

Лекция 9

Дерево ван Эмде Боаса (van Emde Boas tree)

Курносов Михаил Георгиевич

E-mail: mkurnosov@gmail.com

WWW: www.mkurnosov.net

Курс «Алгоритмы и структуры данных»

Сибирский государственный университет телекоммуникаций и информатики (Новосибирск)

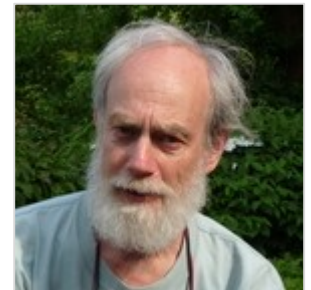
Осенний семестр, 2015

Дерево ван Эмде Боаса (Van Emde Boas tree)

- **Дерево ван Эмде Боаса (van Emde Boas tree, vEB tree)** – это структура данных, реализующая операции Lookup, Insert, Delete, Min, Max, Successor, Predecessor динамического множества за время $O(\log(\log(n)))$
- Операции ExtractMin и DecreaseKey могут быть реализованы на базе операций Min, Delete и Insert

Дерево ван Эмде Боаса (Van Emde Boas tree)

- **Дерево ван Эмде Боаса (van Emde Boas tree, vEB tree)** – это дерево поиска (search tree) для хранения целочисленных m -битных ключей
- Ключ – целое неотрицательное число из множества
$$U = \{0, 1, \dots, u - 1\}, \quad u = 2^m$$
- Основные операции (Insert, Delete, Lookup, Min, Max) выполняются за время $O(\log \log u)$, что асимптотически лучше, чем $O(\log n)$ в сбалансированных бинарных деревьях поиска (AVL tree, red-black tree)
- **Автор: Peter van Emde Boas, 1975**
- Peter van Emde Boas. **Preserving order in a forest in less than logarithmic time** // Proceedings of the 16th Annual Symposium on Foundations of Computer Science, 1975, p. 75-84



Дерево ван Эмде Боаса (Van Emde Boas tree)

- **Дерево ван Эмде Боаса (van Emde Boas tree, vEB tree)** – это дерево поиска (search tree) для хранения целочисленных неотрицательных ключей из множества $\{0, 1, \dots, u - 1\}$
- Например, пусть ключи – это целые числа из множества $\{0, 1, \dots, 1000\}$
- Тогда $u = 1000 + 1 = 1001$ (мощность множества ключей)
- Для хранения ключа с максимальным значением 1000 необходимо $\lceil \log_2 1000 \rceil + 1 = \lceil 9.97 \rceil + 1 = 10$ бит

Максимальное значение ключа	Количество бит для хранения ключа
255	8 (uint8_t)
65 535	16 (uint16_t)
4 294 967 295	32 (uint32_t)
$2^{64} - 1$	64 (uint64_t)

Прямая адресация

- Динамическое множество значений из универсума $U = \{0, 1, \dots, u - 1\}$ можно хранить в массиве $A[0, \dots, u - 1]$ из u бит
- Элемент $A[x]$ хранит 1, если значение x принадлежит множеству, 0 – в противном случае
- Операции Insert, Delete, Member/Lookup выполняются за время $O(1)$
- Min, Max, Successor, Predecessor выполняются за время $O(u)$ – требуется просмотреть весь битовый вектор A

Min

Max

0	1	2	3														18	
1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0

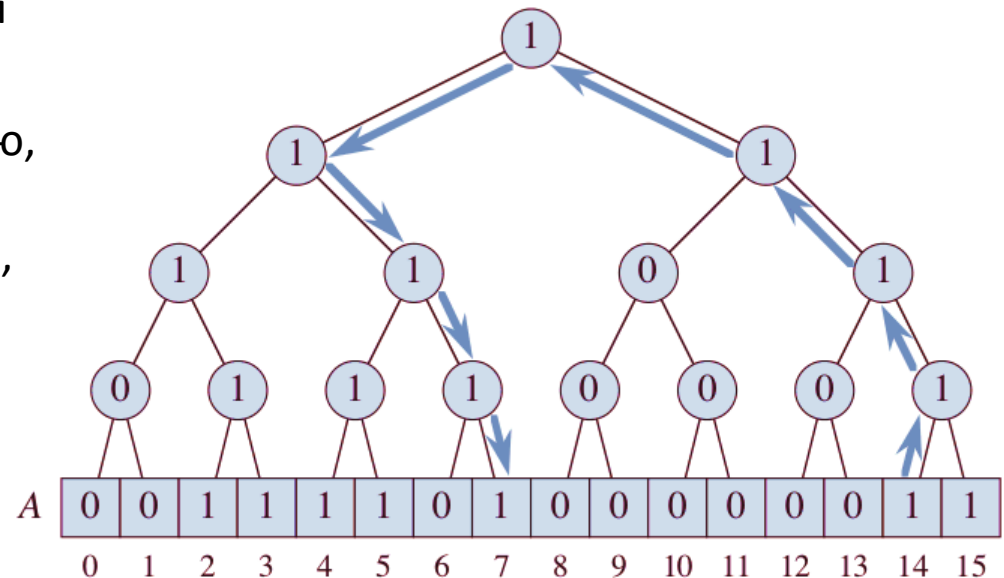
Универсум из 3-х элементов $U = \{0, 3, 18\}$, $u = 20$

Successor(0) = 3, Successor(18) = NULL

Predecessor(18) = 3

Наложение бинарного дерева

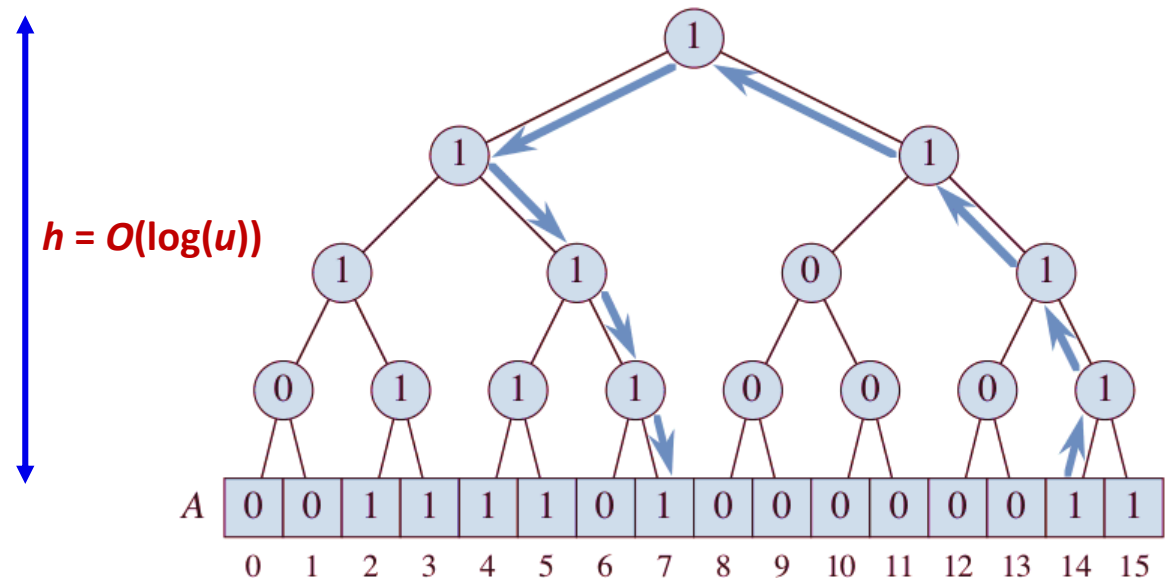
- Динамическое множество значений из универсума $U = \{0, 1, \dots, u - 1\}$ можно хранить в массиве $A[0, \dots, u - 1]$ из u бит
- Элементы битового вектора – это листья бинарного дерева, внутренний узел содержит 1 тогда и только тогда, когда некоторый лист в его поддереве содержит 1 (внутренний узел содержит логическое ИЛИ дочерних узлов)
- **Min** – от корня вниз, выбирая самый левый узел с 1
- **Max** – от корня вправо, выбирая самый правый узел с 1
- **Predecessor(x)** – от листа к корню, пока не войдем справа в узел с левым дочерним элементом 1, затем вниз, выбирай крайний справа узел с 1
- Высота дерева $h = O(\log(u))$
Сложность операций
Min/Max/Predecessor – $O(\log(u))$
- Сложность Lookup – $O(1)$



$U = \{2, 3, 4, 5, 7, 14, 15\}, \quad u = 16$

Наложение дерева постоянной высоты

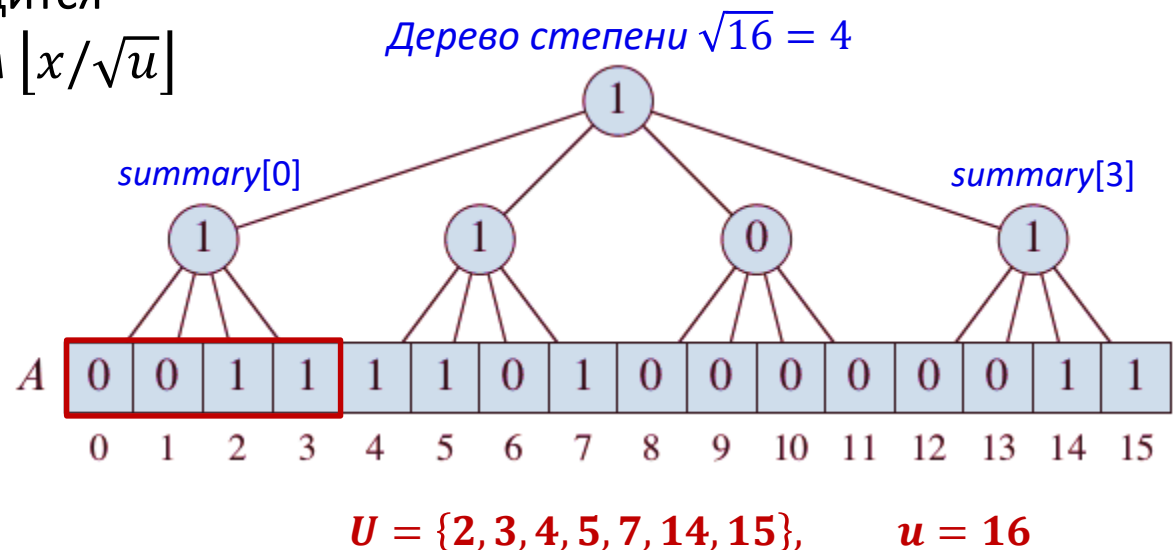
- Динамическое множество значений из универсума $U = \{0, 1, \dots, u - 1\}$ можно хранить в массиве $A[0, \dots, u - 1]$ из u бит
- Элементы битового вектора – это листья бинарного дерева, внутренний узел содержит 1 тогда и только тогда, когда некоторый лист в его поддереве содержит 1 (внутренний узел содержит логическое ИЛИ дочерних узлов)
- Можно ли наложить на бинарный вектор дерево постоянной высоты (не зависящей от u)?



$$U = \{2, 3, 4, 5, 7, 14, 15\}, \quad u = 16$$

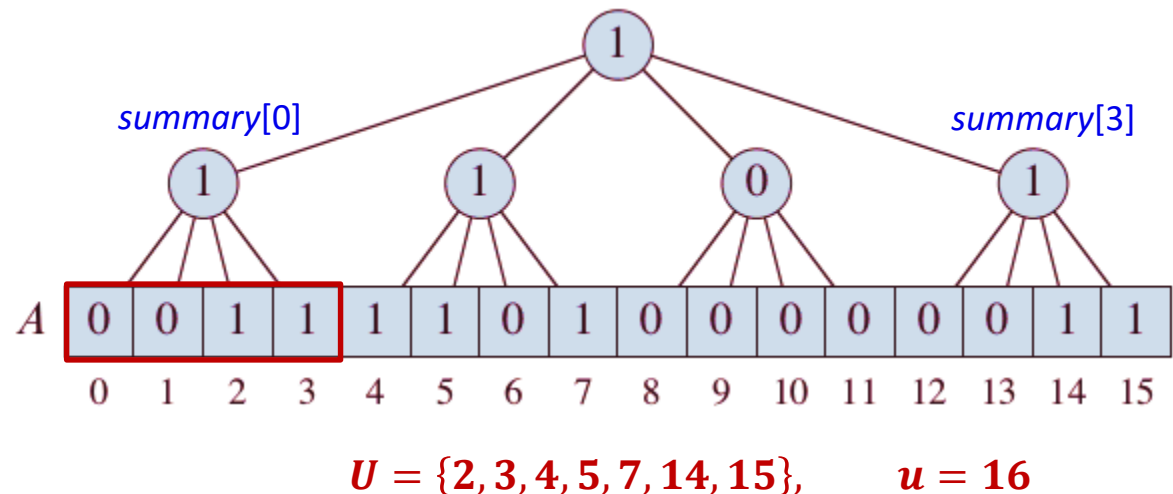
Наложение дерева постоянной высоты

- Пусть размер универсума $u = 2^{2k}$, так что \sqrt{u} целое число
- Наложим на битовый вектор дерево степени \sqrt{u}
- Узлы на уровне 1 – это результат ИЛИ для группы из \sqrt{u} бит ($\sqrt{16} = 4$ бит)
- Узлы на уровне 1 – это элементы массива $summary[0.. \sqrt{u} - 1]$
- $summary[i]$ – это логическое ИЛИ подмассива $A[i\sqrt{u} .. (i + 1)\sqrt{u} - 1]$
- $summary[i]$ – это кластер (cluster) i
- Бит x массива A находится в кластере с номером $\lfloor x/\sqrt{u} \rfloor$



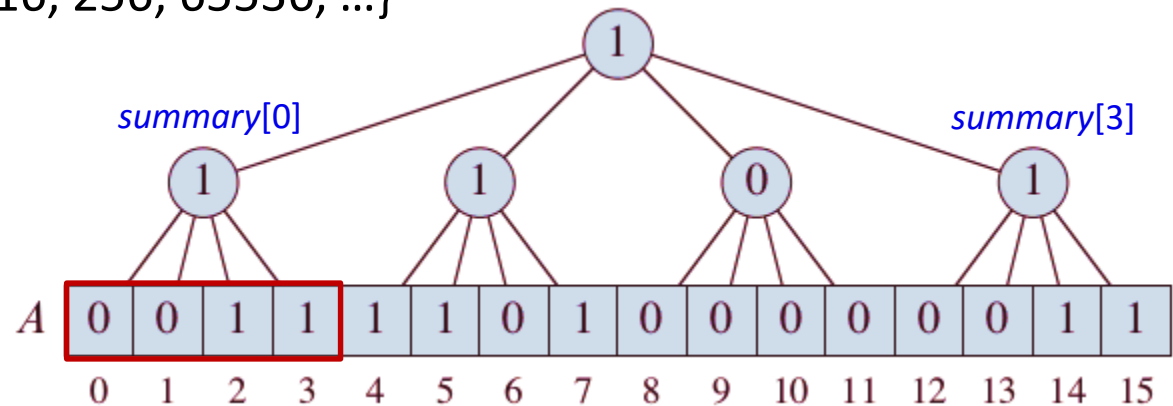
Наложение дерева постоянной высоты

- **Insert:** $A[x] = 1$, $\text{summary}[\lfloor x/\sqrt{u} \rfloor] = 1$ $O(1)$
- **Min/Max:** ищем крайний слева (справа) элемент $\text{summary}[i] = 1$, затем в кластере i находим крайней слева (справа) бит 1 $O(\sqrt{u})$
- **Successor/Predecessor:** ищем в пределах кластера вправо (влево) бит 1, если не нашли, то ищем кластер справа (слева) от $\lfloor x/\sqrt{u} \rfloor$, который содержит 1 и в нем отыскиваем крайний слева (справа) бит 1 $O(\sqrt{u})$
- **Delete:** $A[x] = 0$, $\text{summary}[\lfloor x/\sqrt{u} \rfloor] = \text{логическое ИЛИ битов кластера } i$ $O(\sqrt{u})$



Рекурсивная структура

- Имеем универсум из u элементов
- Создаем структуры хранящие $\sqrt{u} = u^{1/2}$ элементов,
которые хранят структуры по $u^{1/4}$ элементов,
которые хранят структуры по $u^{1/8}$ элементов,
...
до структур по 2 элемента
- Считаем, что $u = 2^{2^k}$, для некоторого целого k
- Тогда u , $u^{1/2}$, $u^{1/4}$ и т.д. – целые числа
- u из множества $\{2, 4, 16, 256, 65536, \dots\}$



$$U = \{2, 3, 4, 5, 7, 14, 15\}, \quad u = 16$$

Рекурсивная структура

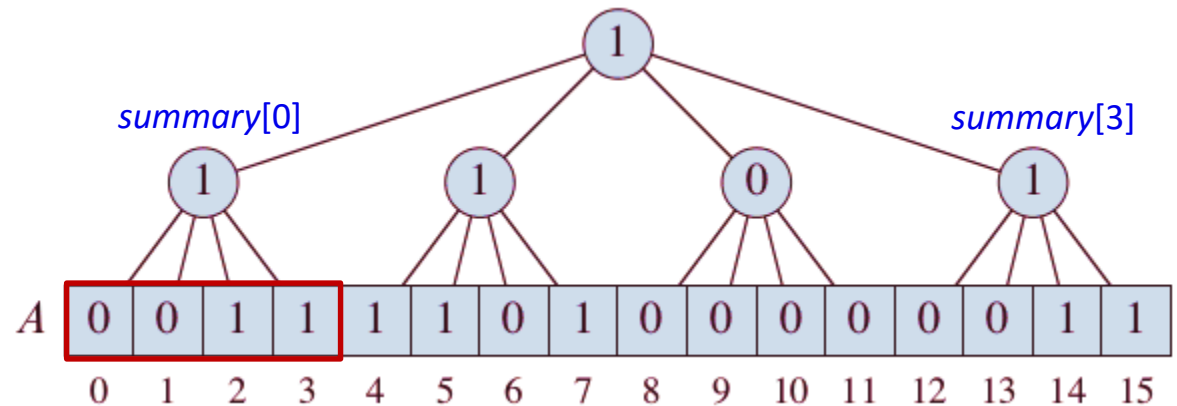
- Значение x располагается в кластере с номером $\lfloor x/\sqrt{u} \rfloor$
- Будем считать, что x – это $\log_2 u$ – битовое число, тогда номер кластера определяется $\log_2(u) / 2$ старшими битами числа x

$$\text{high}(x) = \lfloor x/\sqrt{u} \rfloor$$

- В своем кластере x находится в позиции $x \bmod \sqrt{u}$, которая задается младшими $\log_2(u) / 2$ битами x

$$\text{low}(x) = x \bmod \sqrt{u}$$

$$\text{index}(h, l) = h\sqrt{u} + l$$



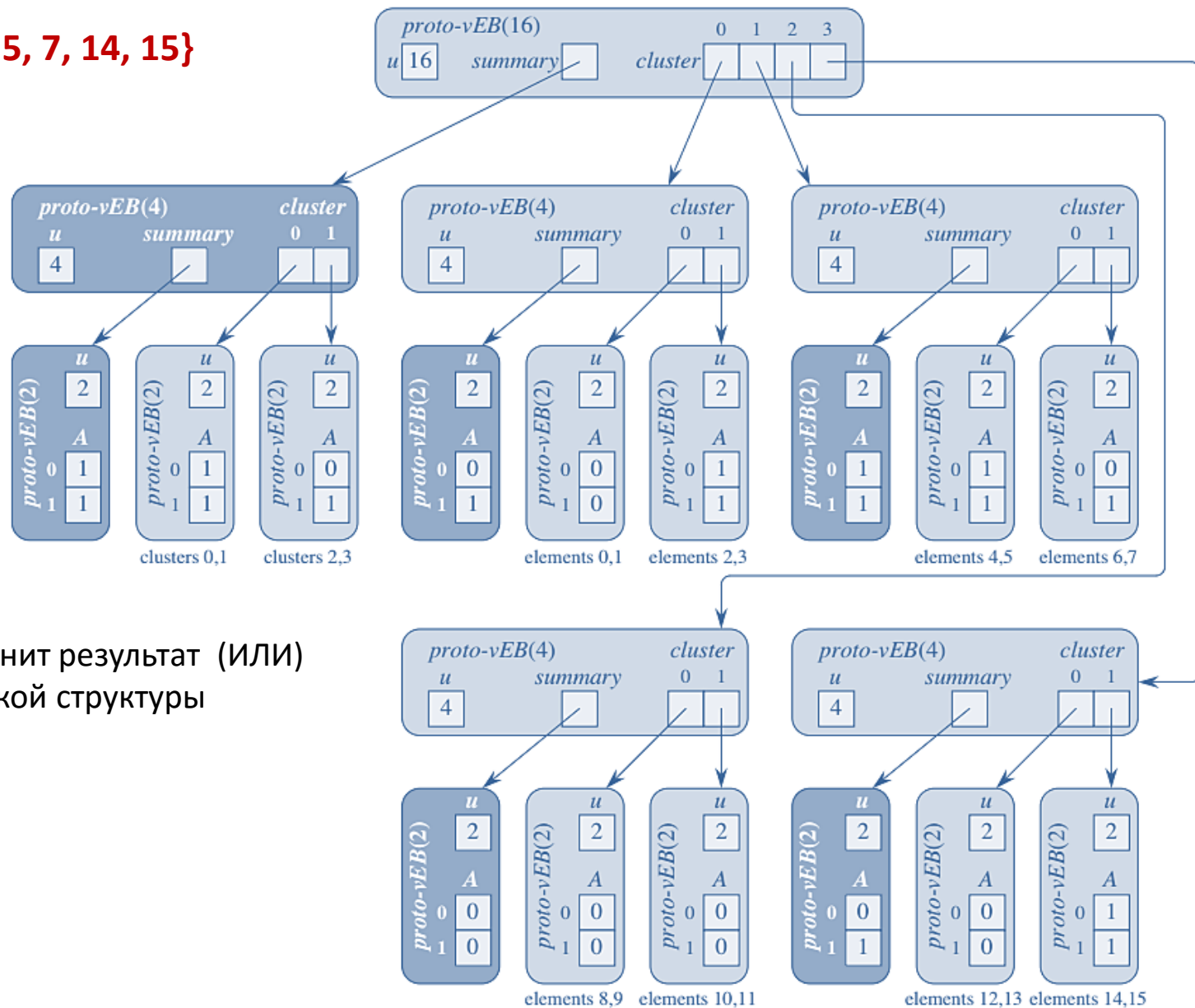
$$U = \{2, 3, 4, 5, 7, 14, 15\}, \quad u = 16$$

Протоструктура ван Эмде Боаса (proto-vEB)

- Обозначим через ***proto-vEB***(u) протоструктуру ван Эмде Боаса, содержащее ключи из множества $\{0, 1, \dots, u - 1\}$
- Каждый узел дерева ***proto-vEB***(u) содержит:
 - значение u – размер универсума
 - если $u = 2$
 - это базовый размер и структура содержит массив $A[0..1]$ из двух бит
 - иначе
 - указатель **summary** на структуру ***proto-vEB***(\sqrt{u})
 - массив **cluster**[$0.. \sqrt{u} - 1$] указателей на структуры ***proto-vEB***(\sqrt{u})

Протоструктура ван Эмде Боаса (proto-vEB)

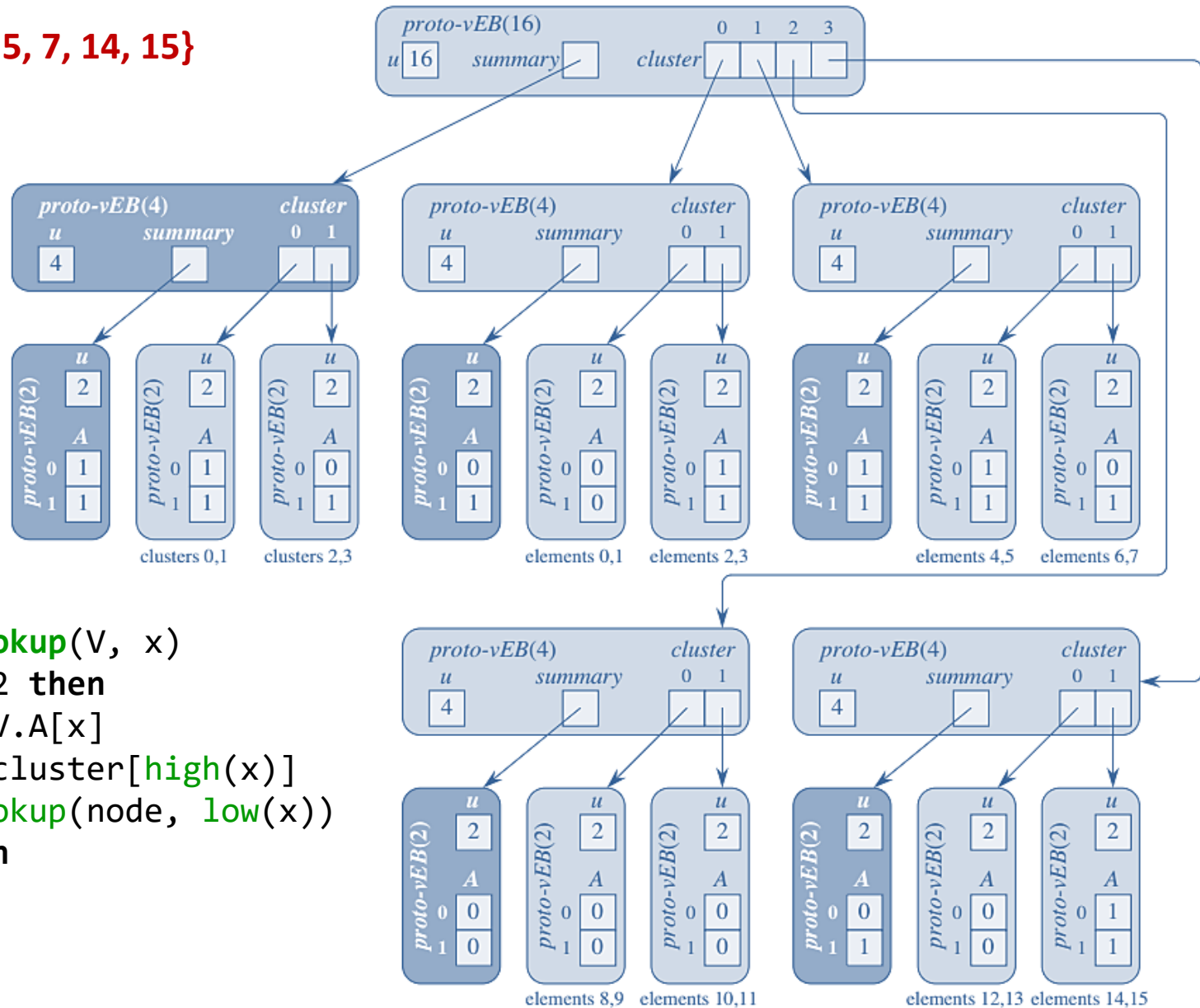
$U = \{2, 3, 4, 5, 7, 14, 15\}$
 $u = 16$



summary – хранит результат (ИЛИ)
 для родительской структуры

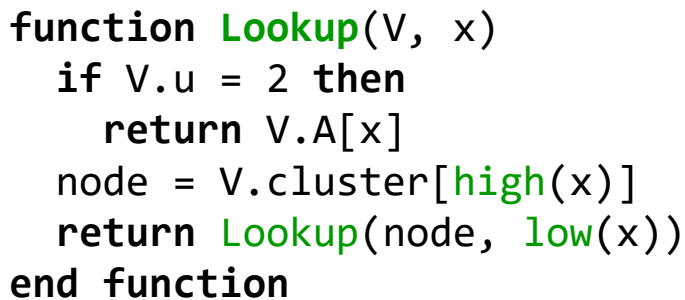
Поиск элемента

$U = \{2, 3, 4, 5, 7, 14, 15\}$
 $u = 16$



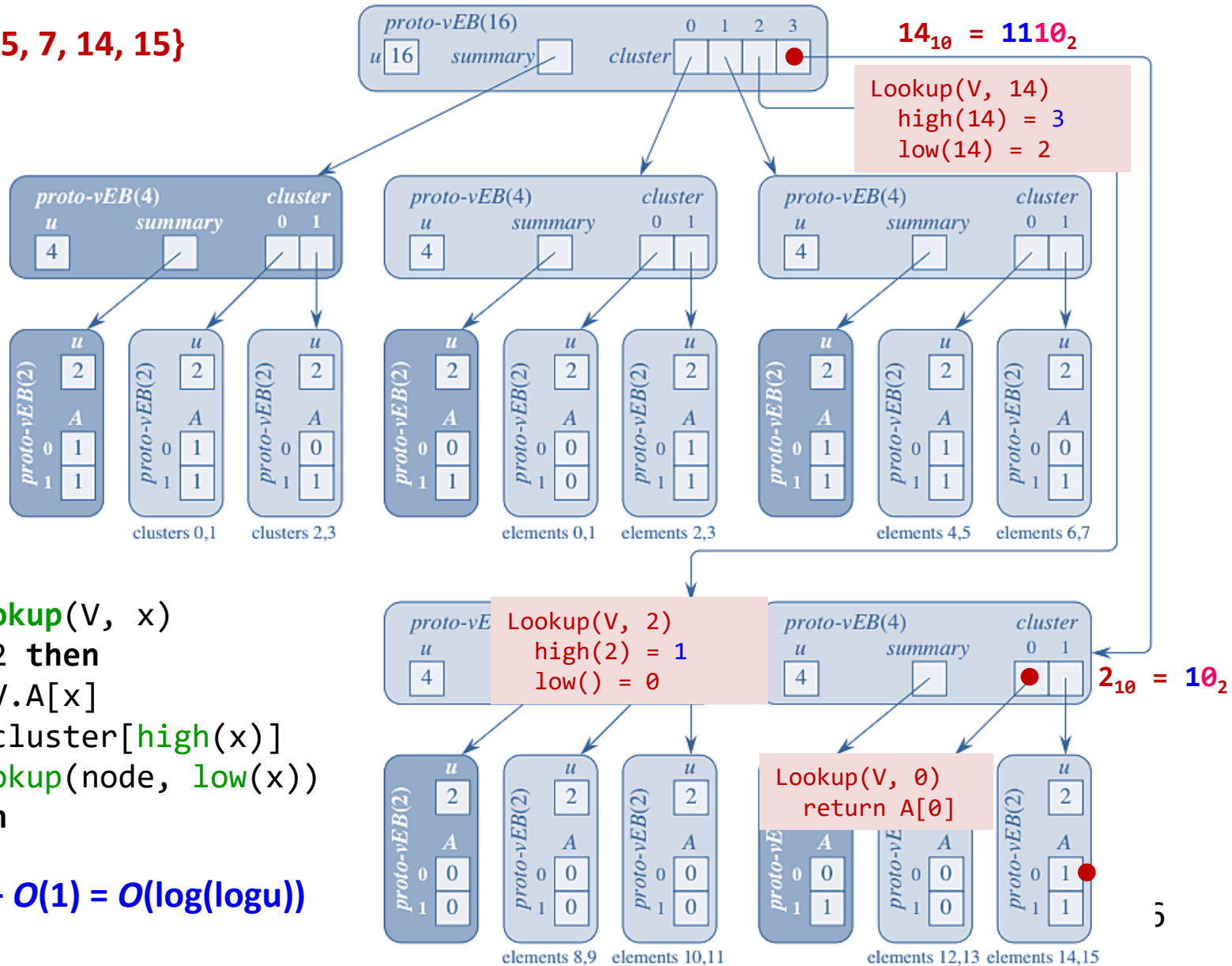
```
function Lookup(V, x)
  if V.u = 2 then
    return V.A[x]
  node = V.cluster[high(x)]
  return Lookup(node, low(x))
end function
```

$U = \{2, 3, 4, 5, 7, 14, 15\}$
 $u = 16$



Поиск элемента

$U = \{2, 3, 4, 5, 7, 14, 15\}$
 $u = 16$

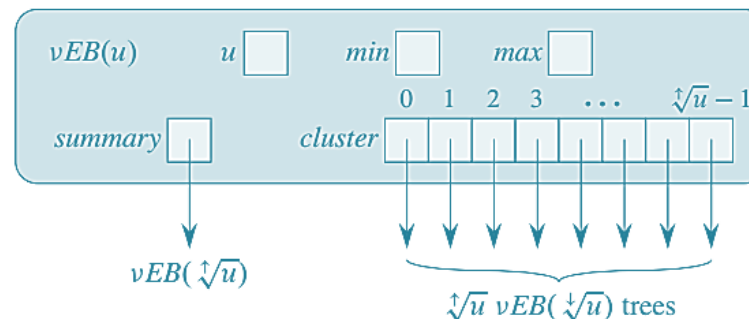


```
function Lookup(V, x)
  if V.u = 2 then
    return V.A[x]
  node = V.cluster[high(x)]
  return Lookup(node, low(x))
end function
```

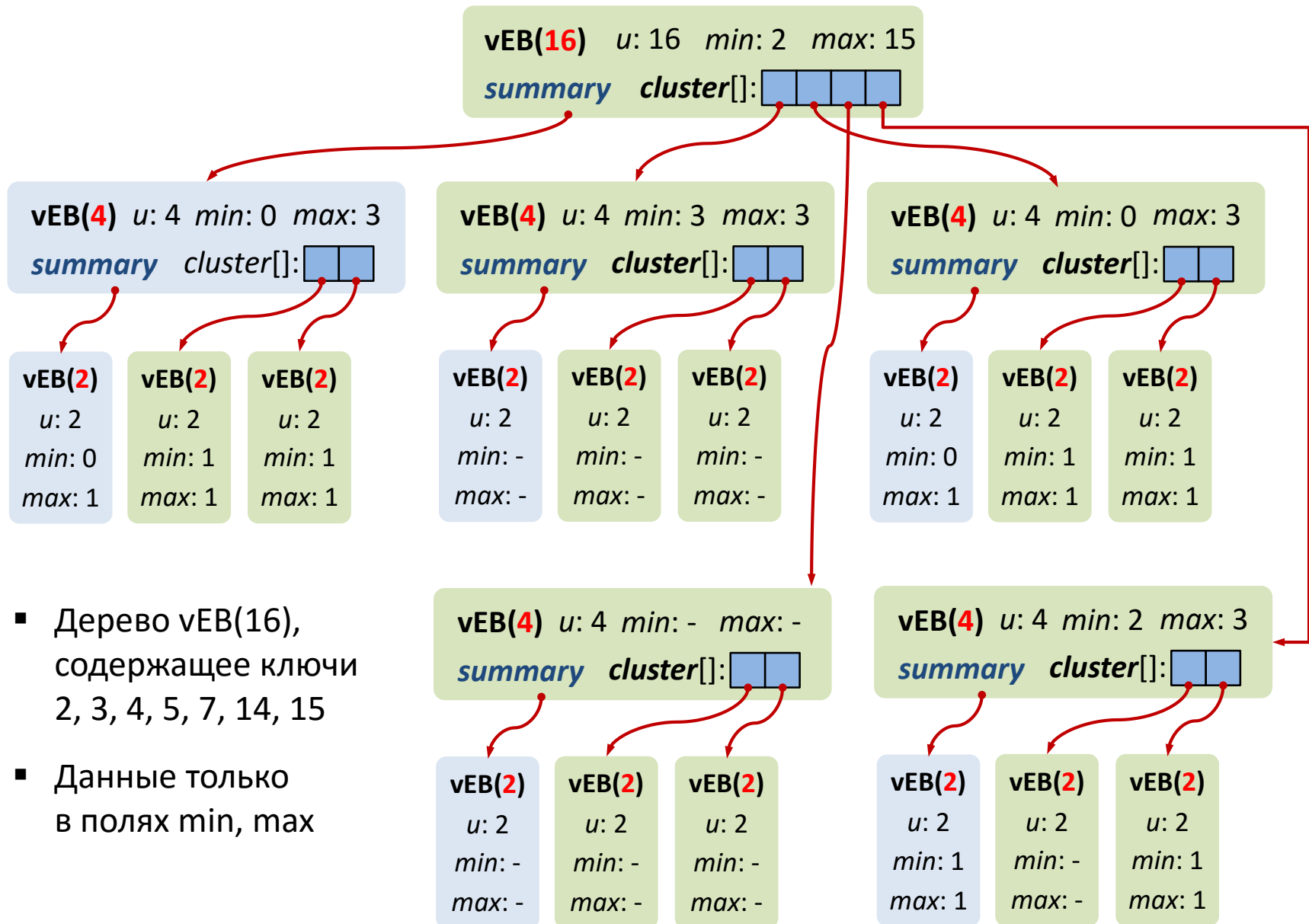
$$T(u) = T(\sqrt{u}) + O(1) = O(\log(\log u))$$

Структура дерева ван Эмде Боаса

- Обозначим через $vEB(u)$ дерево ван Эмде Боаса, содержащее ключи из множества $\{0, 1, \dots, u-1\}$
- Каждый узел дерева $vEB(u)$ содержит:
 - указатель **summary** на дерево $vEB(2^{\lceil \frac{\log_2 u}{2} \rceil})$
 - массив **cluster** $[0..2^{\lceil \frac{\log_2 u}{2} \rceil} - 1]$ указателей на корни деревьев $vEB(2^{\lceil \frac{\log_2 n}{2} \rceil})$
 - минимальный **min** элемент в дереве vEB (копии min нет в поддеревьях cluster[...])
 - максимальный **max** элемент в дереве vEB
 - значение u

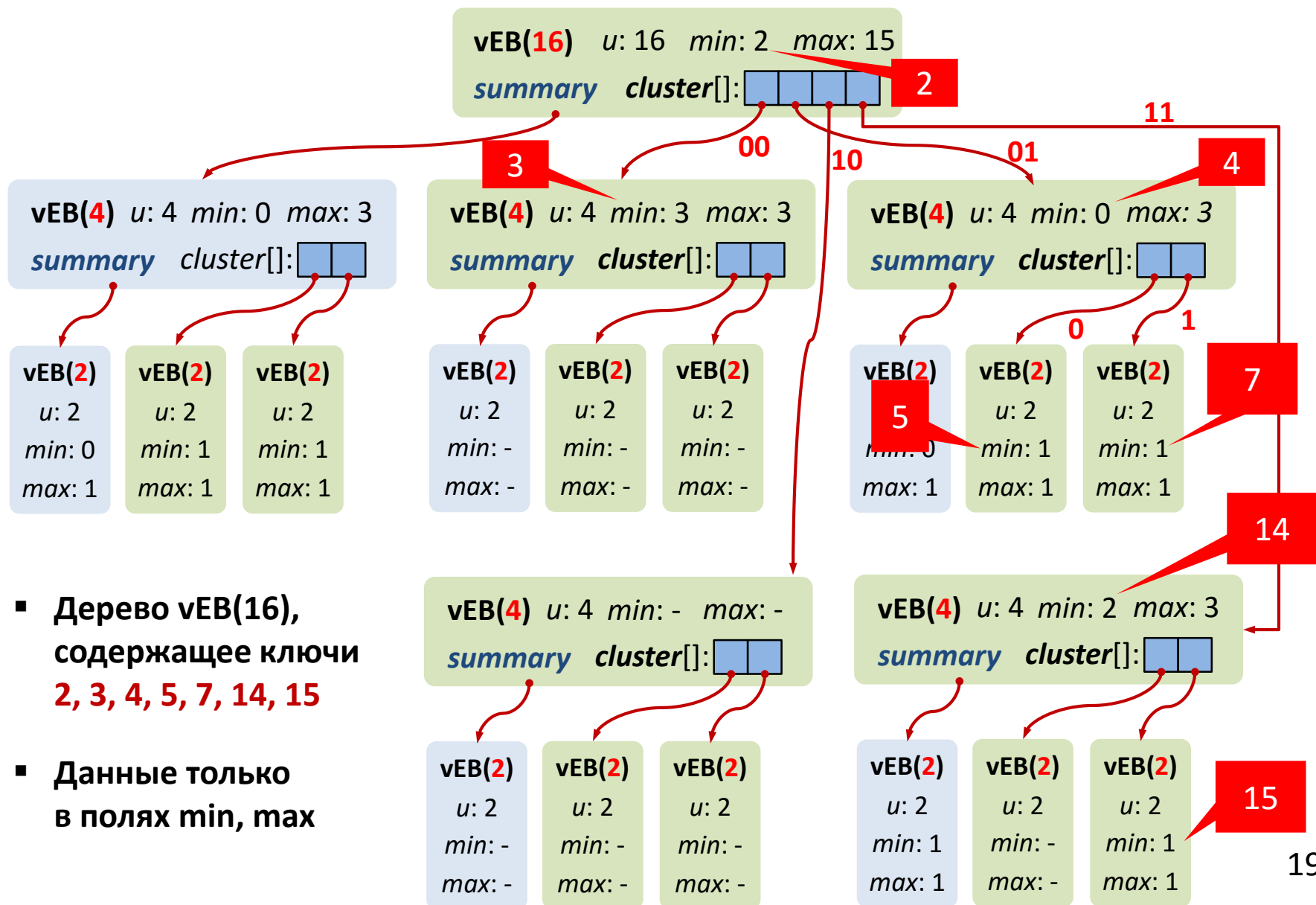


Пример дерева vEB(16)



- Дерево vEB(16),
содержащее ключи
2, 3, 4, 5, 7, 14, 15
- Данные только
в полях min, max

Пример дерева vEB(16)



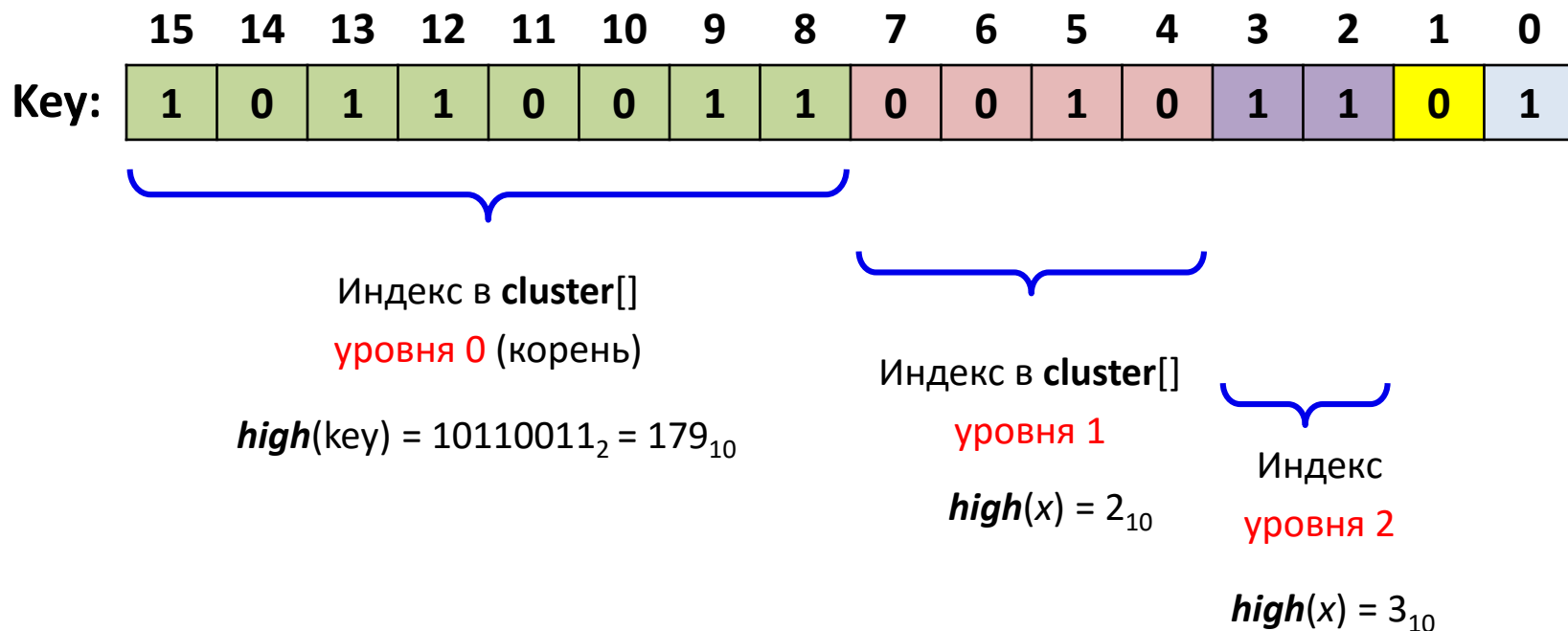
- Дерево vEB(16),
содержащее ключи
2, 3, 4, 5, 7, 14, 15
- Данные только
в полях min, max

Структура дерева ван Эмде Боаса

- Значения (value) ассоциированные с ключами хранятся только в полях min, max узлов (min.value, max.value)
- Ключ не хранится в узлах (аналогия с trie, prefix tree) – биты ключа распределены по узлам на пути от корня к листьям
- Время выполнения операций не зависит от количества n элементов в дереве
- Хранение min и max в узлах позволяет сократить глубину спуска по дереву при поиске минимального/максимального элемента, а также при поиске следующего элемента (successor) и предыдущего (predecessor)

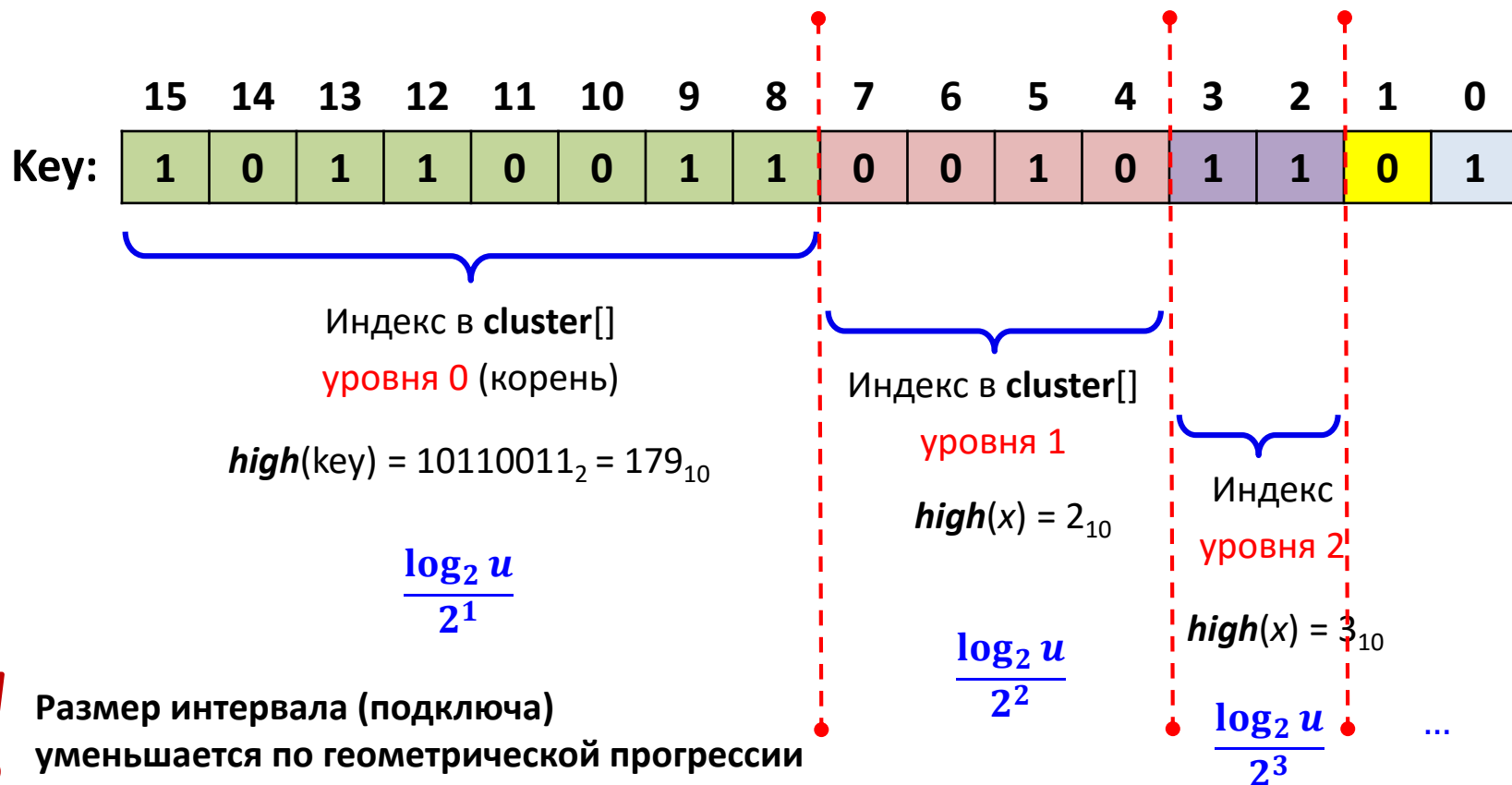
Структура дерева ван Эмде Боаса

- В ключе “закодирован” путь в дереве vEB
- Пример, ключи из множества $\{0, 1, \dots, 65535\}$, $u = 65536 = 2^{16}$
- Дерево vEB(65536)



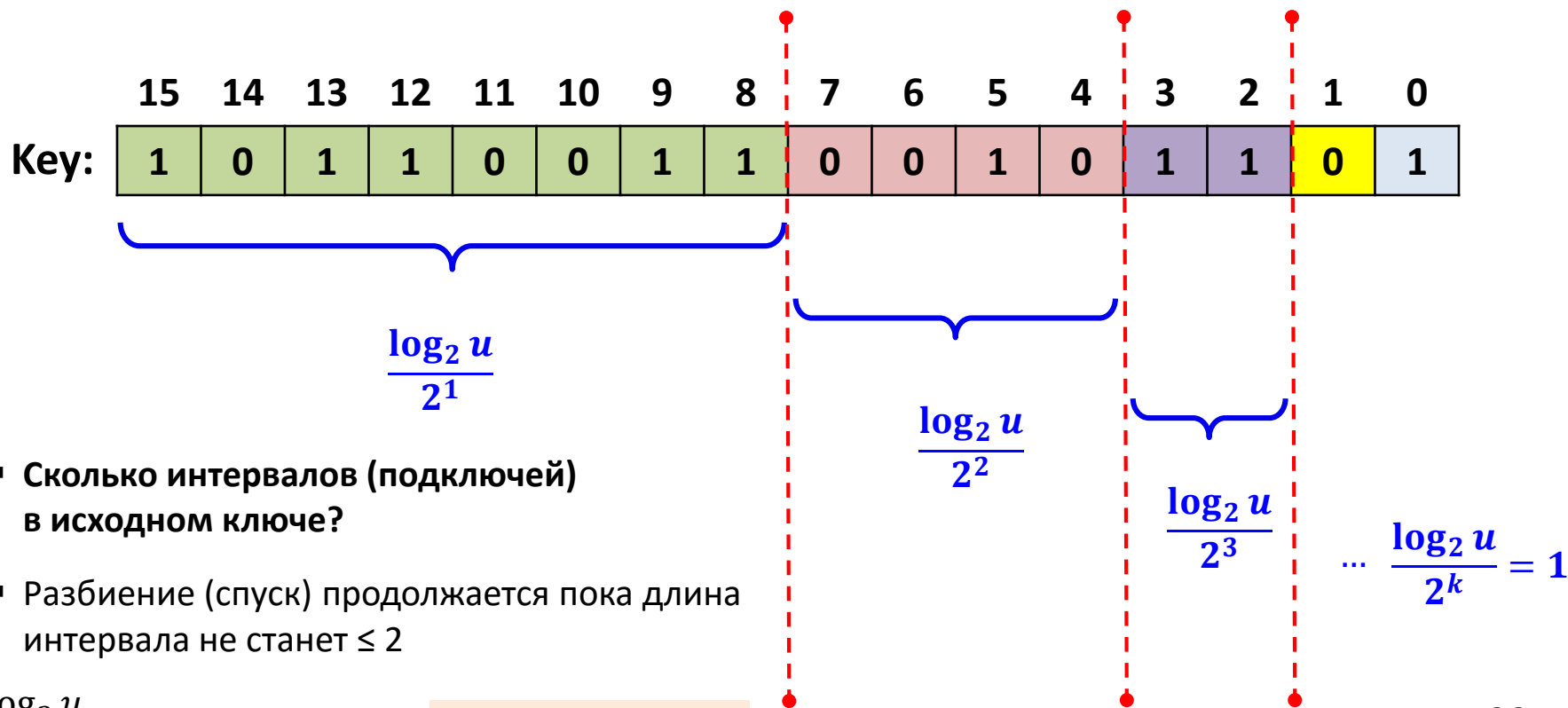
Структура дерева ван Эмде Боаса

- В ключе “закодирован” путь в дереве vEB
- Пример, ключи из множества $\{0, 1, \dots, 65535\}$, $u = 65536 = 2^{16}$
- Дерево vEB(65536)



Структура дерева ван Эмде Боаса

- В ключе “закодирован” путь в дереве vEB
- Пример, ключи из множества $\{0, 1, \dots, 65535\}$, $u = 65536 = 2^{16}$
- Дерево vEB(65536)



- Сколько интервалов (подключей) в исходном ключе?
- Разбиение (спуск) продолжается пока длина интервала не станет ≤ 2

$$\frac{\log_2 u}{2^k} = 1, \quad \log_2 u = 2^k, \quad k = \log(\log u)$$

Структура дерева ван Эмде Боаса

- Дерево vEB(65536), $u = 65536 = 2^{16}$

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Key:	1	0	1	1	0	0	1	1	0	0	1	0	1	1	0	1



Индекс в **cluster**[]

уровня 0 (корень)

$$high(key) = 10110011_2 = 179_{10}$$

$$high(x) = \left\lfloor \frac{x}{2^{\left\lfloor \frac{\log_2 u}{2} \right\rfloor}} \right\rfloor$$

Половина старших битов

$$high(45869) = \left\lfloor \frac{45869}{2^8} \right\rfloor = 179$$



Индекс в **cluster**[]

уровня 1

$$high(x) = 2_{10}$$



Индекс

уровня 2

$$high(x) = 3_{10}$$

$$low(x) = x \bmod 2^{\left\lfloor \frac{\log_2 u}{2} \right\rfloor}$$

Половина младших битов

$$low(45869) = 45869 \bmod 256 = 45$$

Поиск экстремальных (min/max) элементов

- Вместе с деревом хранятся минимальный и максимальный элементы, доступ к ним осуществляется за время $O(1)$

```
function vEB_Min(tree)
    return tree.min
end function
```

$$T_{Min} = O(1)$$

```
function vEB_Max(tree)
    return tree.max
end function
```

$$T_{Max} = O(1)$$

Поиск элемента в дереве vEB

```
function vEB_Lookup(tree, key)
  if key = tree.min.key then
    return tree.min
  if key = tree.max.key then
    return tree.max
```

$$T_{Lookup} = O(\log(\log u))$$

```
// Дерево vEB(2) данных кроме min, max не имеет
if tree.u = 2
  return NULL
```

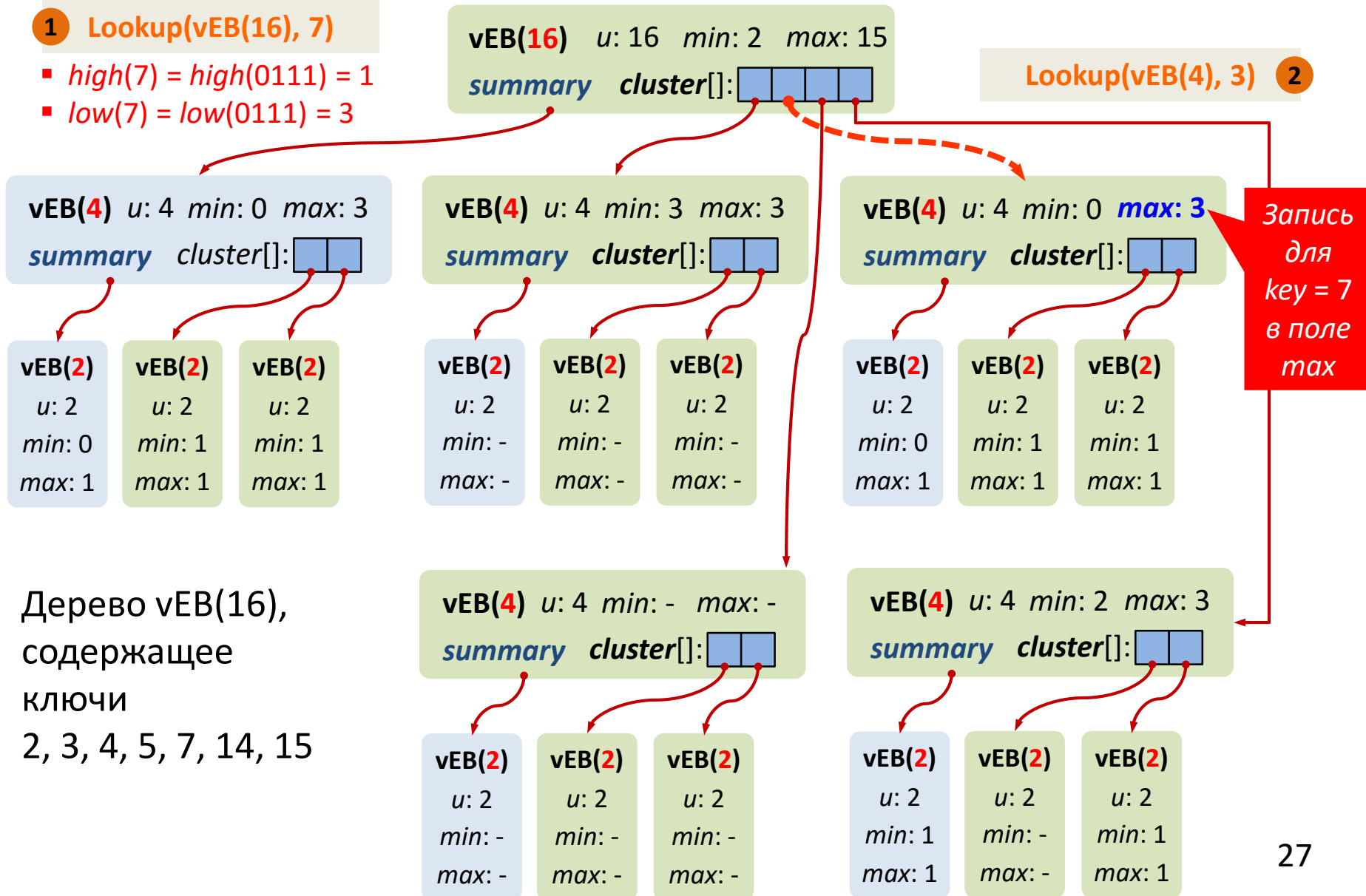
```
hi = high(key)           // Номер поддерева в cluster[]
lo = low(key)             // Новый ключ
return vEB_Lookup(tree.cluster[hi], lo)
end function
```

- Рекурсивно спускаемся по дереву проверяя поля min и max
- С каждым рекурсивным вызовом длина ключа *key* уменьшается:
key -> low(key) -> low(low(key)) -> low(low(low(key))) -> ...

Поиск в дереве vEB(16) элемента 7 (0111₂)

1 Lookup(vEB(16), 7)

- $high(7) = high(0111) = 1$
- $low(7) = low(0111) = 3$



Дерево vEB(16),
содержащее
ключи
2, 3, 4, 5, 7, 14, 15

Добавление элемента в пустое дерево vEB

```
function vEB_AddEmpty(tree, key, value)
    // Дерево пусто – заполняем поля min, max
    tree.min.key = key
    tree.min.value = value
    tree.max.key = key
    tree.max.value = value
end function
```

$$T_{AddEmpty} = O(1)$$

- При вставке элемента в пустое дерево, заносим его в поля min и max
- После вставки в дереве один элемент и min = max

Добавление элемента в дерево vEB

```
function vEB_Add(tree, key, value)
  if tree.min = NULL then
    // Дерево пусто
    vEB_AddEmpty(tree, key, value)
  end if

  if key < tree.min.key then
    // Заменяем min новым значением, а старый min
    // добавим в поддереву
    swap(<min.key, min.value>, <key, value>)
  end if
```

- Добавление элемента в дерево выполняется рекурсивно, при каждом вызове (для каждого поддерева) обрабатываем возможные ситуации:
- Если поддерево пусто, добавляем элемент в поля min и max
- Если ключ меньше ключа min, заменяем в min ключ на новый, а старый ключ min вставим в поддерево позднее (см. ниже)

Добавление элемента в дерево vEB

```
if tree.u > 2 then
    hi = high(key)
    lo = low(key)
    if vEB_Min(tree.cluster[hi]) = NULL then
        // Поддерево hi пусто
        vEB_Add(tree.summary, hi)
        vEB_AddEmpty(tree.cluster[hi], lo)
    else
        vEB_Add(tree.cluster[hi], lo)
end if

if key > tree.max.key then
    // Обновляем max
    tree.max.key = key
    tree.max.value = value
end if
end function
```

$$T_{Add} = O(\log(\log(u)))$$

- Рекурсивно спускаемся пока, не дойдем до узла с $u \leq 2$

Вставка в дерево vEB(16) элемента 6 (0110₂)

1 Add(vEB(16), 6)

- $high(6) = high(0110) = 1$
- $low(6) = low(0110) = 2$

vEB(16) u: 16 min: 2 max: 15

summary cluster[]:

2 Add(vEB(4), 2)

vEB(4) u: 4 min: 0 max: 3

summary cluster[]:

vEB(4) u: 4 min: 3 max: 3

summary cluster[]:

vEB(4) u: 4 min: 0 max: 3

summary cluster[]:

vEB(2)

u: 2
min: 0

vEB(2)

u: 2
min: 1

vEB(2)

u: 2
min: 1

vEB(2)

u: 2
min: -

vEB(2)

u: 2
min: -

vEB(2)

u: 2
min: -

vEB(2)

u: 2
max: 1

vEB(2)

u: 2
max: 1

vEB(2)

u: 2
min: 0
max: 1

6

7

Add(vEB(2), 0)

3

vEB_Add(vEB(16), 6)

hi = 1, lo = 2

vEB_Min(vEB(16).cluster[1]) != NULL

vEB_Add(vEB(16).cluster[1], 2)

hi = 1, lo = 0

vEB_Min(vEB(4).cluster[1]) != NULL

vEB_Add(vEB(4).cluster[1], 0)

Обмениваем min=1 и key=0

vEB(4) u: 4 min: 2 max: 3

summary cluster[]:

vEB(2)

u: 2
min: 1
max: 1

vEB(2)

u: 2
min: -
max: -

vEB(2)

u: 2
min: 1
max: 1

Удаление элемента из дерева vEB


- Рекурсивно спускаемся по дереву vEB и ищем узел, которому соответствует ключ
- Если дерево (узел) содержит один элемент ($\min = \max$), удаляем \min и \max
- Если в дереве (узле) 2 элемента ($u = 2$), удаляем один из них и корректируем \min и \max ; в дереве остается один элемент и $\min = \max$
- ...

HOME WORK

Изучить алгоритм удаления элемента из дерева vEB
[CLRS_3ed, C. 590] “Удаление элемента”

Эффективность дерева vEB

- Высота дерева определяется значением u – длиной ключа
- При **первом** вызове Lookip длина ключа $\log_2 u$
- На **втором** рекурсивном вызове длина ключа $\frac{\log_2 u}{2}$
(в два раза короче – $\text{low}(\text{key})$)
- На **третьем** вызове длина ключа: $\frac{\log_2 u}{2^2}$
(в четыре раза короче – $\text{low}(\text{low}(\text{key}))$)
- и т.д. пока не дойдем до узла с $u = 2$

$$\frac{\log_2 u}{2^1}, \quad \frac{\log_2 u}{2^2}, \quad \frac{\log_2 u}{2^3}, \quad \dots, \quad \frac{\log_2 u}{2^k} = 1$$


$$\log_2 u = 2^k,$$
$$k = \log_2 \log_2 u$$

Дерево $\text{vEB}(u)$ имеет высоту $O(\log_2 \log_2 u)$

Дерево ван Эмде Боаса (van Emde Boas tree)

Операция	Худший случай (worst case)
Add (<i>key</i> , <i>value</i>)	$O(\log_2 \log_2 u)$
Lookup (<i>key</i>)	$O(\log_2 \log_2 u)$
Delete (<i>key</i>)	$O(\log_2 \log_2 u)$
Min	$O(1)$
Max	$O(1)$

- Сложность по памяти: $O(2^{\log u}) = O(u)$
- u – максимальное значение ключа + 1

Плюсы и минусы дерева vEB

- **Достоинства дерева vEB**
- Вычислительная сложность операций $O(\log_2 \log_2 u)$:
 - ❑ асимптотически быстрее чем сбалансированные деревья поиска (AVL tree, red-black tree)
 - ❑ сложность операций не зависит от количества n элементов в дереве (в словаре)
- **Недостатки**
- Применимо лишь в случае целых неотрицательных ключей
- Высокие требования к памяти $O(u) = O(2^{\log u})$

Применение дерева vEB

- **Словарь** (ассоциативный массив) с целочисленными ключами
- **Сортировка** n целочисленных ключей за время $O(n \cdot \log_2 \log_2 u)$ – быстрее чем поразрядная сортировка (Radix sort)
- Реализация **кучи** и применение в алгоритме Дейкстры построения кратчайшего пути в графе: реализация DecreaseKey за время $O(\log_2 \log_2 u)$, таким образом итоговое время работы алгоритма Дейкстры составит $O(E \cdot \log_2 \log_2 u)$

Задания

- Изучить алгоритм удаления элемента из дерева vEB
[CLRS_3ed, C. 590] “Удаление элемента”
- Разобраться с функциями удаления, поиска следующего (Successor) и предыдущего (Predecessor) элементов в vEB