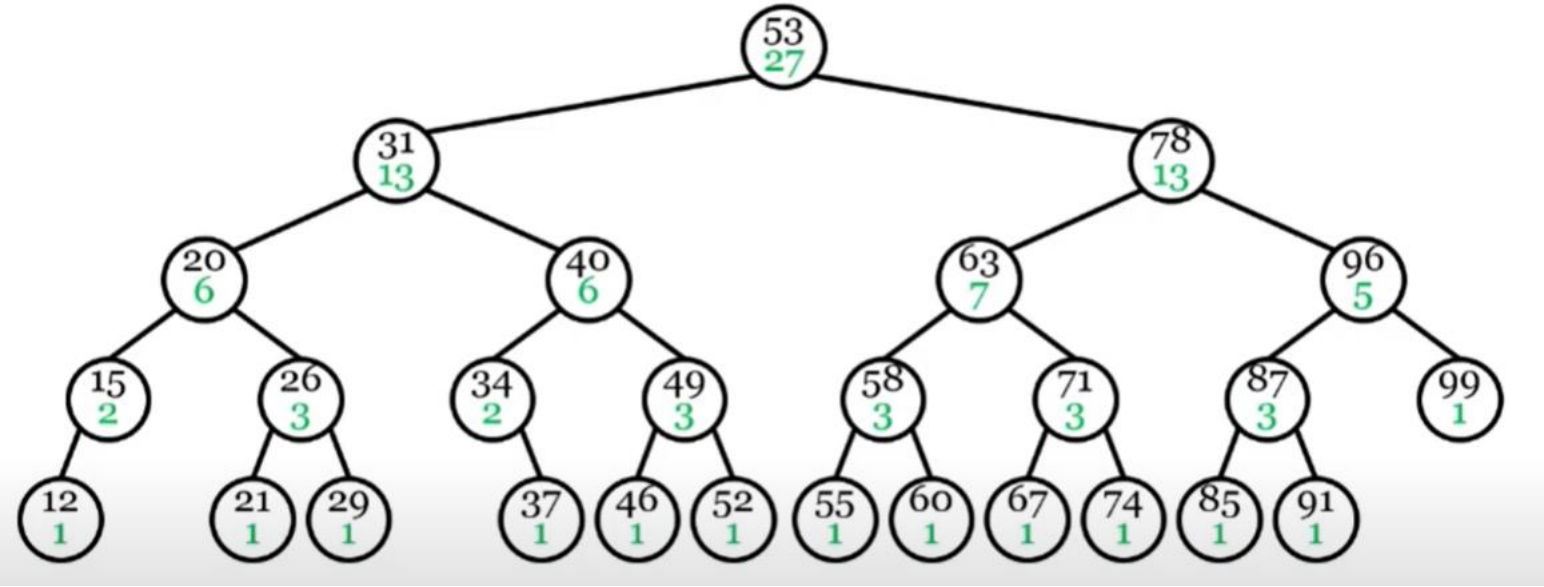
Ответ: (6 + 1) + (2 + 1) + 1 + 1 = 11

Рассмотрим 2 задачи:

1. Найти количество ключей < k (найти «номер» ключа k)
2. Найти i-й по возрастанию ключ (узнать ключ по «номеру» i)

Задача 2 – задача о поиске порядковой статистики

Операция есть в STL для vector (nth_element)



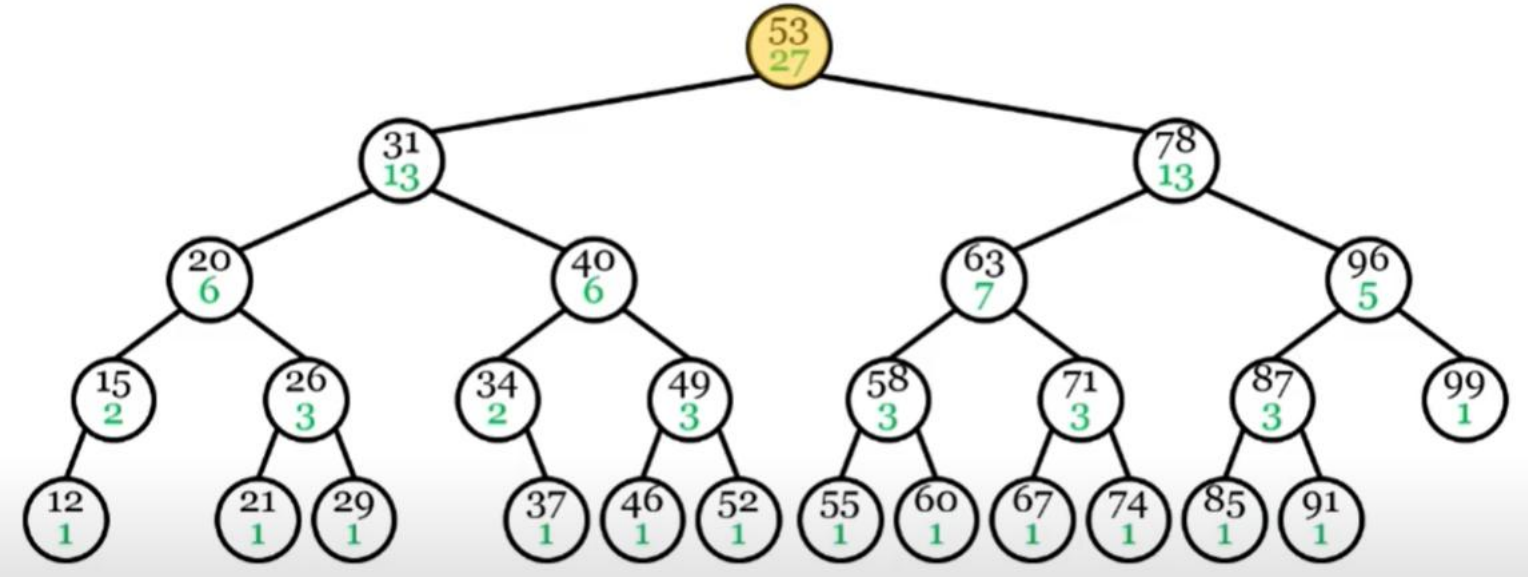
Найдем индекс ключа 95

Рассмотрим 2 задачи:

1. Найти количество ключей < k (найти «номер» ключа k)
2. Найти i-й по возрастанию ключ (узнать ключ по «номеру» i)

Задача 2 – задача о поиске порядковой статистики

Операция есть в STL для vector (nth_element)



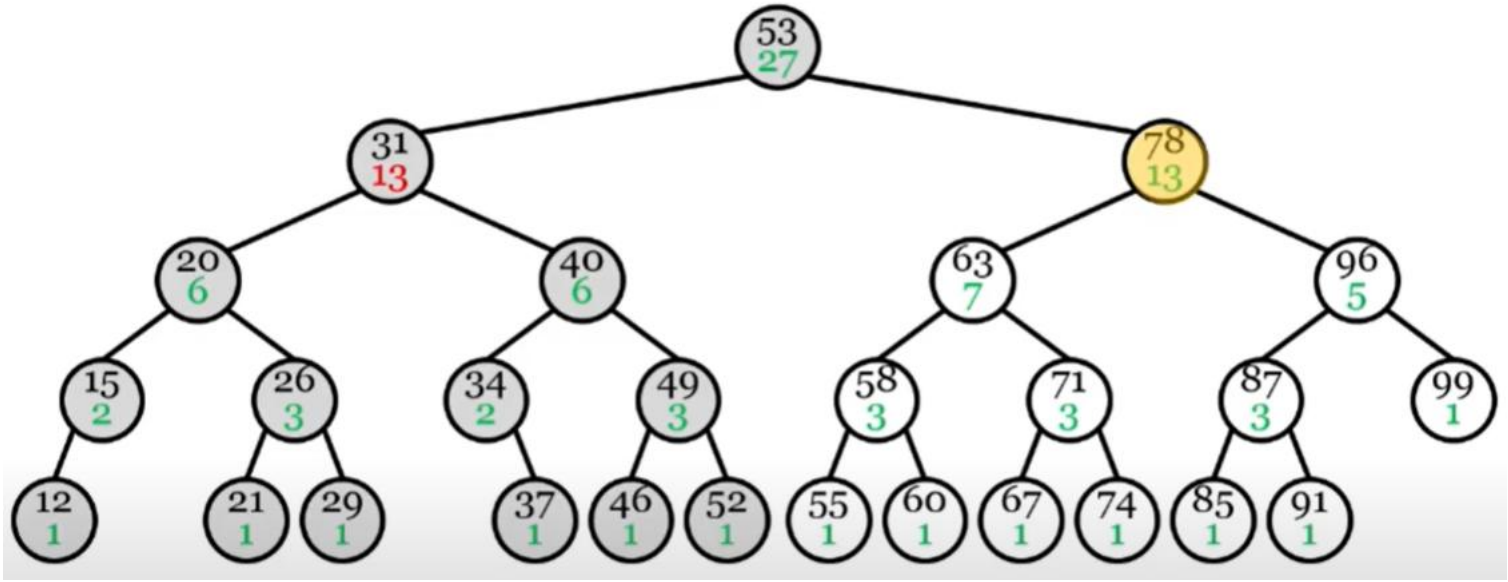
Ответ:

Рассмотрим 2 задачи:

1. Найти количество ключей < k (найти «номер» ключа k)
2. Найти i-й по возрастанию ключ (узнать ключ по «номеру» i)

Задача 2 – задача о поиске порядковой статистики

Операция есть в STL для vector (nth_element)



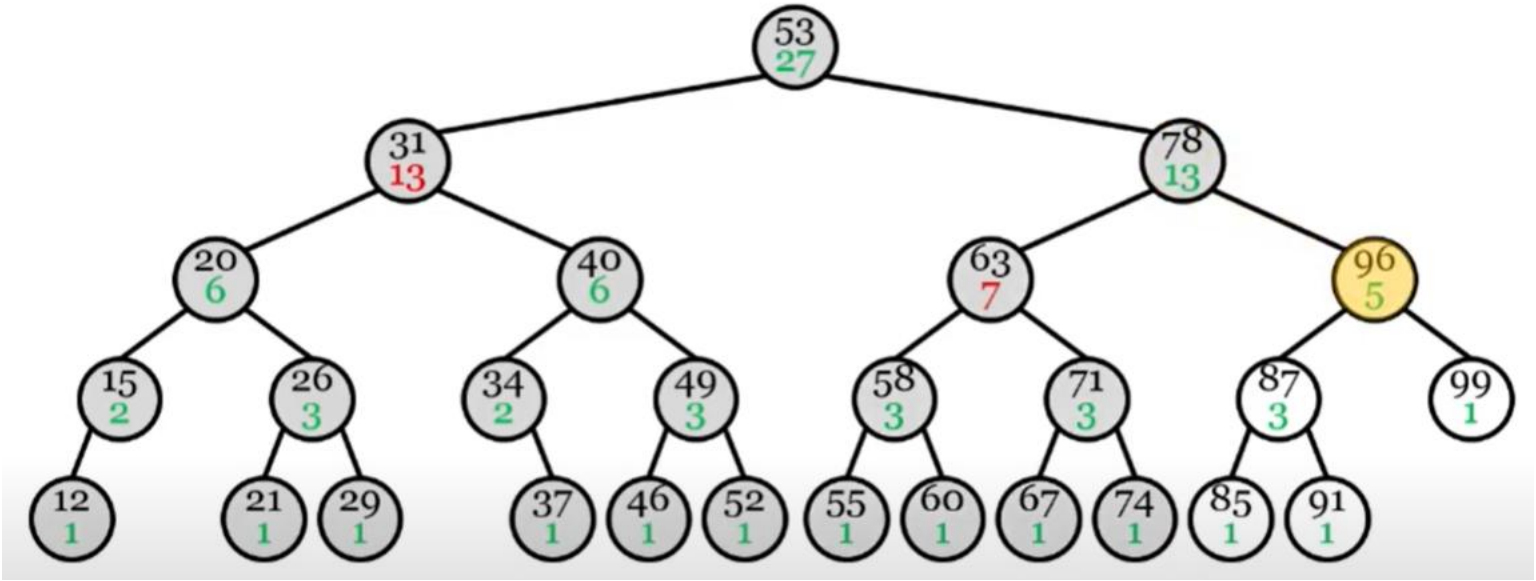
Ответ: 13 + 1

Рассмотрим 2 задачи:

1. Найти количество ключей < k (найти «номер» ключа k)
2. Найти i-й по возрастанию ключ (узнать ключ по «номеру» i)

Задача 2 – задача о поиске порядковой статистики

Операция есть в STL для vector (nth_element)



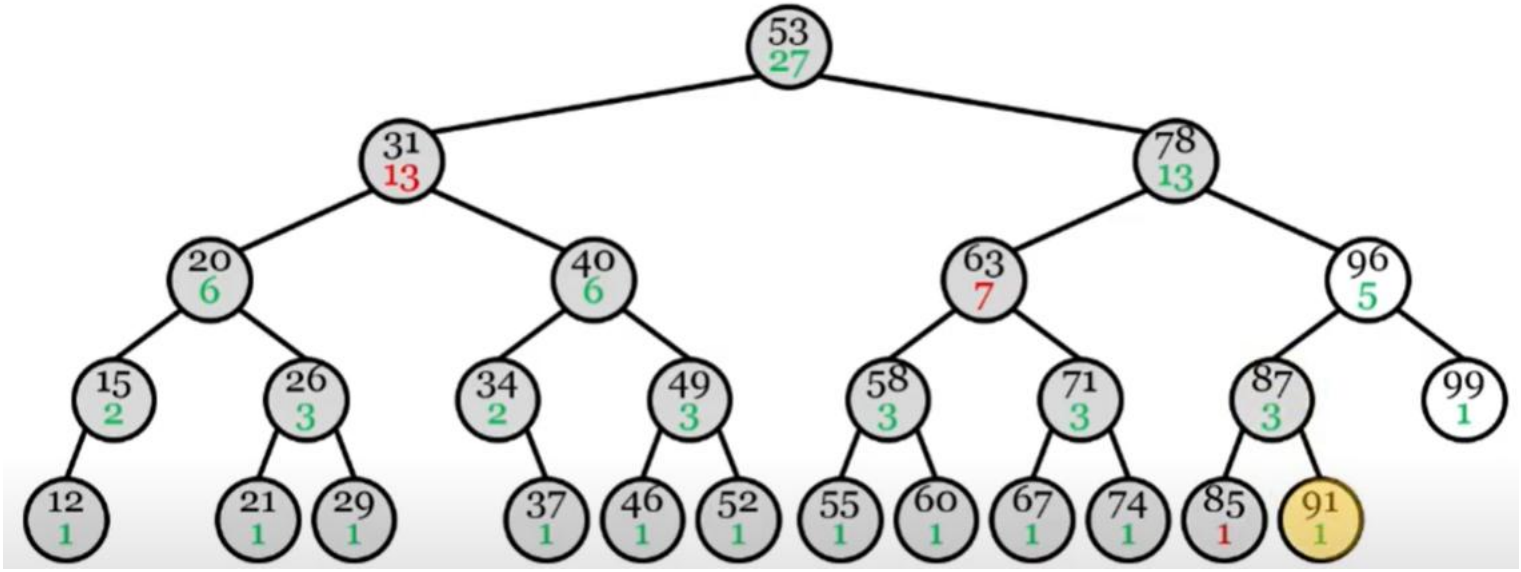
Ответ: (13 + 1) + (7 + 1)

Рассмотрим 2 задачи:

1. Найти количество ключей < k (найти «номер» ключа k)
2. Найти i-й по возрастанию ключ (узнать ключ по «номеру» i)

Задача 2 – задача о поиске порядковой статистики

Операция есть в STL для vector (nth_element)



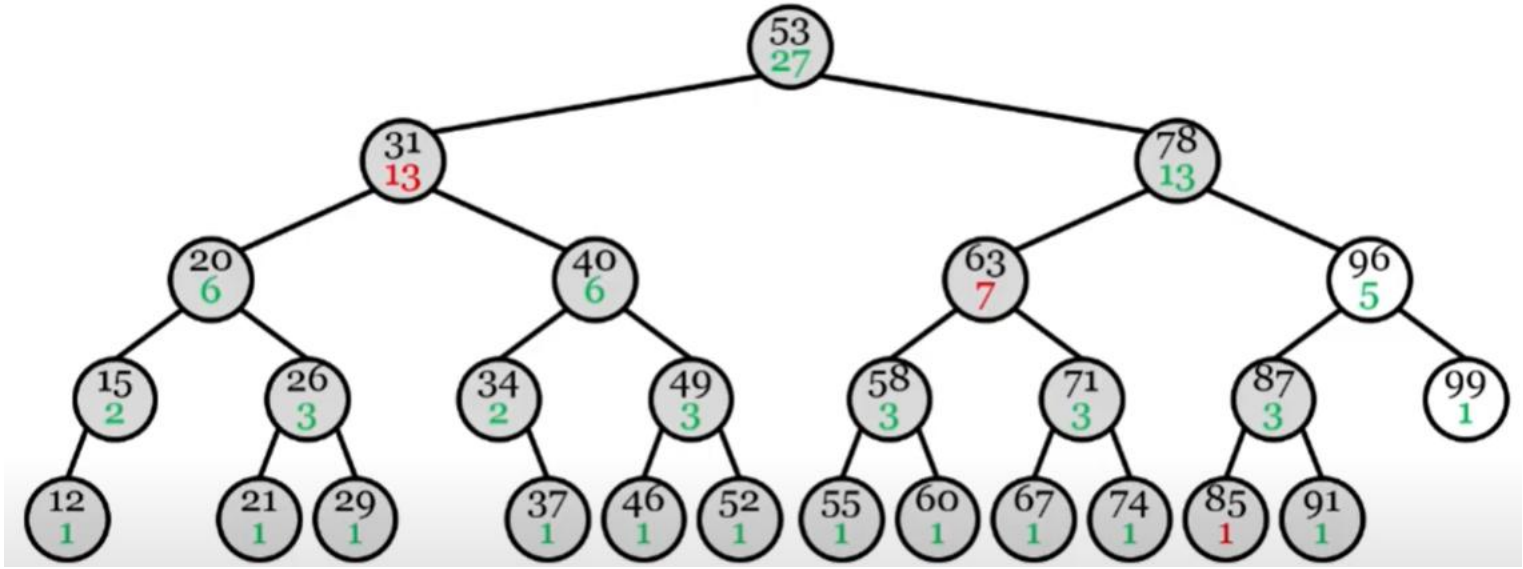
Ответ: (13 + 1) + (7 + 1) + (1 + 1)

Рассмотрим 2 задачи:

1. Найти количество ключей < k (найти «номер» ключа k)
2. Найти i-й по возрастанию ключ (узнать ключ по «номеру» i)

Задача 2 – задача о поиске порядковой статистики

Операция есть в STL для vector (nth_element)

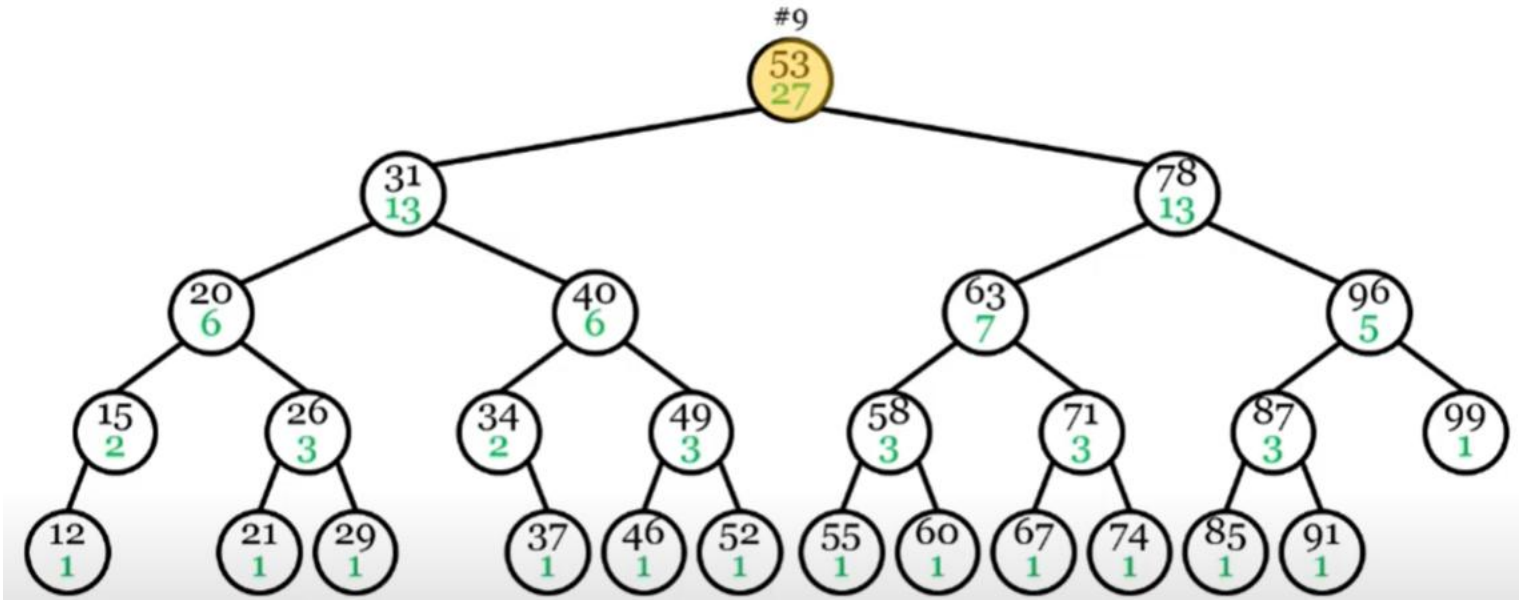


Ответ: (13 + 1) + (7 + 1) + (1 + 1) + (0 + 1) = 25

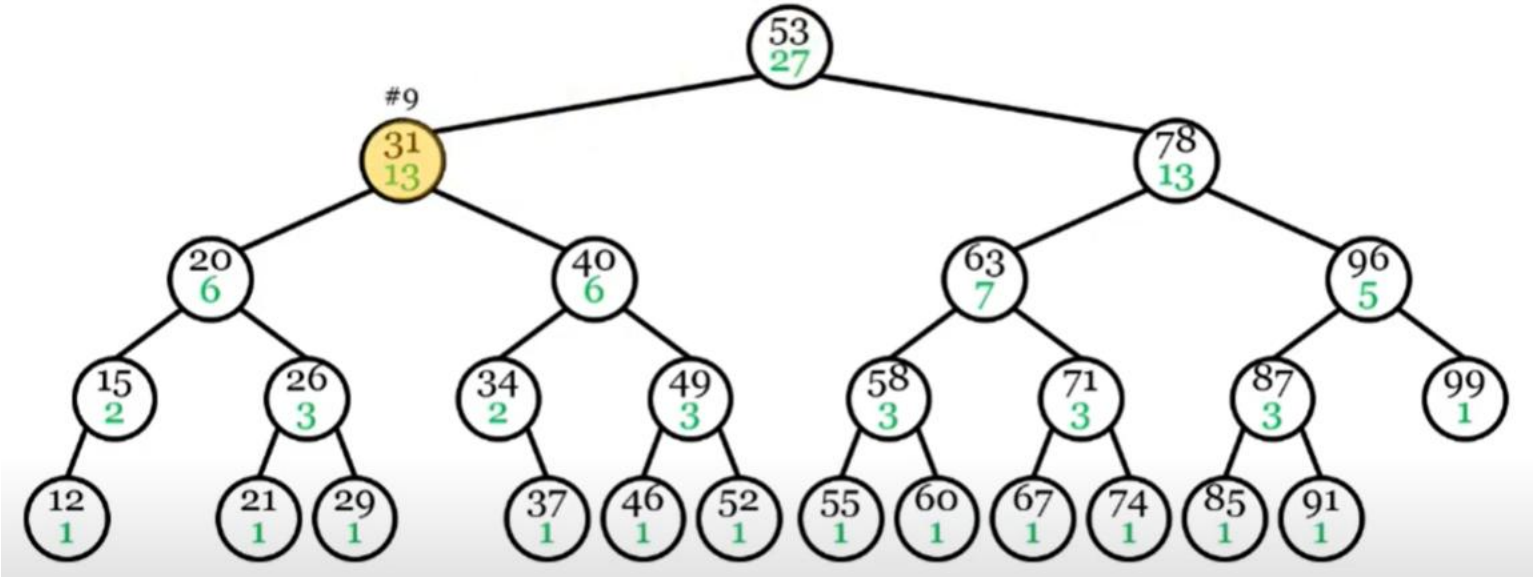
Алгоритм поиска ключа:

1. Ищем ключ стандартно
2. При каждом смещении добавляем к ответу «размер левого поддерева» + 1
3. В конце добавляем к ответу размер левого потомка (если он есть)

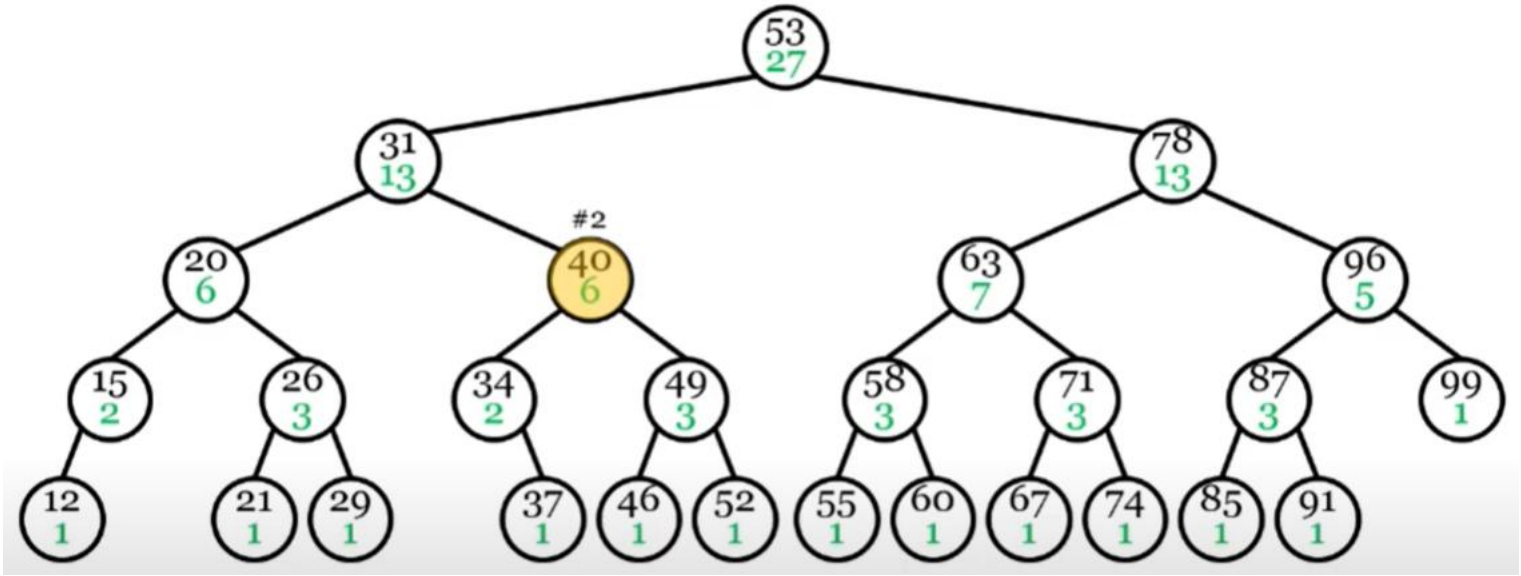
Найти ключ с индексом 9



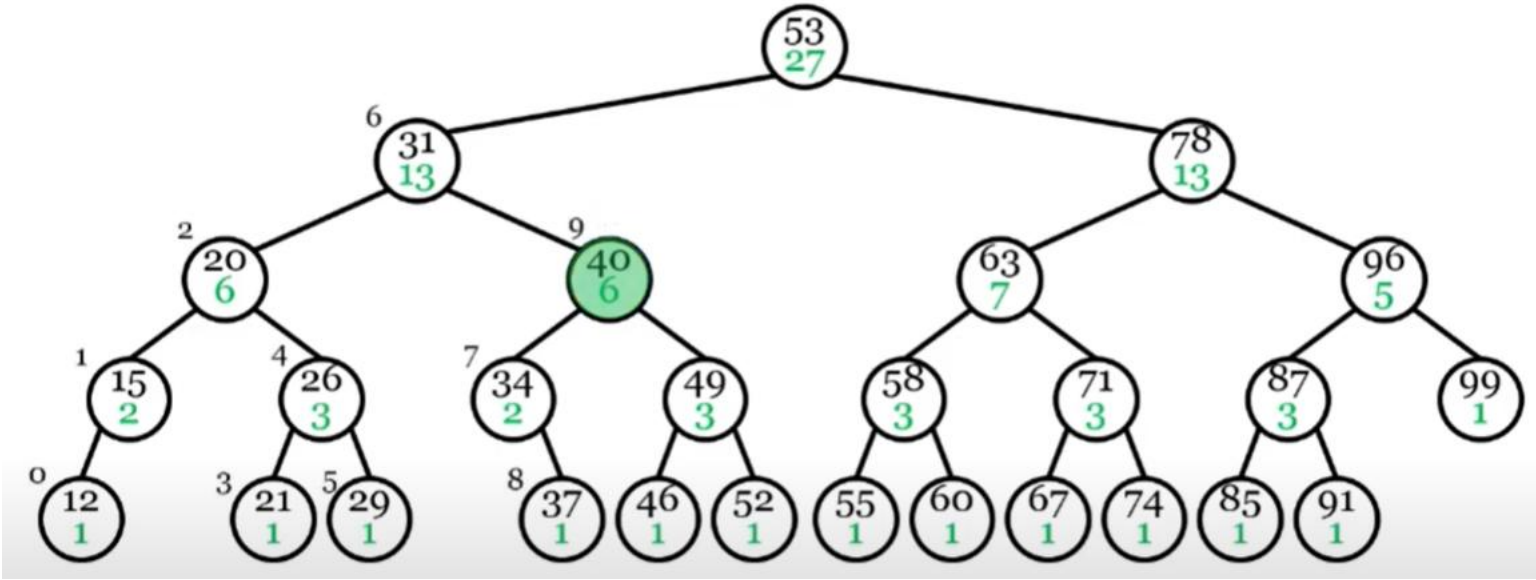
Найти ключ с индексом 9



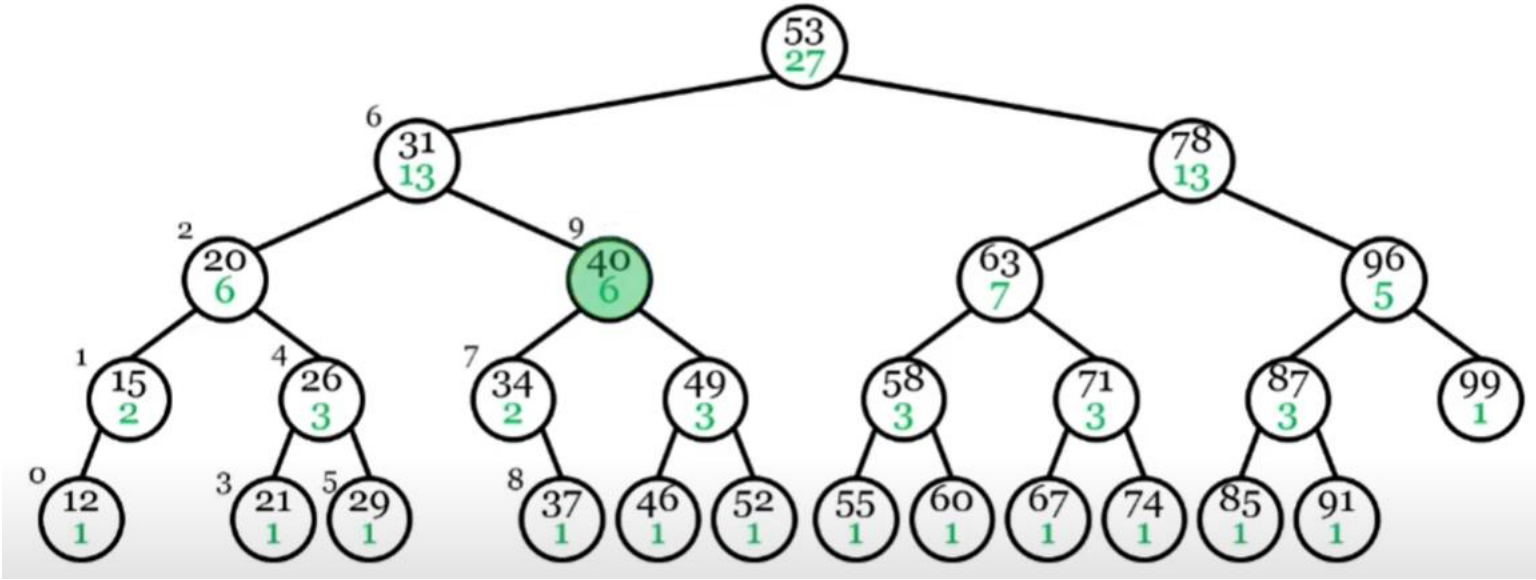
Найти ключ с индексом 9



Найти ключ с индексом 9



Найти ключ с индексом 9



Алгоритм поиска ключа с номером i:

1. i == n -> l -> size => ответ – n -> key
2. i < n -> l -> size => ищем в n -> l
3. i > n -> l -> size => ищем индекс i – n -> l -> size – 1 в n -> r

```
struct Node {
    int key, priority, size;
    Node *l = nullptr, *r = nullptr;
    Node (int key): key(key), priority(generator()), size(1) {}
} *root = nullptr;

int get_size(Node *n){
    return n ? n -> size: 0;
}
```

```
struct Node {
    int key, priority, size;
    Node *l = nullptr, *r = nullptr;
    Node (int key): key(key), priority(generator()), size(1) {}
} *root = nullptr;

int get_size(Node *n){
    return n ? n -> size: 0;
}


  static Node *merge(Node *a, Node *b){
      if (!a || !b){
          return a ? a : b;
      }
      if (a->priority > b->priority){
          a->r = merge(a->r, b);
          return a;
      }
      else {
          b->l = merge(a, b->l);
          return b;
      }
  }
```

```
struct Node {
    int key, priority, size;
    Node *l = nullptr, *r = nullptr;
    Node (int key): key(key), priority(generator()), size(1) {}
} *root = nullptr;

int get_size(Node *n){
    return n ? n -> size: 0;
}
```

```
static Node *merge(Node *a, Node *b){
    if (!a || !b){
        return a ? a : b;
    }
    if (a->priority > b->priority){
        a->r = merge(a->r, b);
        return a;
    }
    else {
        b->l = merge(a, b->l);
        return b;
    }
}
```

```
static void update(Node *&n){
    n -> size = get_size(n);
}

static Node *merge(Node *a, Node *b){
    if (!a || !b){
        return a ? a : b;
    }
    if (a->priority > b->priority){
        a->r = merge(a->r, b);
        update(a);
        return a;
    }
    else {
        b->l = merge(a, b->l);
        update(b);
        return b;
    }
}
```

```
struct Node {
    int key, priority, size;
    Node *l = nullptr, *r = nullptr;
    Node (int key): key(key), priority(generator()), size(1) {}
} *root = nullptr;

int get_size(Node *n){
    return n ? n -> size: 0;
}
```

```
  static void split(Node *n, int key, Node *&a, Node *&b){
      if (!n){
          a = b = nullptr;
          return ;
      }
      if (n -> key < key){
          //      a = n
          // n->l        a' b'
          split(n->r, key, n->r, b);
          a = n;
      }
      else {
          split(n->l, key, a, n->l);
          b = n;
      }
  }
```

```
struct Node {
    int key, priority, size;
    Node *l = nullptr, *r = nullptr;
    Node (int key): key(key), priority(generator()), size(1) {}
} *root = nullptr;

int get_size(Node *n){
    return n ? n -> size: 0;
}
```

```
static void split(Node *n, int key, Node *&a, Node *&b){
    if (!n){
        a = b = nullptr;
        return ;
    }
    if (n -> key < key){
        //      a = n
        // n->l        a' b'
        split(n->r, key, n->r, b);
        a = n;
    }
    else {
        split(n->l, key, a, n->l);
        b = n;
    }
}
```

```
static void split(Node *n, int key, Node *&a, Node *&b){
    if (!n){
        a = b = nullptr;
        return ;
    }
    if (n -> key < key){
        //      a = n
        // n->l        a' b'
        split(n->r, key, n->r, b);
        a = n;
    }
    else {
        split(n->l, key, a, n->l);
        b = n;
    }
    update(a);
    update(b);
}
```

```
int indexByKey (int key) {
    Node *less, *greater;
    split(root, key, less, greater);
    int result = get_size(less);
    root = merge(less, greater);
    return result;
}
```

```
static void update(Node *&n){
    if (n)
        n -> size = get_size(n);
}
```

```
int keyByIndex(Node *n, int index){
    int leftSize = get_size(n -> l);
    if (index == leftSize){
        return n -> key;
    }
    if (index < leftSize){
        return keyByIndex(n -> l, index);
    }
    return keyByIndex(n -> r, index - leftSize - 1);
}
```

```
int keyByIndex(int index){
    return keyByIndex(root, key);
}
```

```
struct Node {
    int key, priority;
    long long sum;
    Node *l = nullptr, *r = nullptr;
    Node (int key): key(key), priority(generator()), sum(key) {}
} *root = nullptr;

static long long get_sum(Node *n){
    return n ? get_sum(n -> l) + (n -> key) + get_sum(n -> r) : 0;
}

static void update(Node *&n){
    if (n){
        n -> sum = get_sum(n);
    }
}


long long get_sum(long long l, long long r){
    Node *left, *middle, *right;
    split(root, l, left, middle);
    split(middle, r + 1, middle, right);
    long long ans = get_sum(middle);
    root = merge(merge(left, middle), right);
    return ans;
}
```

Постепенно перейдем от ключей к значениям

1. Добавить ключ k со значением v
2. Определить количество значений
3. Определить количество значений с ключами от l до r
4. Все значения увеличить на x
5. Увеличить на x значения с ключами от l до r

```
struct Node {
    int key, priority;
    int value;
    Node *l = nullptr, *r = nullptr;
    Node (int key, int value): key(key), priority(generator()), value(value) {}
} *root = nullptr;


int get(int key){
    Node *greater, *equal, *less;
    split(root, key, less, greater);
    split(greater, key + 1, equal, greater);
    if (!equal){
        ....
    }
    result = equal -> value;
    root = merge(merge(less, equal), greater);
    return result;
}
```

В новых элитных электричках каждому пассажиру положено сидячее место. Естественно, количество сидячих мест ограничено и на всех их может не хватить. Маршрут электрички проходит через N+1 станция, занумерованные от 0 до N. Когда человек хочет купить билет, он называет два числа x и y – номера станций, откуда и куда он хочет ехать. При наличии хотя бы одного сидячего места на этом участке на момент покупки ему продается билет, иначе выдается сообщение «билетов нет» и билет не продается. Ваша задача – написать программу, обслуживающую такого рода запросы в порядке их прихода.

Входные данные

В первой строке содержаться три числа N – количество станций ($1 \le N \le 200\,000$), K – количество мест в электричке ($1 \le K \le 1000$) и M – количество запросов ($1 \le M \le 100\,000$). В следующих M строках описаны запросы, каждый из которых состоит из двух чисел x и y ($0 \le x < y <= N$).

Выходные данные

На каждый запрос ваша программа должна выдавать результат в виде числа 0 если билет не продается и 1 если билет был продан. Каждый результат должен быть на отдельной строке

Примеры

входные данные
5 2 4 0 4 1 2 1 4 2 4
выходные данные
1 1 0 1

Примеры

входные данные
5 2 4 0 4 1 2 1 4 2 4

выходные данные
1 1 0 1

Примеры

входные данные
5 2 4 0 4 1 2 1 4 2 4

выходные данные
1 1 0 1

0	1	2	3	4	5
0	0	0	0	0	0

Примеры

входные данные
5 2 4 0 4 1 2 1 4 2 4

выходные данные
1 1 0 1

0	1	2	3	4	5
0	0	0	0	0	0

0	1	2	3	4	5
1	1	1	1	0	0

Примеры

входные данные
5 2 4 0 4 1 2 1 4 2 4

выходные данные
1 1 0 1

0	1	2	3	4	5
0	0	0	0	0	0

0	1	2	3	4	5
1	1	1	1	0	0

0	1	2	3	4	5
1	2	1	1	0	0

Примеры

входные данные
5 2 4 0 4 1 2 1 4 2 4

выходные данные
1 1 0 1

0	1	2	3	4	5
0	0	0	0	0	0

0	1	2	3	4	5
1	1	1	1	0	0

0	1	2	3	4	5
1	2	1	1	0	0

0	1	2	3	4	5
1	2	2	2	0	0

Примеры

входные данные
5 2 4 0 4 1 2 1 4 2 4
выходные данные
1 1 0 1

0	1	2	3	4	5
0	0	0	0	0	0

0	1	2	3	4	5
1	1	1	1	0	0

0	1	2	3	4	5
1	2	1	1	0	0

0	1	2	3	4	5
1	2	2	2	0	0

1. Найти максимальное значение среди ключей на отрезке [l; r – 1]
2. Увеличить на 1 значение всех ключей на отрезке [l; r – 1]

Ключ – номер остановки, значение – количество занятых мест

```
struct Node {
    int key, priority;
    int value, max_val;
    Node *l = nullptr, *r = nullptr;
    Node (int key, int value): key(key), priority(generator()), value(value), max_val(value) {}
} *root = nullptr;


static int getMaxValue(Node *n){
    return n ? n -> max_val : -1e9;
}

static void update(Node *&n){
    if (n){
        n -> max_val = max(max(getMaxValue(n -> l), n -> value), getMaxValue(n -> r));
    }
}

long long getMax(int l, int r){
    Node *left, *middle, *right;
    split(root, l, left, middle);
    split(middle, r + 1, middle, right);
    long long ans = getMaxValue(middle);
    root = merge(merge(left, middle), right);
    return ans;
}
```

```
struct Node {
    int key, priority;
    int value, max_val, add = 0;
    Node *l = nullptr, *r = nullptr;
    Node (int key, int value): key(key), priority(generator()), value(value), max_val(value) {}
} *root = nullptr;
```

```
struct Node {
    int key, priority;
    int value, max_val, add = 0;
    Node *l = nullptr, *r = nullptr;
    Node (int key, int value): key(key), priority(generator()), value(value), max_val(value) {}
} *root = nullptr;


void rangeAdd(int l, int r, int value){
    Node *left, *middle, *right;
    split(root, l, left, middle);
    split(middle, r + 1, middle, right);
    if (middle){
        middle -> add += value;
    }
    root = merge(merge(left, middle), right);
}
```

```
struct Node {
    int key, priority;
    int value, max_val, add = 0;
    Node *l = nullptr, *r = nullptr;
    Node (int key, int value): key(key), priority(generator()), value(value), max_val(value) {}
} *root = nullptr;
```

```
void rangeAdd(int l, int r, int value){
    Node *left, *middle, *right;
    split(root, l, left, middle);
    split(middle, r + 1, middle, right);
    if (middle){
        middle -> add += value;
    }
    root = merge(merge(left, middle), right);
}
```

```
static int getMaxValue(Node *n){
    return n ? n -> max_val + n -> add : -1e9;
}
```

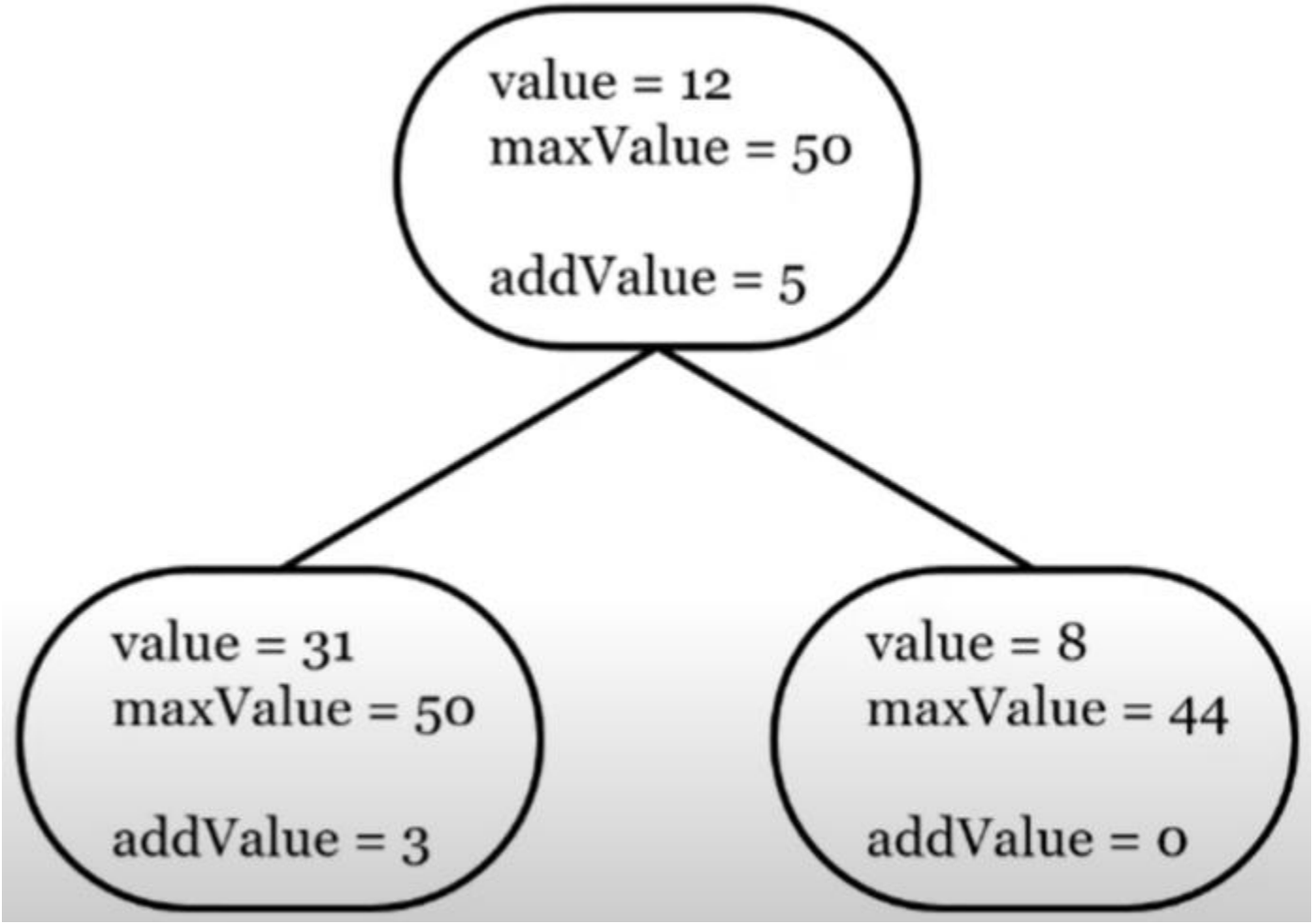
```
struct Node {
    int key, priority;
    int value, max_val, add = 0;
    Node *l = nullptr, *r = nullptr;
    Node (int key, int value): key(key), priority(generator()), value(value), max_val(value) {}
} *root = nullptr;
```

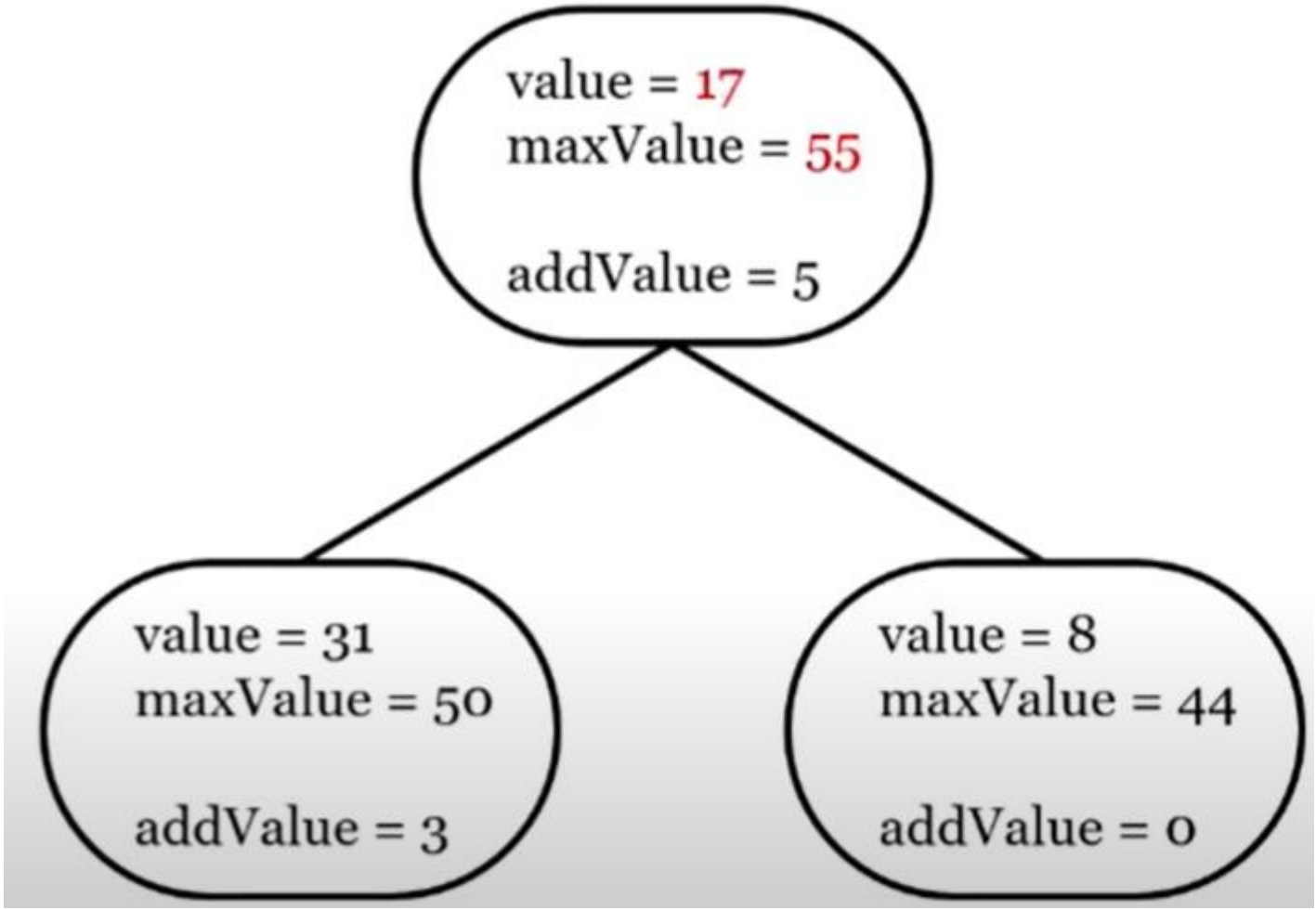
```
void rangeAdd(int l, int r, int value){
    Node *left, *middle, *right;
    split(root, l, left, middle);
    split(middle, r + 1, middle, right);
    if (middle){
        middle -> add += value;
    }
    root = merge(merge(left, middle), right);
}
```

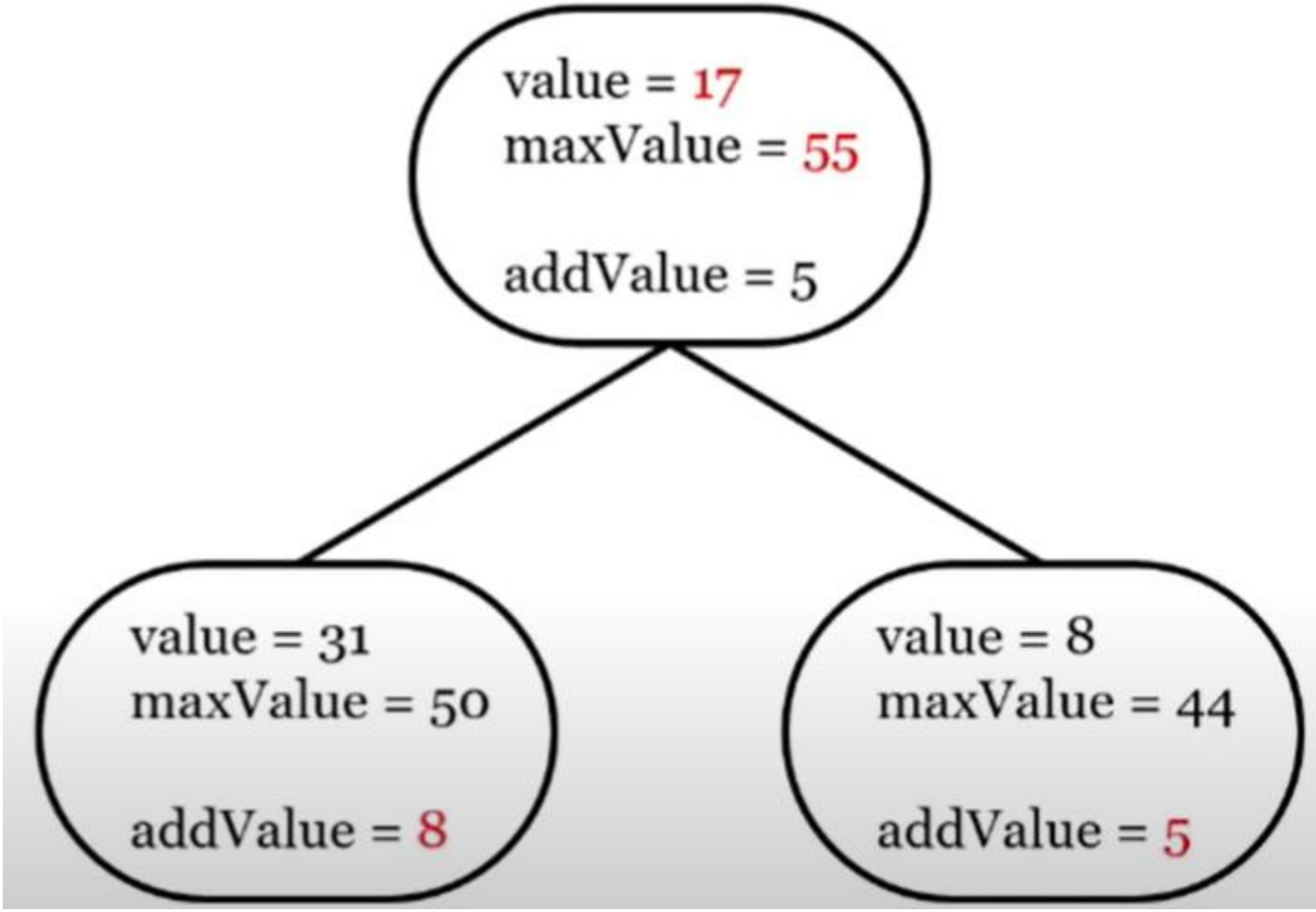
```
static int getMaxValue(Node *n){
    return n ? n -> max_val + n -> add : -1e9;
}
```

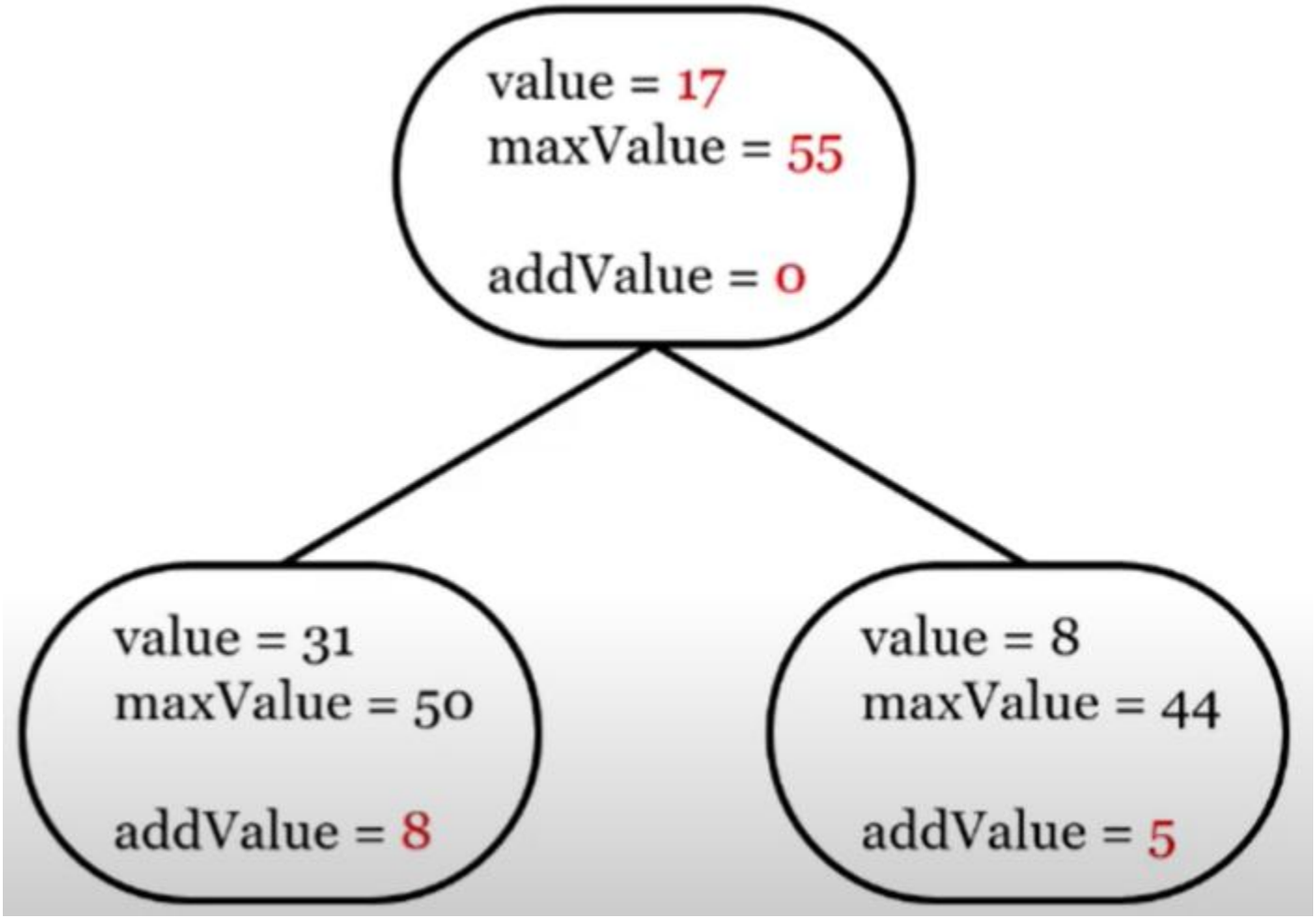
```
static Node *merge(Node *a, Node *b){
    if (!a || !b){
        return a ? a : b;
    }
    if (a->priority > b->priority){
        a->r = merge(a->r, b);
        update(a);
        return a;
    }
    else {
        b->l = merge(a, b->l);
        update(b);
        return b;
    }
}
```

```
static void split(Node *n, int key, Node *&a, Node *&b){
    if (!n){
        a = b = nullptr;
        return ;
    }
    if (n -> key < key){
        //      a = n
        // n->l        a' b'
        split(n->r, key, n->r, b);
        a = n;
    }
    else {
        split(n->l, key, a, n->l);
        b = n;
    }
    update(a);
    update(b);
}
```


value = 12
maxValue = 50
addValue = 5
value = 31
maxValue = 50
addValue = 3
value = 8
maxValue = 44
addValue = 0


value = 17
maxValue = 55
addValue = 5
value = 31
maxValue = 50
addValue = 3
value = 8
maxValue = 44
addValue = 0


value = 17
maxValue = 55
addValue = 5
value = 31
maxValue = 50
addValue = 8
value = 8
maxValue = 44
addValue = 5


value = 17
maxValue = 55
addValue = 0
value = 31
maxValue = 50
addValue = 8
value = 8
maxValue = 44
addValue = 5

```
struct Node {
    int key, priority;
    int value, max_val, add = 0;
    Node *l = nullptr, *r = nullptr;
    Node (int key, int value): key(key), priority(generator()), value(value), max_val(value) {}
} *root = nullptr;
```

```
void rangeAdd(int l, int r, int value){
    Node *left, *middle, *right;
    split(root, l, left, middle);
    split(middle, r + 1, middle, right);
    if (middle){
        middle -> add += value;
    }
    root = merge(merge(left, middle), right);
}
```

```
static void push(Node *n){
    if (n && n -> add){
        n -> value += n -> add;
        n -> max_val += n -> add;
        if (n -> l){
            n -> l -> add += n -> add;
        }
        if (n -> r){
            n -> r -> add += n -> add;
        }
        n -> add = 0;
    }
}
```

```
static int getMaxValue(Node *n){
    return n ? n -> max_val + n -> add : -1e9;
}
```

```
struct Node {
    int key, priority;
    int value, max_val, add = 0;
    Node *l = nullptr, *r = nullptr;
    Node (int key, int value): key(key), priority(generator()), value(value), max_val(value) {}
} *root = nullptr;
```

```
void rangeAdd(int l, int r, int value){
    Node *left, *middle, *right;
    split(root, l, left, middle);
    split(middle, r + 1, middle, right);
    if (middle){
        middle -> add += value;
    }
    root = merge(merge(left, middle), right);
}
```

```
static int getMaxValue(Node *n){
    return n ? n -> max_val + n -> add : -1e9;
}
```

```
struct Node {
    int key, priority;
    int value, max_val, add = 0;
    Node *l = nullptr, *r = nullptr;
    Node (int key, int value): key(key), priority(generator()), value(value), max_val(value) {}
} *root = nullptr;
```

```
void rangeAdd(int l, int r, int value){
    Node *left, *middle, *right;
    split(root, l, left, middle);
    split(middle, r + 1, middle, right);
    if (middle){
        middle -> add += value;
    }
    root = merge(merge(left, middle), right);
}
```

```
static int getMaxValue(Node *n){
    return n ? n -> max_val + n -> add : -1e9;
}
```

```
static Node *merge(Node *a, Node *b){
    push(a);
    push(b);
    if (!a || !b){
        return a ? a : b;
    }
    if (a->priority > b->priority){
        a->r = merge(a->r, b);
        update(a);
        return a;
    }
    else {
        b->l = merge(a, b->l);
        update(b);
        return b;
    }
}
```

```
static void split(Node *n, int key, Node *&a, Node *&b){
    push(n);
    if (!n){
        a = b = nullptr;
        return ;
    }
    if (n -> key < key){
        //      a = n
        // n->l        a' b'
        split(n->r, key, n->r, b);
        a = n;
    }
    else {
        split(n->l, key, a, n->l);
        b = n;
    }
    update(a);
    update(b);
}
```

```
struct Node {
    int key, priority;
    int value, cnt = 1, sum, add = 0;
    Node *l = nullptr, *r = nullptr;
    Node (int key, int value): key(key), priority(generator()), value(value), sum(value) {}
} *root = nullptr;

static long long getSum(Node *n){
    return n ? n -> sum + n -> add * 1ll * n -> cnt : 0;
}

static long long getCnt(Node *n){
    return n ? n -> cnt : 0;
}

static void update(Node *&n){
    if (n){
       n -> sum = getSum(n -> l) + n -> value + getSum(n -> r);
       n -> cnt = getCnt(n -> l) + 1 + getCnt(n -> r);
    }
}
```