Лекция 7.
Фибоначчиевы кучи

Даниил Михайлович Берлизов
Старший преподаватель Кафедры вычислительных систем СибГУТИ
E-mail: sillyhat34@gmail.com

Курс «Структуры и алгоритмы обработки данных»
Осенний семестр, 2021 г.

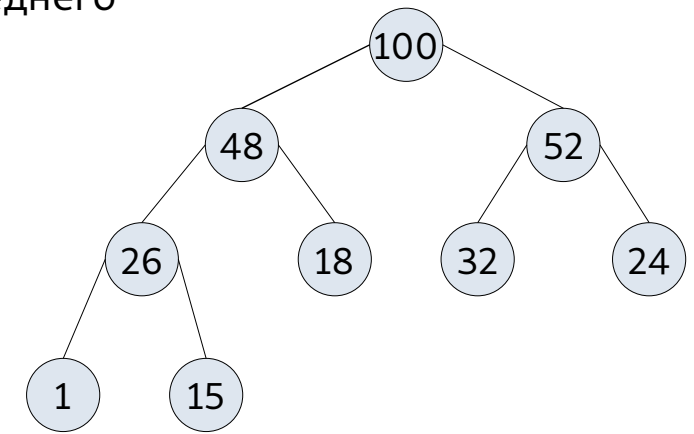
Очередь с приоритетом (priority queue)

- **Очередь с приоритетом** (*priority queue*) — это очередь, в которой элементы имеют приоритет (вес)
- **Поддерживаемые операции:**
  - **Insert(*key*, *value*)** — добавление в очередь значения *value* с приоритетом (весом, ключом) *key*
  - **DeleteMin / DeleteMax** — удаление элемента с минимальным / максимальным приоритетом
  - **Min / Max** — возврат элемента с минимальным / максимальным ключом
  - **DecreaseKey** — изменение приоритета (значения ключа) заданного элемента
  - **Merge(q_1, q_2)** — слияние двух очередей в одну

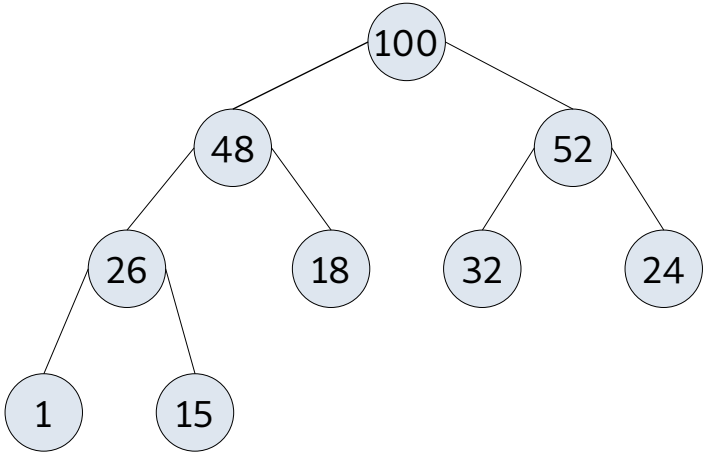
Значение (*value*)	Приоритет (*key*)
Слон	4
Кит	1
Борис	12

Бинарная куча (binary heap)

- **Бинарная куча** (*пирамида, сортирующее дерево, binary heap*) — бинарное дерево, удовлетворяющее следующим условиям:
  - Приоритет (ключ) любой вершины **не меньше** (для *max-heap*) или **не больше** (для *min-heap*) приоритета её потомков
  - Дерево является завершённым бинарным деревом (*complete binary tree*) — все уровни заполнены слева направо, возможно за исключением последнего

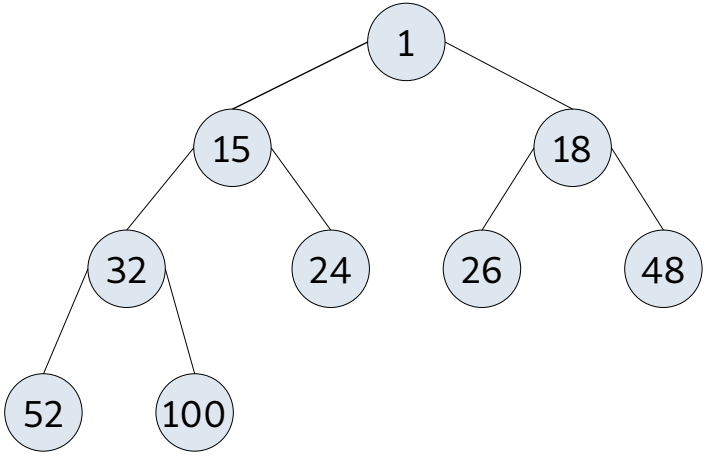


Бинарная куча (binary heap)



max-heap

Приоритет любой вершины **не меньше (≥)** приоритета потомков



min-heap

Приоритет любой вершины **не больше (≤)** приоритета потомков

Очередь с приоритетом (priority queue)

- В таблице приведены трудоёмкости операций различных очередей с приоритетом в худшем случае (*worst case*)
- Символом "*" отмечена амортизированная сложность операций

Операция	Binary heap	Binomial heap	Fibonacci heap	Pairing heap	Brodal heap
FindMin	$\Theta(1)$	$O(\log n)$	$\Theta(1)^*$	$\Theta(1)^*$	$\Theta(1)$
DeleteMin	$\Theta(\log n)$	$\Theta(\log n)$	$O(\log n)^*$	$O(\log n)^*$	$O(\log n)$
Insert	$\Theta(\log n)$	$O(\log n)$	$\Theta(1)^*$	$\Theta(1)^*$	$\Theta(1)$
DecreaseKey	$\Theta(\log n)$	$\Theta(\log n)$	$\Theta(1)^*$	$O(\log n)^*$	$\Theta(1)$
Merge / Union	$\Theta(n)$	$\Omega(\log n)$	$\Theta(1)$	$\Theta(1)^*$	$\Theta(1)$

Фибоначчиевы кучи (Fibonacci heaps)

- **Фибоначчиевы кучи*** (*Fibonacci heaps*) эффективны, когда количество операций DeleteMin() и Delete(*x*) относительно мало по сравнению с количеством других операций (Min, Insert, Merge, DecreaseKey)
- Фибоначчиевы кучи нашли широкое применение в алгоритмах на графах (поиск кратчайшего пути в графе, поиск минимального остовного дерева)
- Например, в алгоритме Дейкстры фибоначчиевы кучи (min-heap) можно использовать для хранения вершин и быстрого уменьшения кратчайшего пути до них (DecreaseKey)

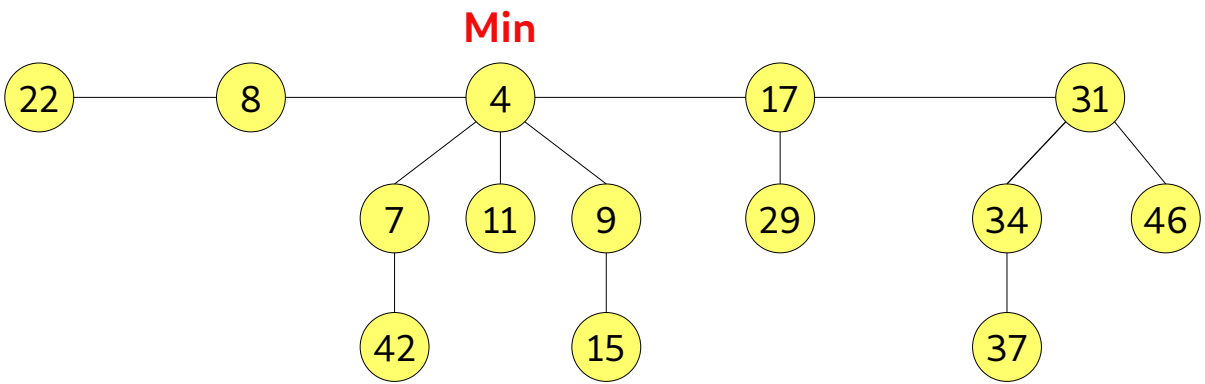
- **Авторы:** Michael L. Friedman, Robert E. Tarjan, 1984

* Friedman M. L., Tarjan R. E. **Fibonacci heaps and their uses in improved network optimization algorithms.** — Journal of the Association for Computer Machinery. — 1987, 34 (3). — pp. 596-615

Фибоначчиевы кучи (Fibonacci heaps)

- **Фибоначчиева куча** (*Fibonacci heap*) — это совокупность деревьев, которые удовлетворяют свойствам кучи (min-heap или max-heap)
- Деревья могут иметь различные степени (*degree*)
- Максимальная степень узла в фибоначчиевой куче из *n* элементов:

$$D(n) \leq \lfloor \log n \rfloor$$

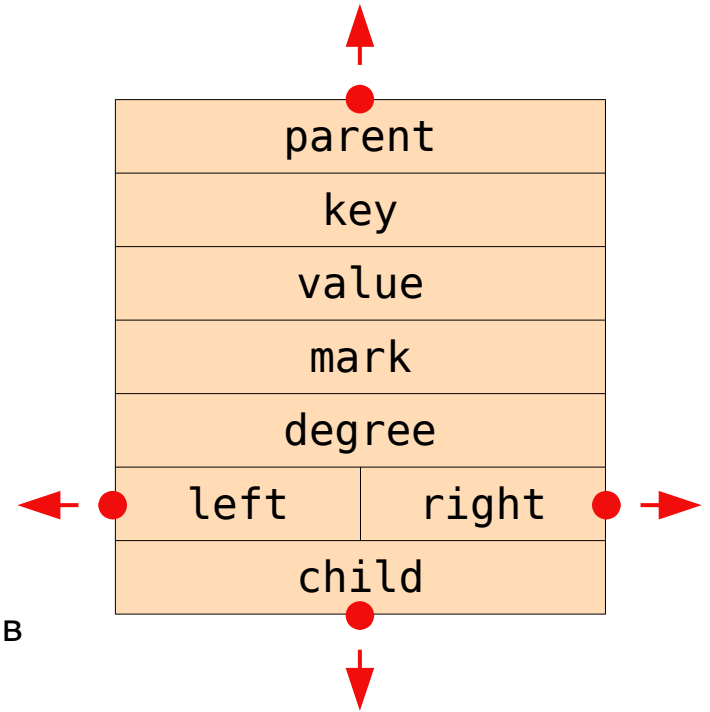


min-heap

(5 деревьев, 14 узлов)

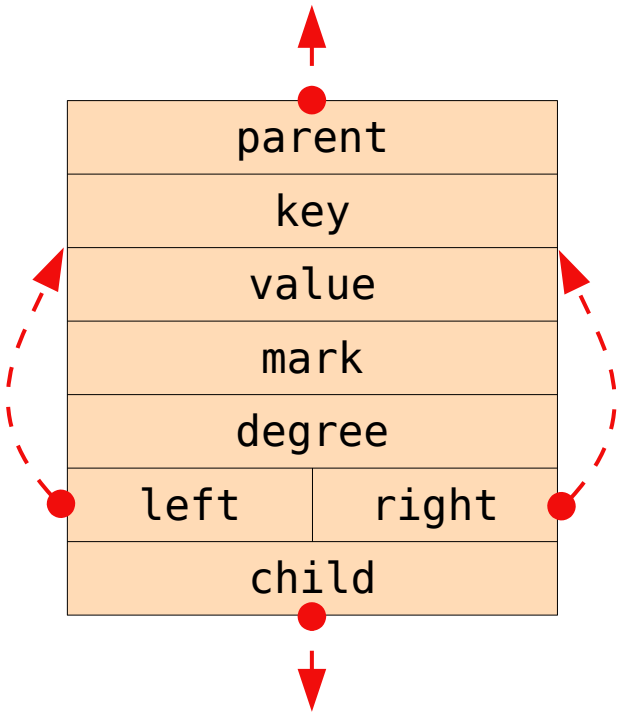
Узел фибоначчиевой кучи

- Дочерние узлы (включая корни) объединены в циклический дважды связный список (doubly linked list)
- Каждый узел фибоначчиевой кучи содержит следующие поля:
  - ***key*** — приоритет узла (вес, ключ)
  - ***value*** — данные
  - ***parent*** — указатель на родительский узел
  - ***child*** — указатель на один из дочерних узлов
  - ***left*** — указатель на левый сестринский узел
  - ***right*** — указатель на правый сестринский узел
  - ***degree*** — количество дочерних узлов
  - ***mark*** — индикатор, были ли потери узлом его дочерних узлов



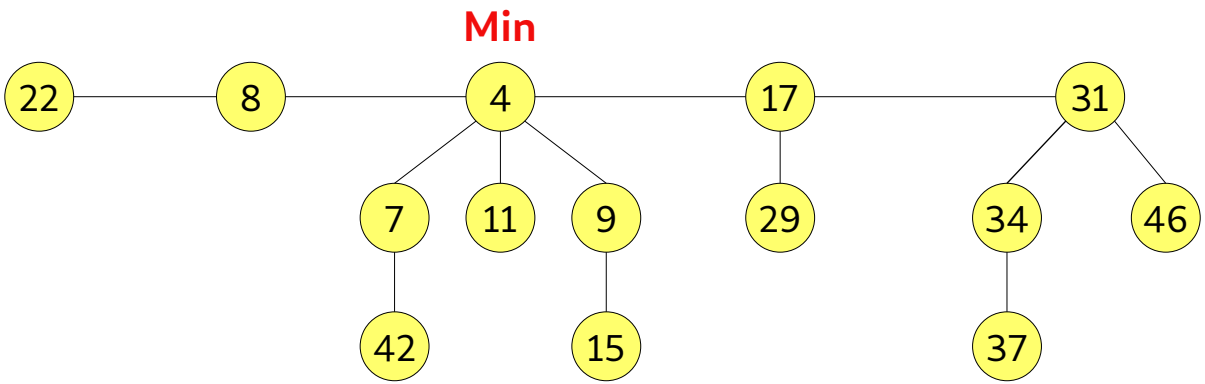
Узел фибоначчиевой кучи

- Если узел является единственным дочерним узлом, ***x.left = x.right = x***



Добавление узла в кучу (Insert)

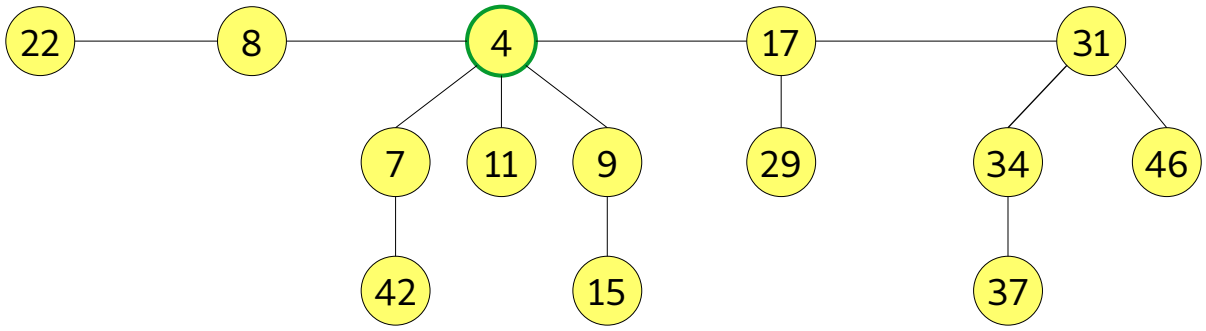
- Доступ к фибоначчиевой куче осуществляется по указателю на корень дерева с минимальным ключом (или с максимальным, в случае max-heap)



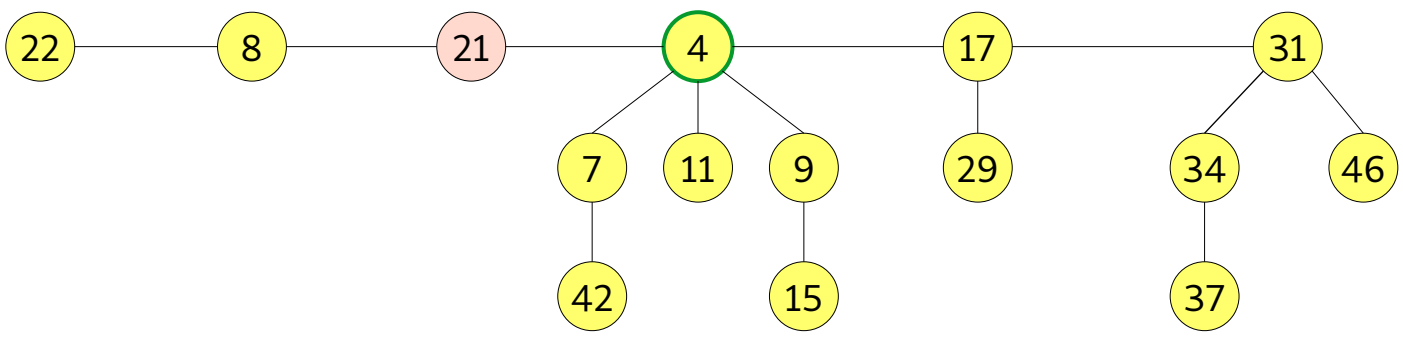
min-heap

(5 деревьев, 14 узлов)

Добавление узла в кучу (Insert)



Добавление узла с ключом 21: создаём новый узел (21) и помещаем его слева от узла с минимальным ключом (4)



Добавление узла в кучу (Insert)

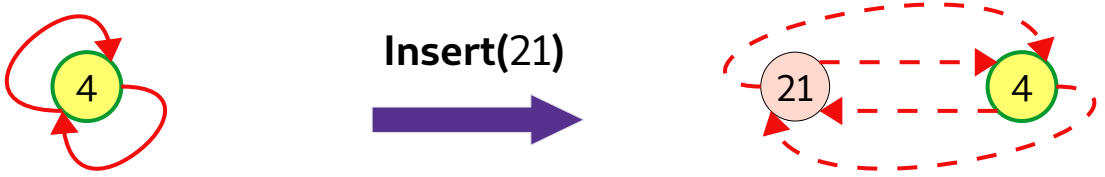
```
function FibHeapInsert(heap, key, value)
    node = AllocateMemory()
    node.key = key
    node.value = value
    node.degree = 0
    node.mark = false
    node.parent = NULL
    node.child = NULL
    node.left = node
    node.right = node
    /* Добавляем узел в список корней heap */
    FibHeapAddNodeToRootList(node, heap.min)
    if heap.min = NULL OR node.key < heap.min.key then
        heap.min = node
    end if
    heap.nnodes = heap.nnodes + 1
    return heap
end function
```

$T_{Insert} = O(1)$

Добавление узла в список корней

- Необходимо поместить новый узел *node* в список корней кучи *h*
- **Случай 1: список корней содержит одно дерево**

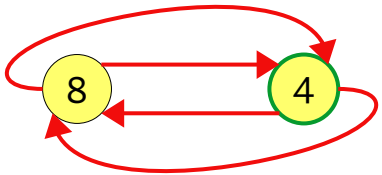
  Добавляем новый узел слева от корня дерева и корректируем циклические связи



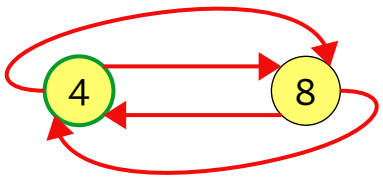
Добавление узла в список корней

- Необходимо поместить новый узел *node* в список корней кучи *h*
- **Случай 2: список корней содержит больше одного дерева**

  Добавляем новый узел слева от корня дерева и корректируем циклические связи

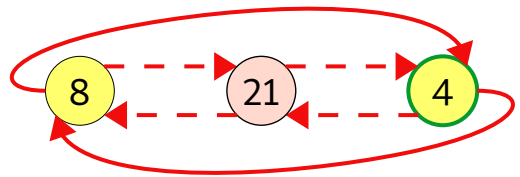


или



h.left ≠ h ≠ h.right

Insert(21**)**




```
lnode = h.left
h.left = node
node.right = h
node.left = lnode
lnode.right = node
```

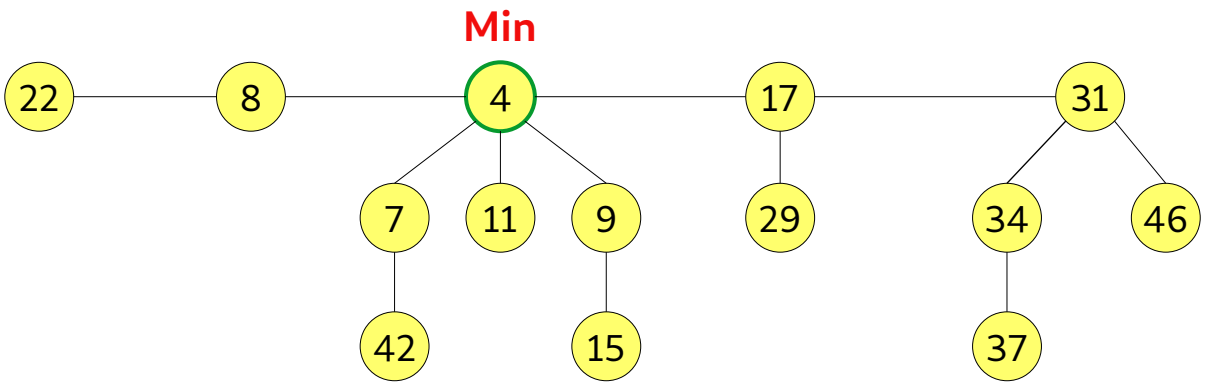
Добавление узла в список корней

```
function FibHeapAddNodeToRootList(node, h)
    if h = NULL then
        return
    if h.left = h then      /* Случай 1: список h содержит один корень */
        h.left = node
        h.right = node
        node.right = h
        node.left = h
    else                    /* Случай 2: список h содержит более одного корня */
        lnode = h.left
        h.left = node
        node.right = h
        node.left = lnode
        lnode.right = node
    end if
end function
```

$T_{AddNodeToRootList} = O(1)$

Поиск минимального узла (Min)

- В фибоначчиевой куче поддерживается указатель на корень дерева с минимальным ключом
- Поиск минимального узла выполняется за время $O(1)$



min-heap

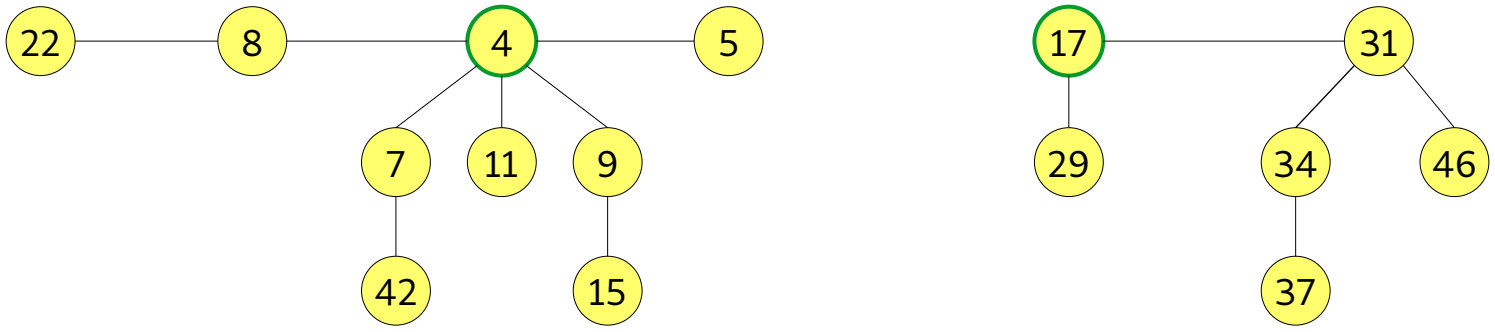
(5 деревьев, 14 узлов)

Поиск минимального узла (Min)

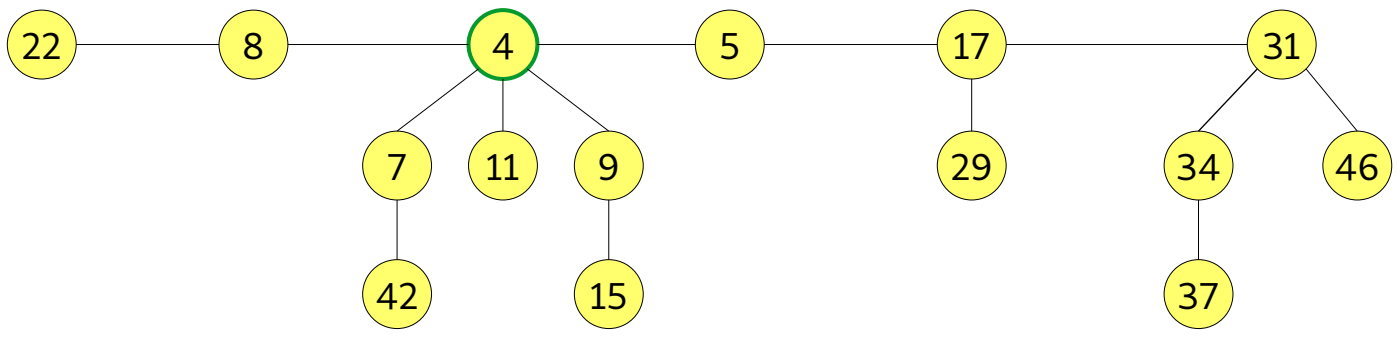
```
function FibHeapMin(heap)
    return heap.min
end function
```

$T_{Min} = O(1)$

Слияние фибоначчиевых куч (Union)



Слияние двух куч: объединяем списки корней, корректируем указатель на минимальный узел



Слияние фибоначчиевых куч (Union)

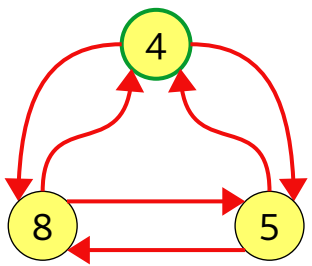
```
function FibHeapUnion(heap1, heap2)
    heap = AllocateMemory()
    heap.min = heap1.min
    FibHeapLinkLists(heap1.min, heap2.min)
    if (heap1.min = NULL) OR (heap2.min != NULL AND heap2.min.key < heap.min.key) then
        heap.min = heap2.min
    end if
    heap.nnodes = heap1.nnodes + heap2.nnodes
    FreeMemory(heap1)
    FreeMemory(heap2)
    return heap
end function
```

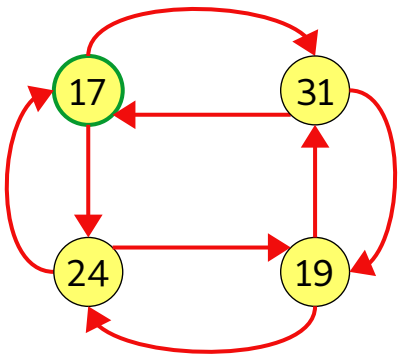
$T_{Union} = O(1)$

Слияние пары списков корней

- Имеется два списка корней — два двусвязных циклических списка
- Каждый список задан указателем на одну из вершин (корень дерева)
- Требуется слить два списка в один

Heap 1

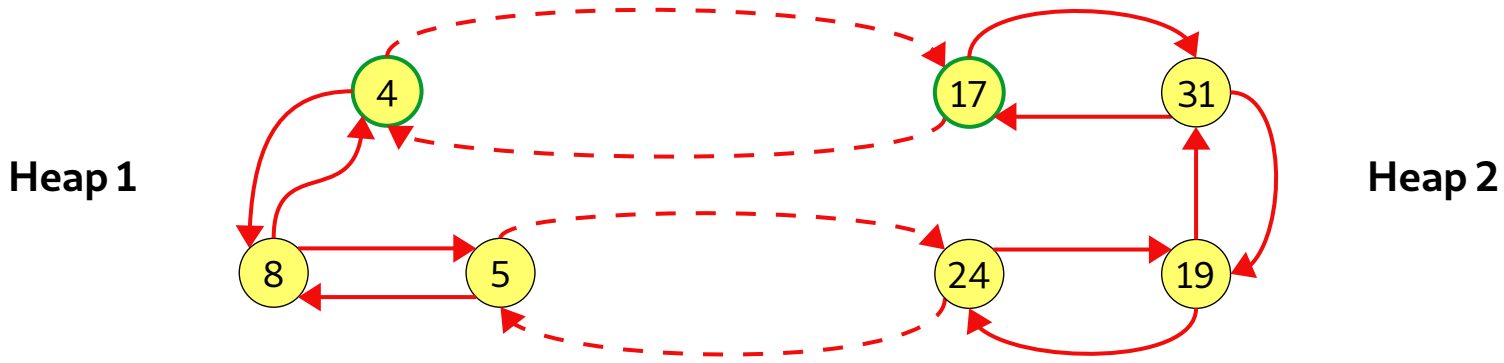




Heap 2

Слияние пары списков корней

- Имеется два списка корней — два двусвязных циклических списка
- Каждый список задан указателем на одну из вершин (корень дерева)
- Требуется слить два списка в один



- Требуется корректировка 4 указателей
- Объединение списков выполняется за время $O(1)$

Слияние пары списков корней

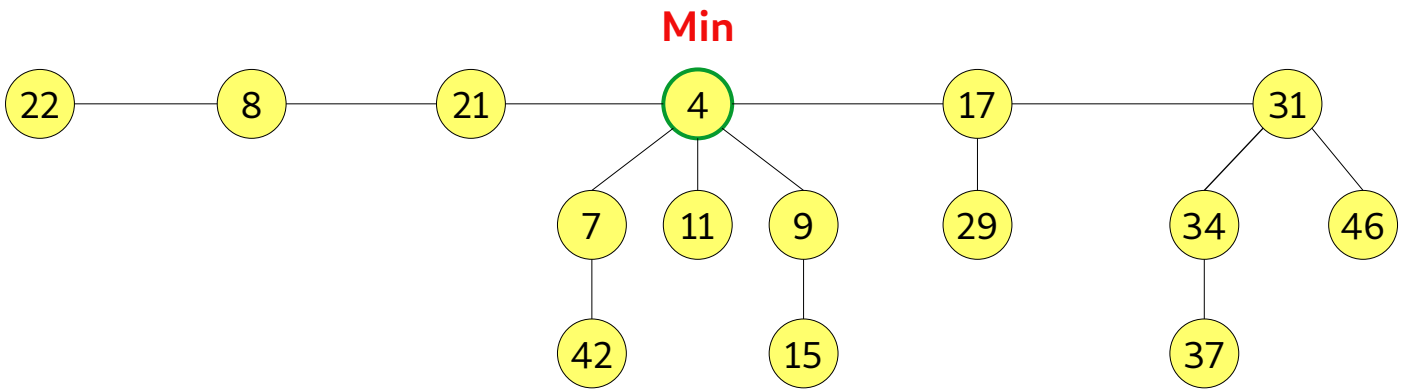
```
function FibHeapLinkLists(heap1, heap2)
    if heap1 = NULL OR heap2 = NULL then
        return

    left1 = heap1.left
    left2 = heap2.left
    left1.right = heap2
    heap2.left = left1
    heap1.left = left2
    left2.right = heap1
    return heap1
end function
```

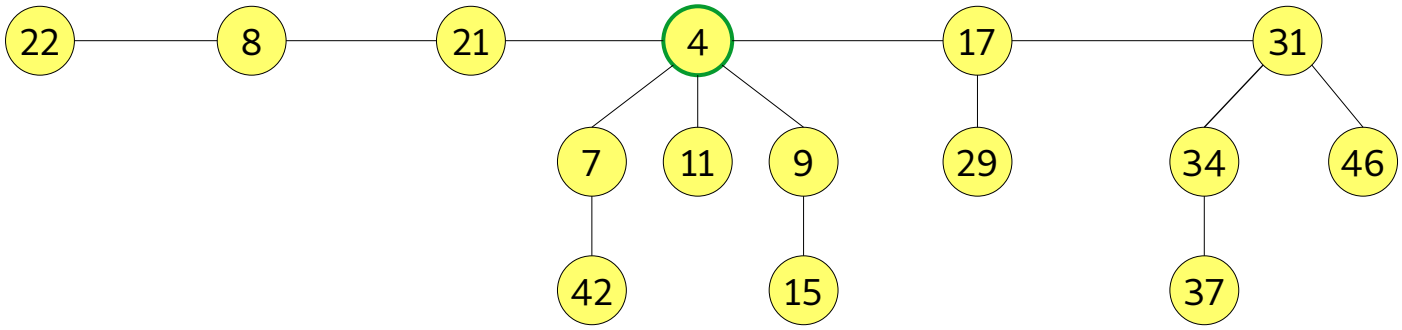
$T_{LinkLists} = O(1)$

Удаление минимального узла (DeleteMin)

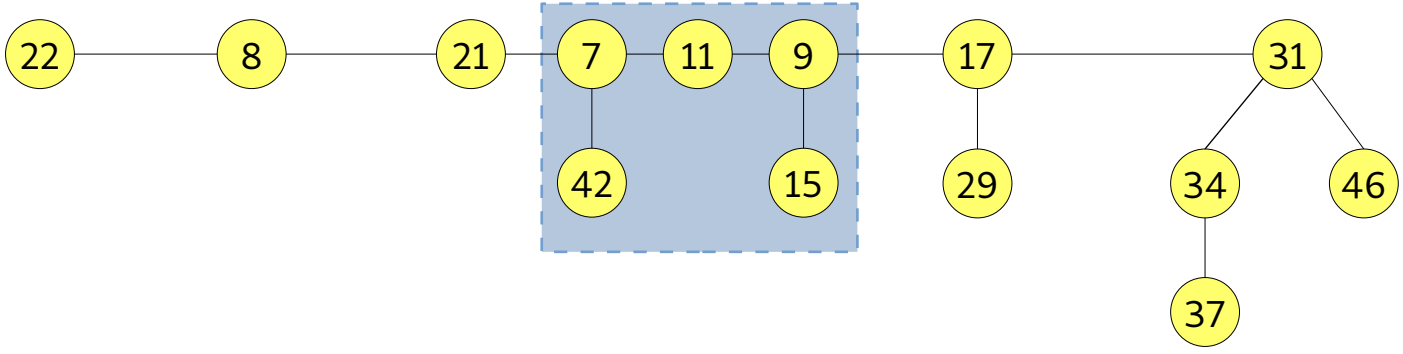
- Получаем указатель на минимальный узел *z*
- Удаляем из списка корней узел *z*
- Перемещаем в список корней все дочерние узлы элемента *z*
- Уплотняем список корней (Consolidate) — связываем корни деревьев одинаковой степени, пока в списке корней останется не больше одного дерева (корня) каждой степени



Удаление минимального узла (DeleteMin)



Удаление минимального элемента: поднимаем в список корней дочерние элементы минимального узла (4)

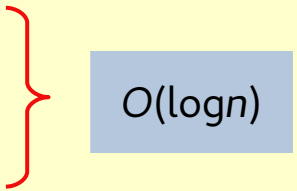


Удаление минимального узла (DeleteMin)

```
function FibHeapDeleteMin(heap)
    z = heap.min
    if z = NULL then
        return NULL
    for each x in z.child do
        FibHeapAddNodeToRootList(x, heap)     /* Добавляем дочерний узел x в список корней */
        x.parent = NULL
    end for
    FibHeapRemoveNodeFromRootList(z, heap)    /* Удаляем z из списка корней */
    if z = z.right then
        heap.min = NULL
    else
        heap.min = z.right
        FibHeapConsolidate(heap)
    end if
    heap.nnodes = heap.nnodes - 1
    return z
end function
```

Удаление минимального узла (DeleteMin)

```
function FibHeapDeleteMin(heap)
    z = heap.min
    if z = NULL then
        return NULL
    for each x in z.child do
        FibHeapAddNodeToRootList(x, heap)
        x.parent = NULL
    end for
    FibHeapRemoveNodeFromRootList(z, heap)
    if z = z.right then
        heap.min = NULL
    else
        heap.min = z.right
        FibHeapConsolidate(heap)
    end if
    heap.nnodes = heap.nnodes - 1
    return z
end function
```



Уплотнение списка корней (Consolidate)

- **Уплотнение списка корней выполняется до тех пор, пока все корни не будут иметь различные значения поля *degree***
  1. Находим в списке корней два корня *x* и *y* с одинаковой степенью (*x.degree* = *y.degree*) такие, что *x.key* ≤ *y.key*
  2. Делаем *y* дочерним узлом *x* (или наоборот, в случае max-heap); поле *x.degree* увеличивается, а пометка *y.mark* снимается (если была установлена)
- Процедура Consolidate использует вспомогательный массив указателей $A[0, 1, \ldots, D(n)]$, $D(n) \leq \lfloor \log n \rfloor$
- Если $A[i] = y$, то корень *y* имеет степень *i*

Уплотнение списка корней (Consolidate)

```
function FibHeapConsolidate(heap)
    for i = 0 to D(heap.nnodes) do
        A[i] = NULL
    for each w in heap.min do
        x = w
        d = x.degree
        while A[d] != NULL do
            y = A[d]
            if x.key > y.key then
                FibHeapSwap(x, y)
            FibHeapLink(heap, y, x)
            A[d] = NULL
            d = d + 1
        end while
        A[d] = x
    end for
```

$O(\log n)$

/* Цикл по всем узлам списка корней */

/* Степень y совпадает со степенью x */

/* Обмениваем x и y */

Уплотнение списка корней (Consolidate)

```
    /* Находим минимальный узел */
    heap.min = NULL
    for i = 0 to D(heap.nnodes) do
        if A[i] != NULL then
            FibHeapAddNodeToRootList(A[i], heap)   /* Добавляем A[i] в список корней */
            if heap.min = NULL OR A[i].key < heap.min.key then
                heap.min = A[i]
            end if
        end if
     end for
end function
```

```
function D(n)
    return floor(log(2, n))
end function
```

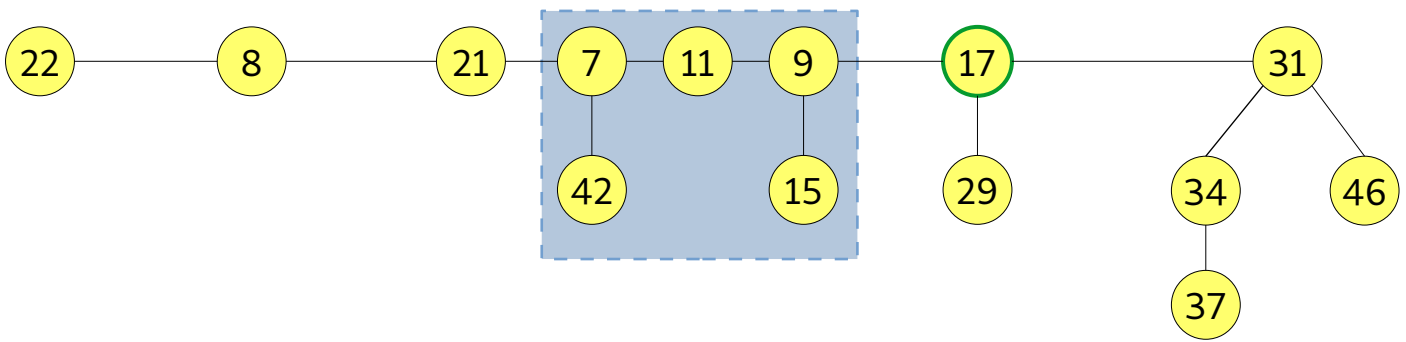
Уплотнение списка корней (Consolidate)

```
function FibHeapLink(heap, y, x)
    x.degree = x.degree + 1
    /* Делаем y дочерним узлом x */
    FibHeapRemoveNodeFromRootList(y, heap)
    y.parent = x
    FibHeapAddNodeToRootList(y, x.child)
    y.mark = FALSE
end function
```

$T_{Link} = O(1)$

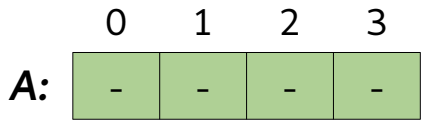
Уплотнение списка корней (Consolidate)

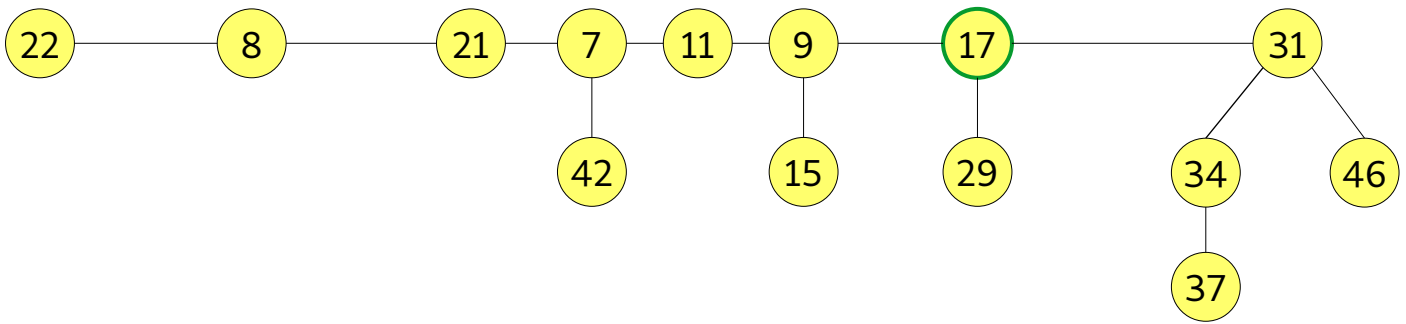
- В список корней подняли дочерние элементы минимального узла (4), минимальный узел из списка удалён (см. DeleteMin)
- Текущим минимальным узлом стал узел (17) — правый сосед узла (4): *heap.min* = *z.right* (причём, *z.right* может и не являться минимальным узлом)



Уплотнение списка корней (Consolidate)

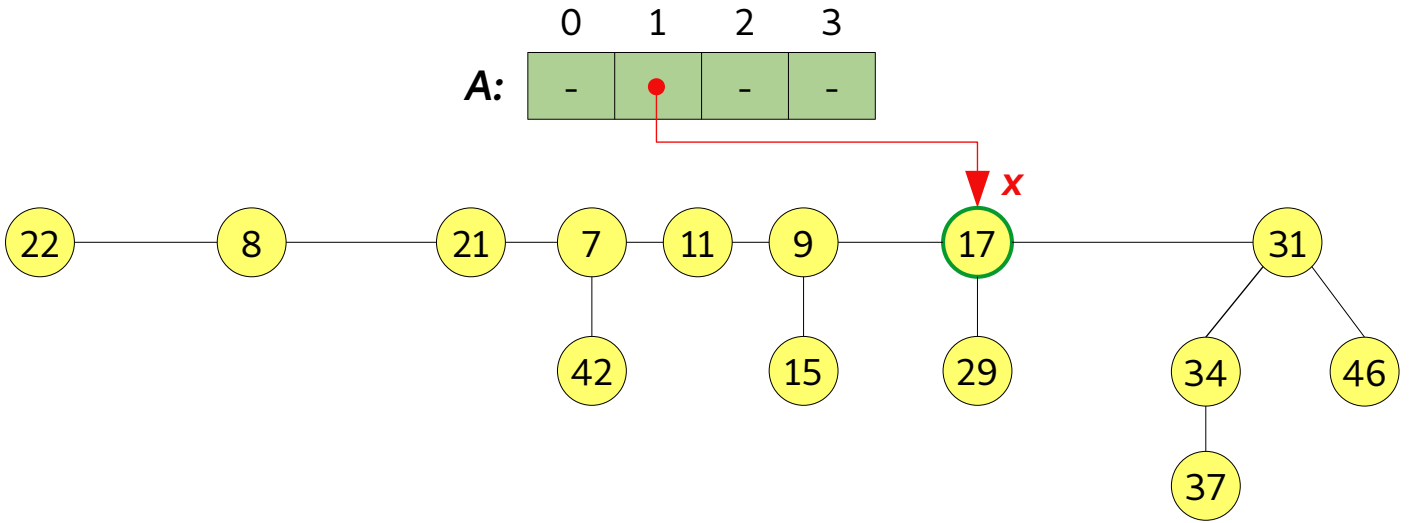
- В куче n = 14 узлов
- Максимальная степень корня дерева $D(n) \le \lfloor \log n \rfloor = 3$
- Массив $A[0, 1, ..., D(n)] = A[0, ..., 3]$
- $A[i]$ = NULL





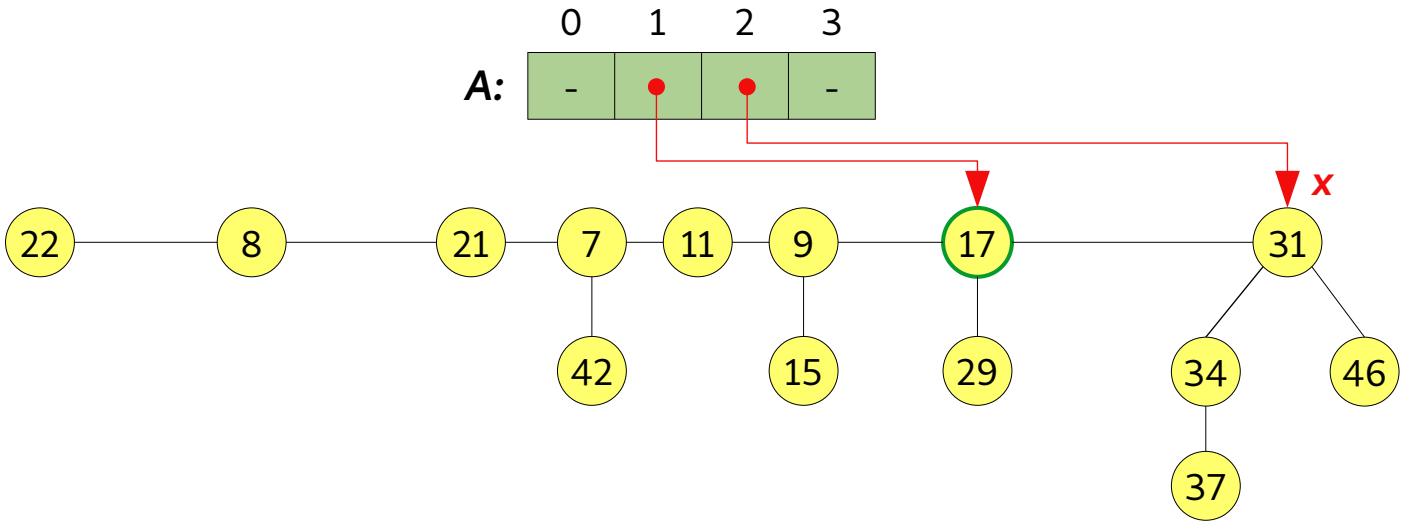
Уплотнение списка корней (Consolidate)

- Обход списка корней начинаем с узла heap.min (узел **17**)
- Степень узла 17 равна 1, ищем корни степени 1
- *A*[1] = NULL: в списке нет других корней степени 1



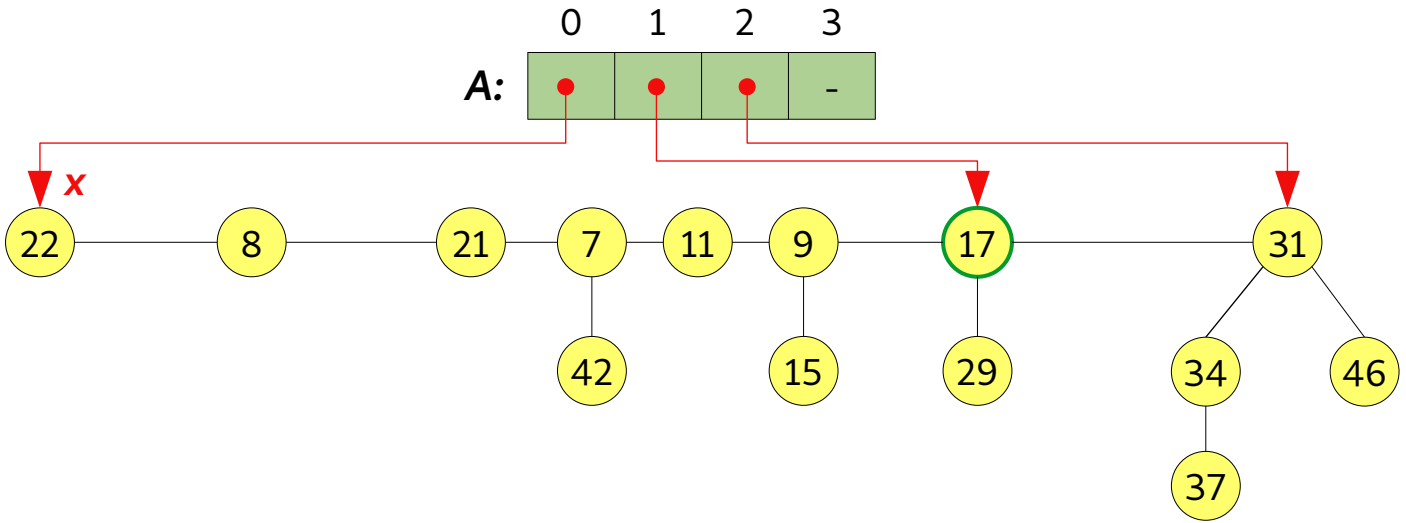
Уплотнение списка корней (Consolidate)

- Переходим к следующему элементу — узлу **31**
- Степень узла 31 равна 2, ищем корни степени 2
- *A*[2] = NULL, в списке нет других корней степени 2



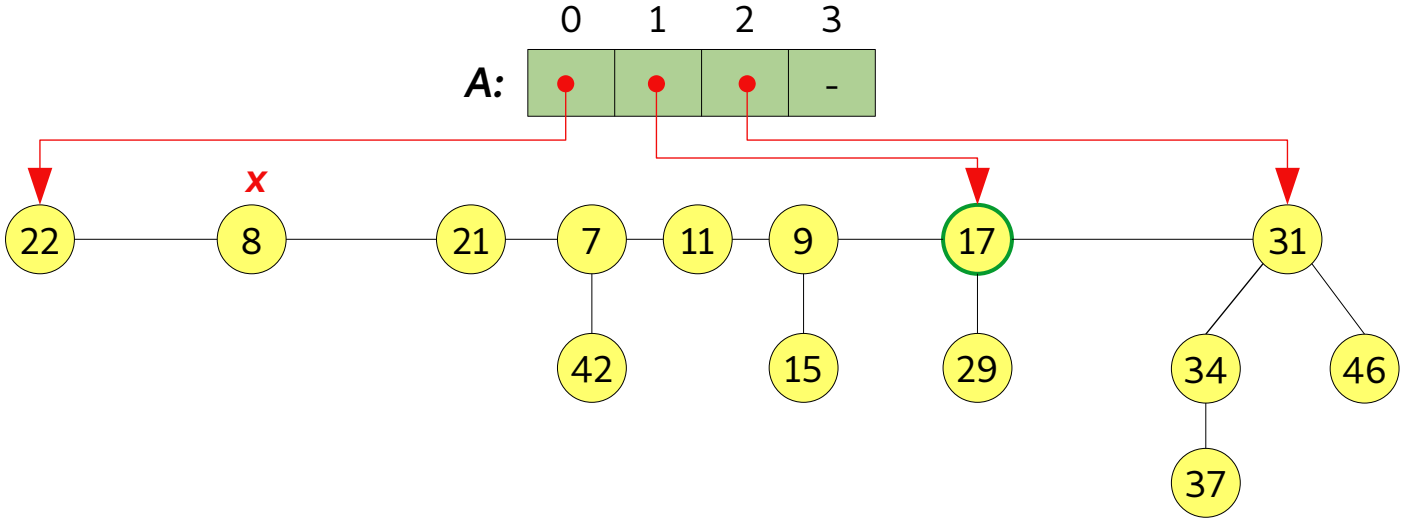
Уплотнение списка корней (Consolidate)

- Переходим к следующему элементу — узлу **22**
- Степень узла 22 равна 0, ищем корни степени 0
- *A*[0] = NULL, в списке нет других корней степени 0



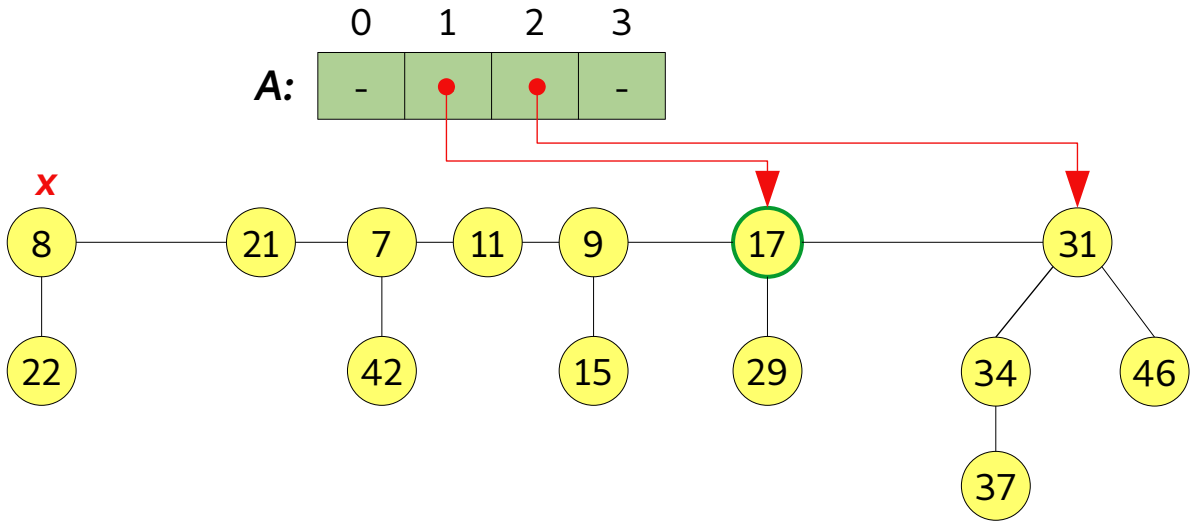
Уплотнение списка корней (Consolidate)

- Переходим к следующему элементу — узлу **8**
- Степень узла 8 равна 0, ищем корни степени 0
- *A*[0] = (22): узел 22 также имеет степень 0



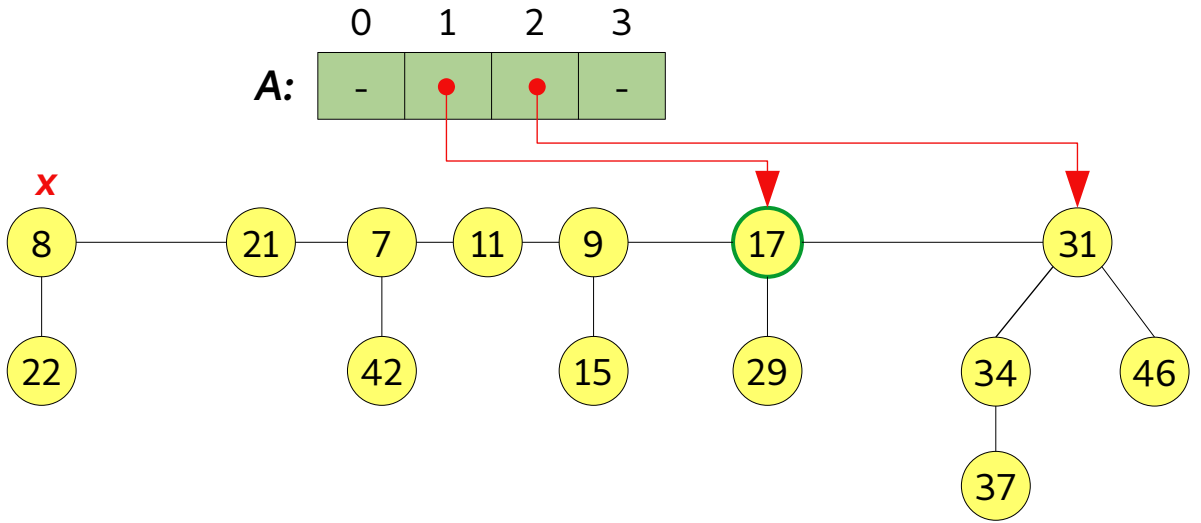
Уплотнение списка корней (Consolidate)

- Переходим к следующему элементу — узлу **8**
- Степень узла 8 равна 0, ищем корни степени 0
- *A*[0] = (22): узел 22 также имеет степень 0 — **делаем 22 дочерним узлом элемента 8 (8 < 22)**
- *A*[0] = NULL



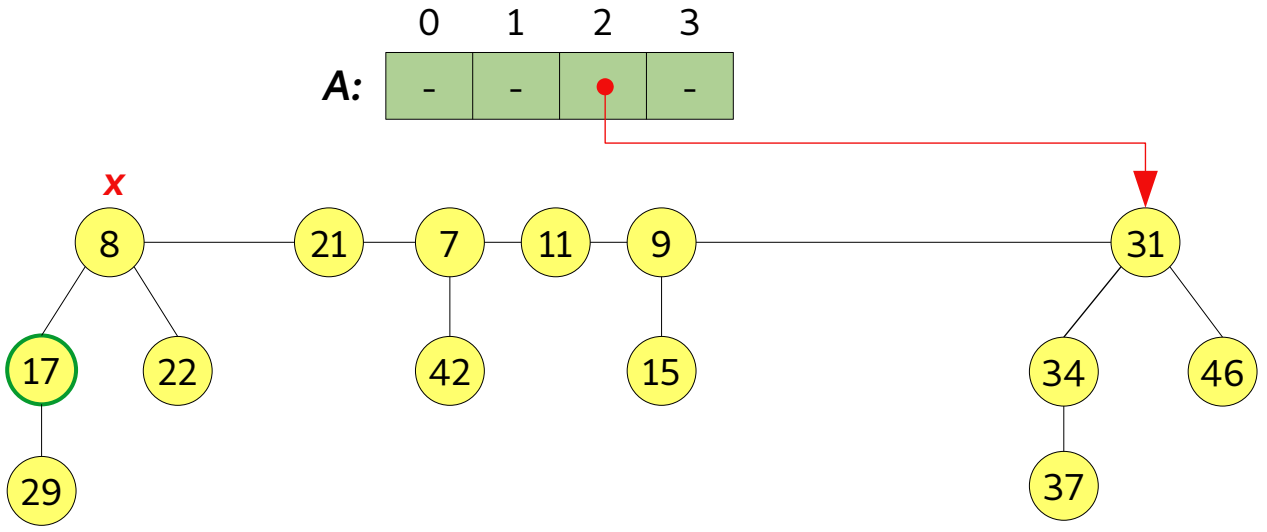
Уплотнение списка корней (Consolidate)

- Степень узла **8** увеличена до 1, ищем корни степени 1
- *A*[1] = (17): узел 17 также имеет степень 1



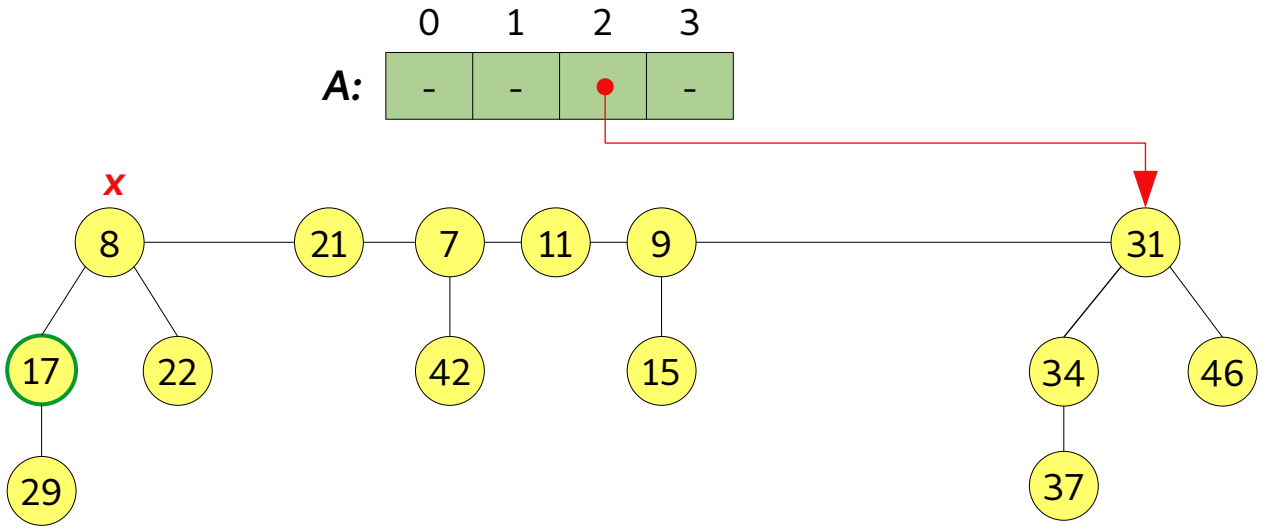
Уплотнение списка корней (Consolidate)

- Степень узла **8** увеличена до 1, ищем корни степени 1
- *A*[1] = (17): узел 17 также имеет степень 1
- **Делаем 17 дочерним узлом элемента 8 (8 < 17)**
- *A*[1] = NULL



Уплотнение списка корней (Consolidate)

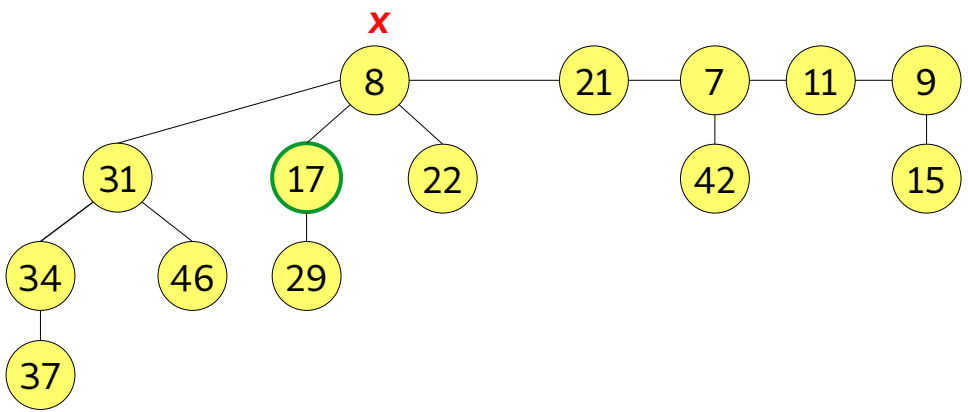
- Степень узла **8** увеличена до 2, ищем корни степени 2
- *A*[2] = (31): узел 31 также имеет степень 2



Уплотнение списка корней (Consolidate)

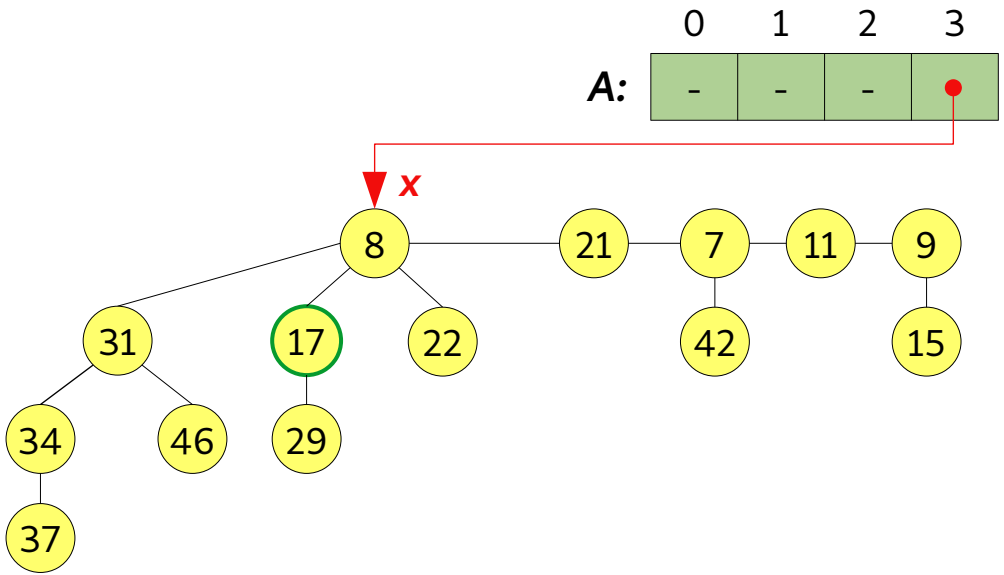
- Степень узла **8** увеличена до 2, ищем корни степени 2
- *A*[2] = (31): узел 31 также имеет степень 2
- **Делаем 31 дочерним узлом элемента 8 (8 < 31)**
- *A*[2] = NULL

	0	1	2	3
A:	-	-	-	-



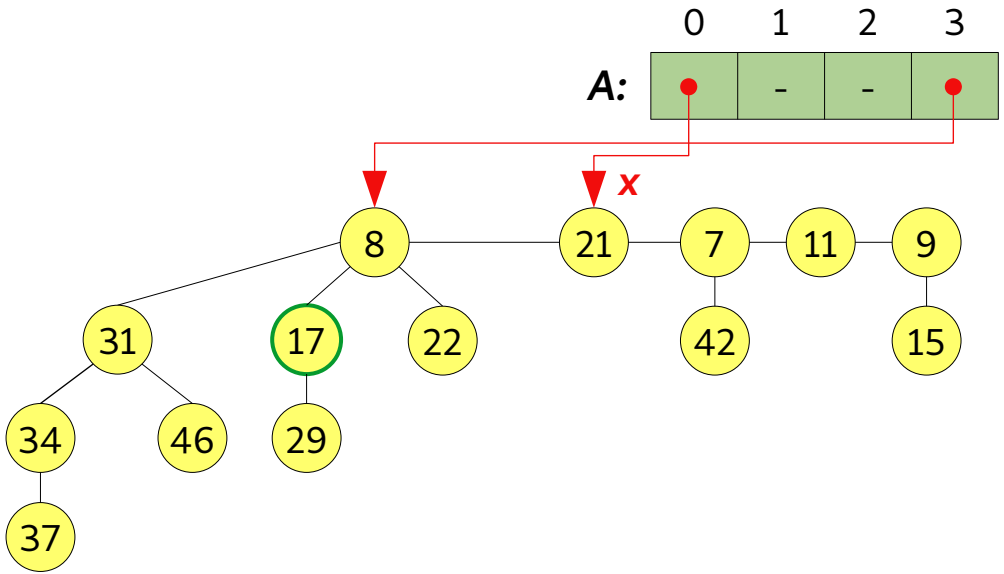
Уплотнение списка корней (Consolidate)

- Степень узла **8** увеличена до 3, ищем корни степени 3
- *A*[3] = NULL
- Устанавливаем указатель *A*[3] на узел 8



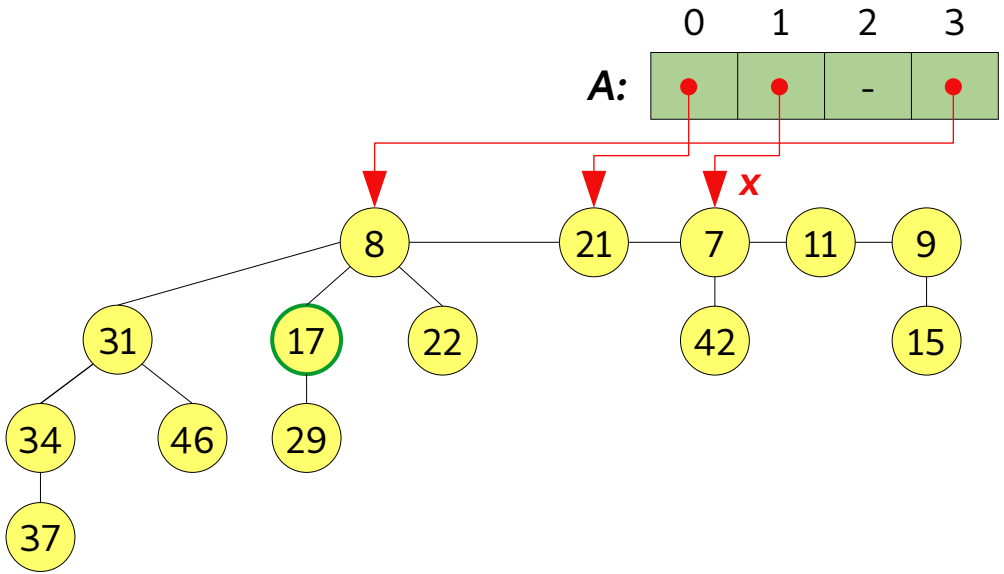
Уплотнение списка корней (Consolidate)

- Переходим к следующему элементу — узлу **21**
- Степень узла 21 равна 0, ищем корни степени 0
- *A*[0] = NULL, устанавливаем *A*[0] = (21)



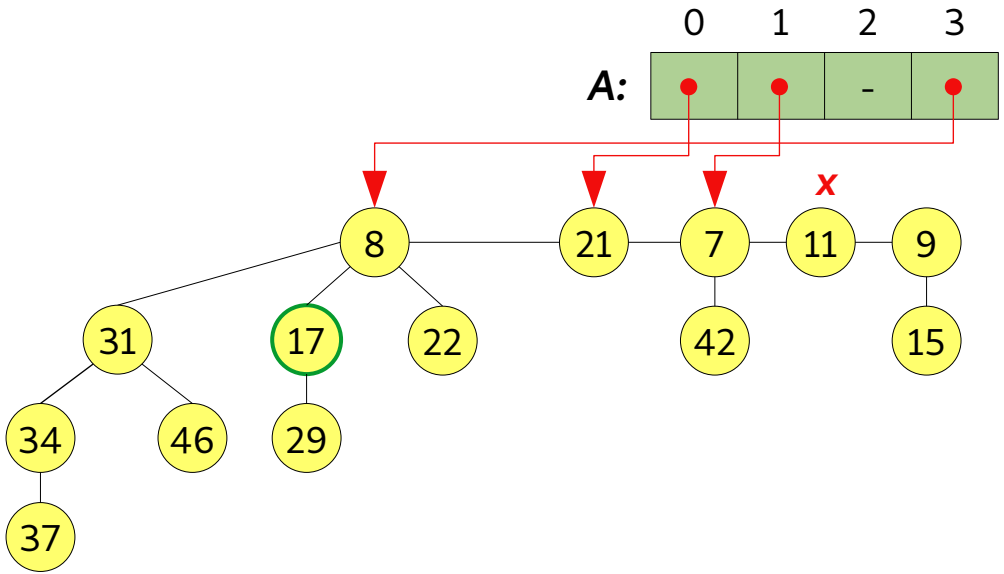
Уплотнение списка корней (Consolidate)

- Переходим к следующему элементу — узлу **7**
- Степень узла 7 равна 1, ищем корни степени 1
- *A*[1] = NULL, устанавливаем *A*[1] = (7)



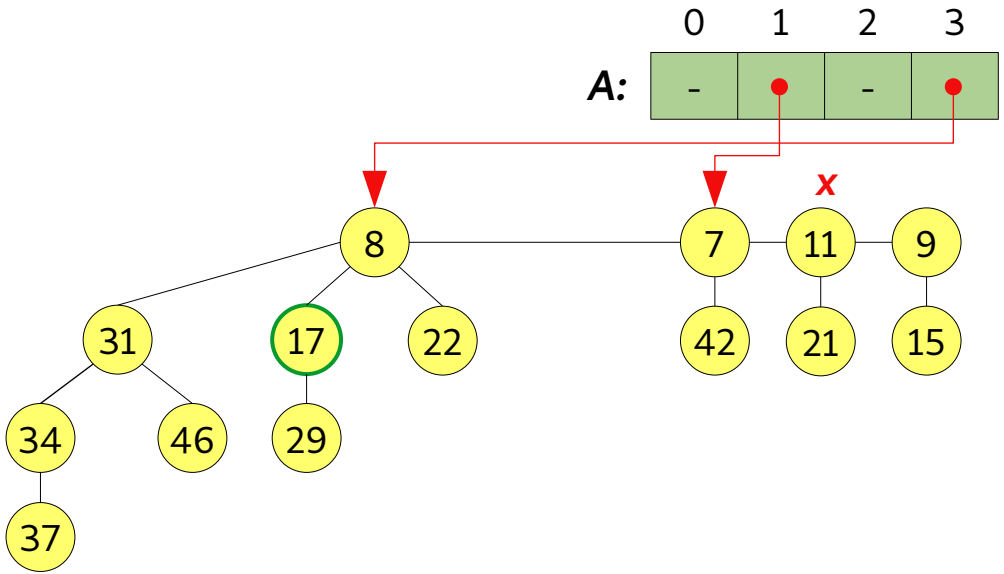
Уплотнение списка корней (Consolidate)

- Переходим к следующему элементу — узлу **11**
- Степень узла 11 равна 0, ищем корни степени 0
- *A*[0] = (21): узел 21 также имеет степень 0



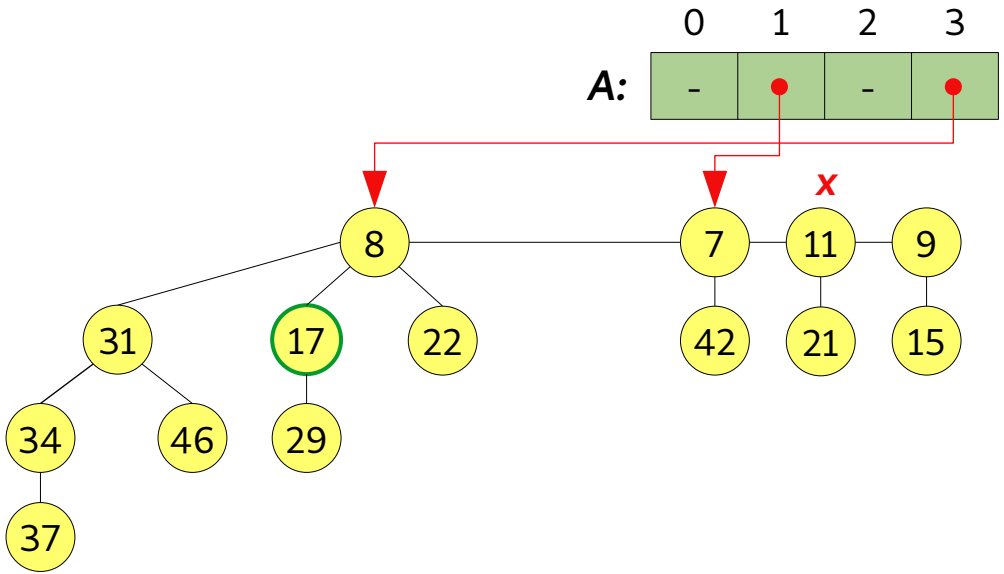
Уплотнение списка корней (Consolidate)

- Переходим к следующему элементу — узлу **11**
- Степень узла 11 равна 0, ищем корни степени 0
- *A*[0] = (21): узел 21 также имеет степень 0 — **делаем 21 дочерним узлом элемента 11 (11 < 22)**



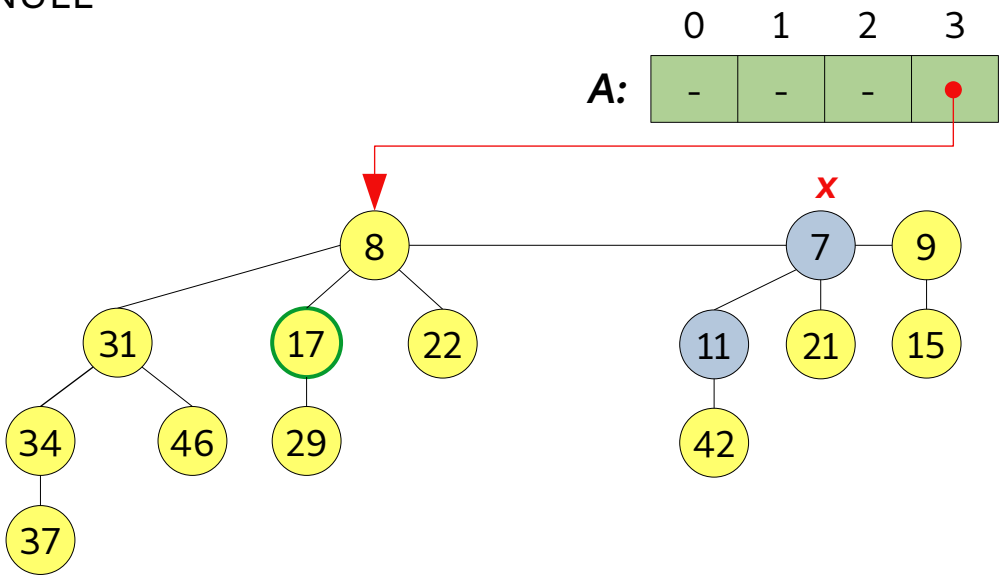
Уплотнение списка корней (Consolidate)

- Степень узла **11** увеличена до 1, ищем корни степени 1
- *A*[1] = (7): узел 7 также имеет степень 1



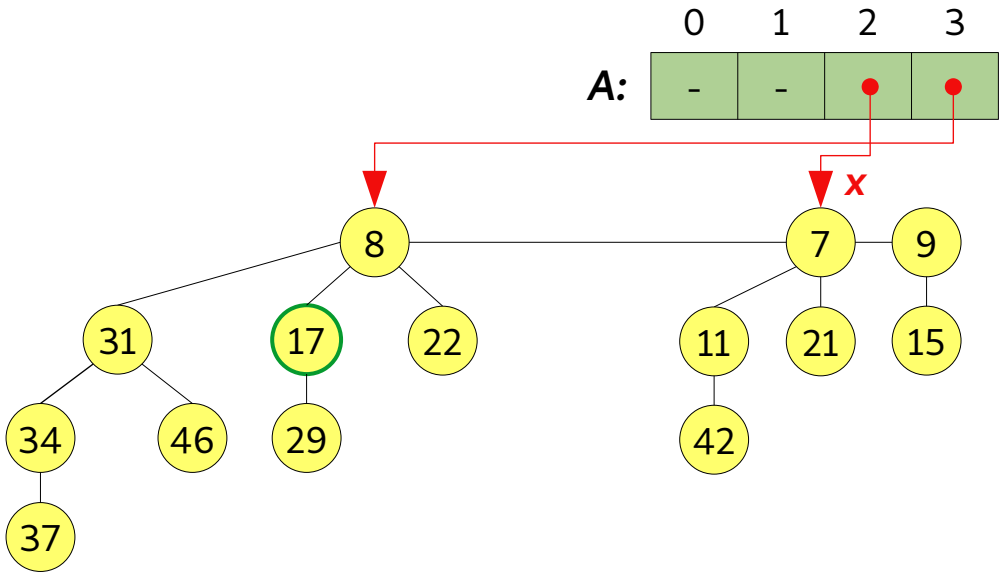
Уплотнение списка корней (Consolidate)

- Степень узла **11** увеличена до 1, ищем корни степени 1
- *A*[1] = (7): узел 7 также имеет степень 1
- **11 > 7: обмениваем узлы 11 и 7, делаем 11 дочерним узлом элемента 7**
- *A*[1] = NULL



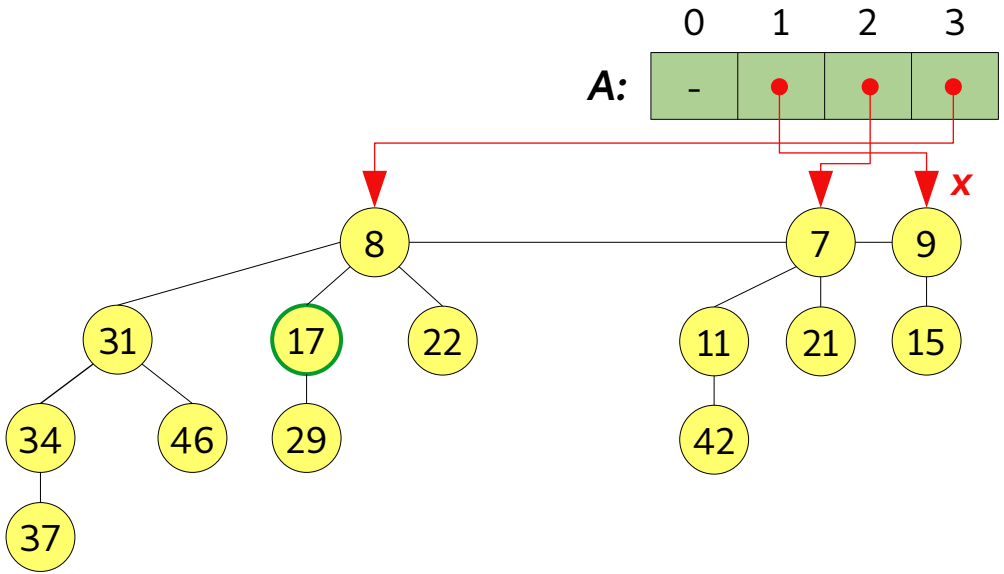
Уплотнение списка корней (Consolidate)

- Степень узла **7** увеличена до 2, ищем корни степени 2
- *A*[2] = NULL
- Устанавливаем *A*[2] = (7)



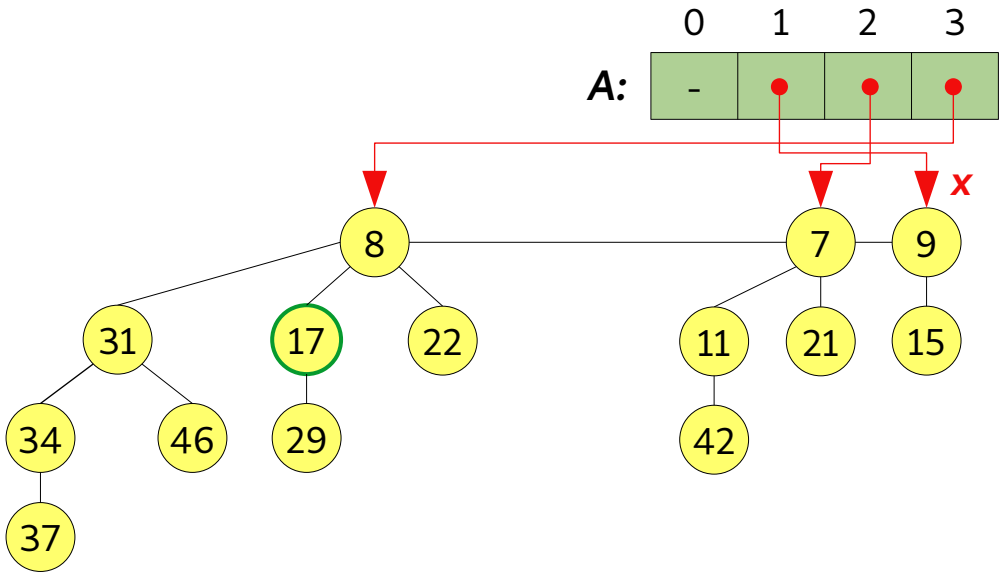
Уплотнение списка корней (Consolidate)

- Переходим к следующему узлу — узлу **9**
- Степень узла 9 равна 1, ищем корни степени 1
- *A*[1] = NULL, устанавливаем *A*[1] = (9)



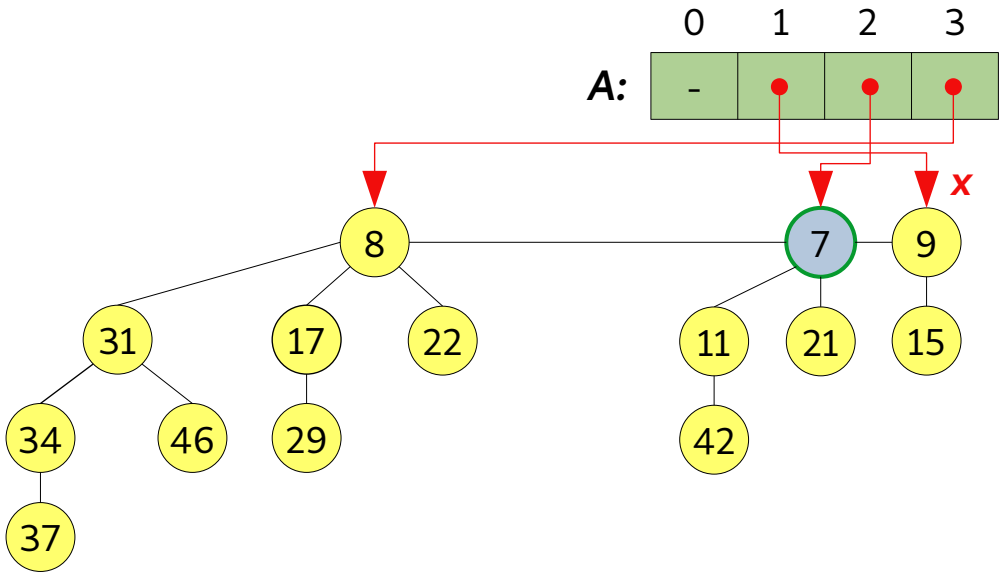
Уплотнение списка корней (Consolidate)

- **Обход списка корней завершён**
- Все корни имеют различные степени: 3, 2, 1



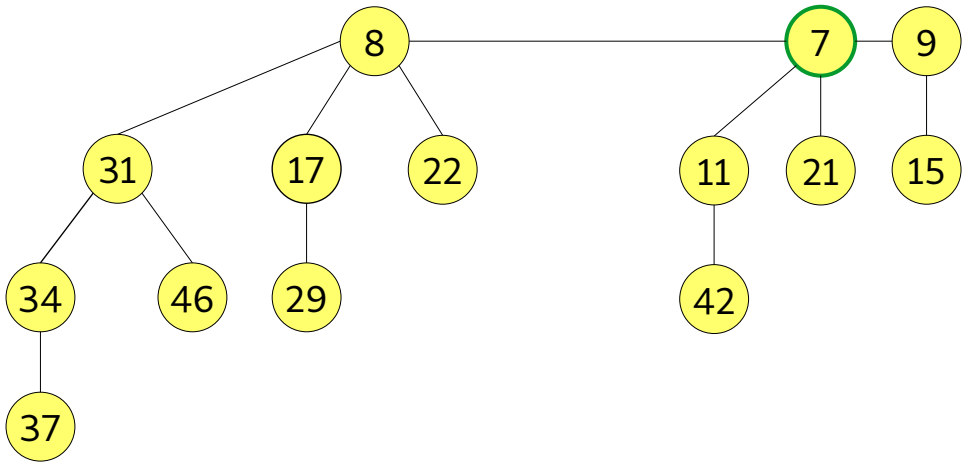
Уплотнение списка корней (Consolidate)

- **Необходимо выполнить поиск минимального узла (установить heap.min)**
- Проходим по указателям массива $A[0, 1, ..., D(n)] = A[0, ..., 3]$ и запоминаем указатель на корень с минимальным ключом (требуется $O(\log n)$ шагов)



Уменьшение ключа (DecreaseKey)

- Изменяем ключ заданного узла на новое значение
- Если нарушены свойства кучи (min-heap), ключ текущего узла меньше ключа родителя, то осуществляем восстановление свойств кучи



Уменьшение ключа (DecreaseKey)

```
function FibHeapDecreaseKey(heap, x, newkey)
    if newkey > x.key then
        return     /* Новый ключ больше текущего значения ключа */
    x.key = newkey
    y = x.parent
    if y != NULL AND x.key < y.key then
        /* Нарушены свойства min-heap: ключ родителя больше */
        /* Вырезаем x и переносим его в список корней */
        FibHeapCut(heap, x, y)
        FibHeapCascadingCut(heap, y)
    end if
    /* Корректируем указатель на минимальный узел */
    if x.key < heap.min.key then
        heap.min = x
end function
```

Уменьшение ключа (DecreaseKey)

```
function FibHeapCut(heap, x, y)
    /* Удаляем x из списка дочерних узлов y */
    FibHeapRemoveNodeFromRootList(x, y)
    y.degree = y.degree - 1
    /* Добавляем x в список корней кучи heap */
    FibHeapAddNodeToRootList(x, heap)
    x.parent = NULL
    x.mark = FALSE
end function
```

$T_{Cut} = O(1)$

- Функция **FibHeapCut** вырезает связь между *x* и его родительским узлом *y*, делая *x* корнем

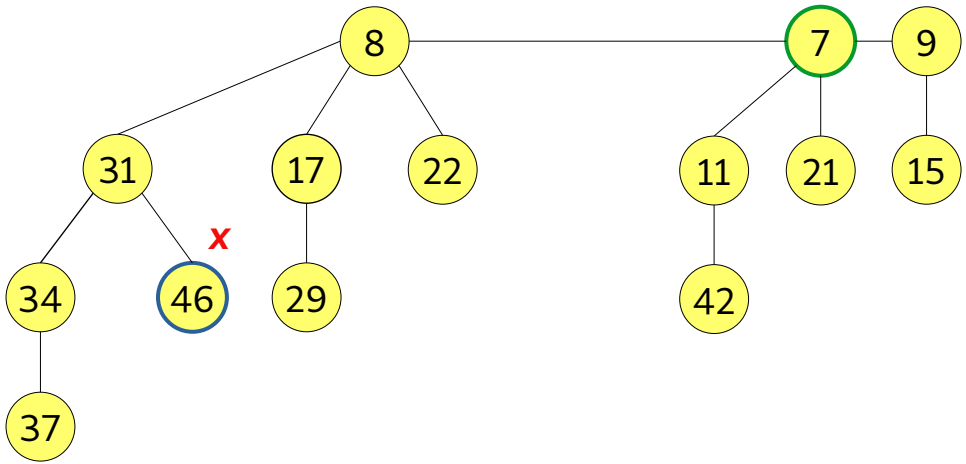
Уменьшение ключа (DecreaseKey)

```
function FibHeapCascadingCut(heap, y)
    z = y.parent
    if z = NULL then
        return
    if y.mark = FALSE then
        y.mark = TRUE
    else
        FibHeapCut(heap, y, z)
        FibHeapCascadingCut(heap, z)
    end if
end function
```

- Функция **FibHeapCascadingCut** реализует каскадное вырезание над *y*

Уменьшение ключа (DecreaseKey)

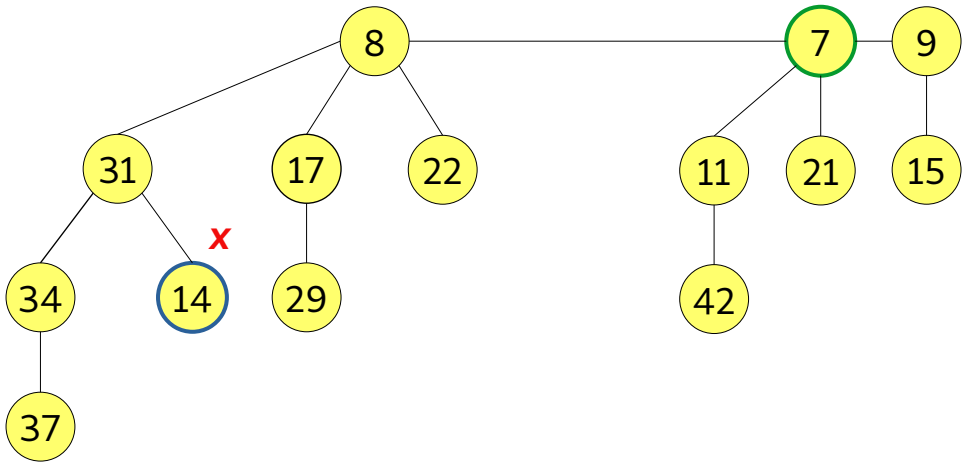
Изменим ключ 46 до значения 14



Уменьшение ключа (DecreaseKey)

Изменим ключ 46 до значения 14

- Устанавливаем в узле 46 новый ключ 14
- Нарушены свойства кучи min-heap: 14 < 31

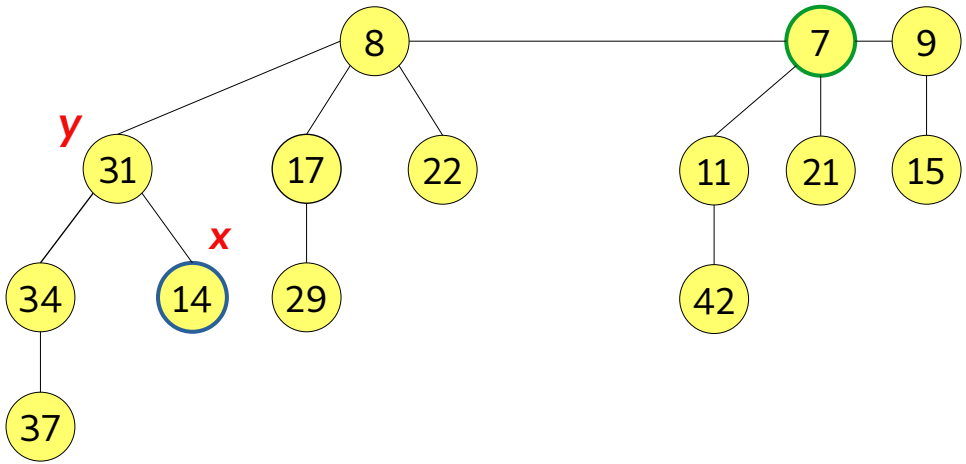


Уменьшение ключа (DecreaseKey)

Изменим ключ 46 до значения 14

- Устанавливаем в узле 46 новый ключ 14
- Нарушены свойства кучи min-heap: 14 < 31

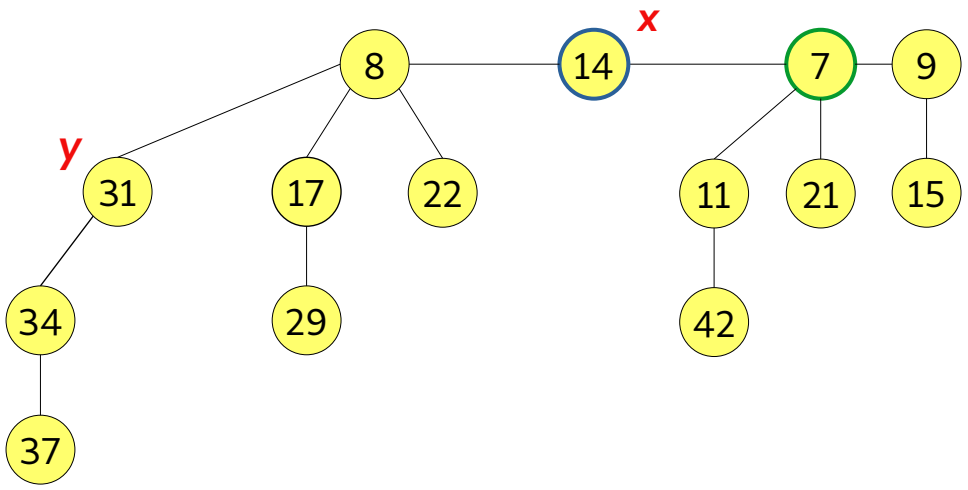
  Выполняем **FibHeapCut(<14>, <31>), FibHeapCascadingCut(<31>)**



Уменьшение ключа (DecreaseKey)

Выполняем **FibHeapCut(<14>, <31>)**

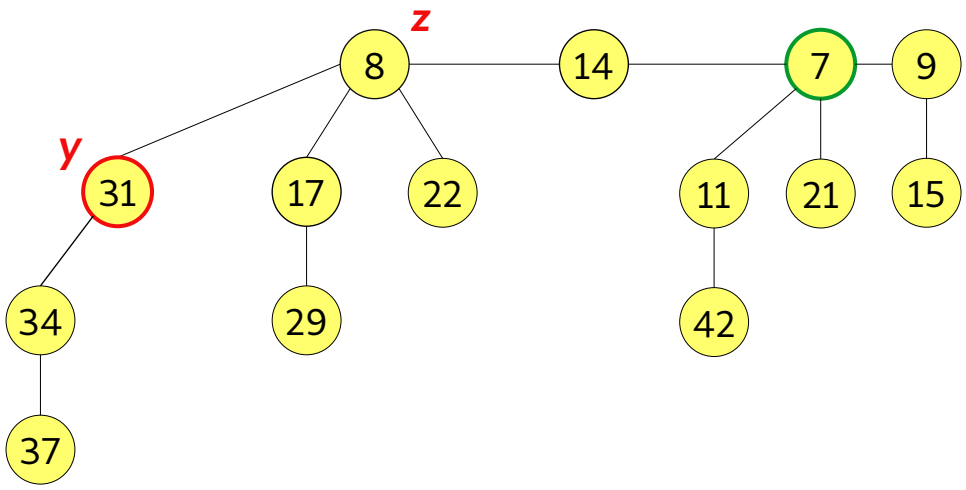
- Переносим <14> в список корней кучи
- Устанавливаем <14>.*mark* = FALSE



Уменьшение ключа (DecreaseKey)

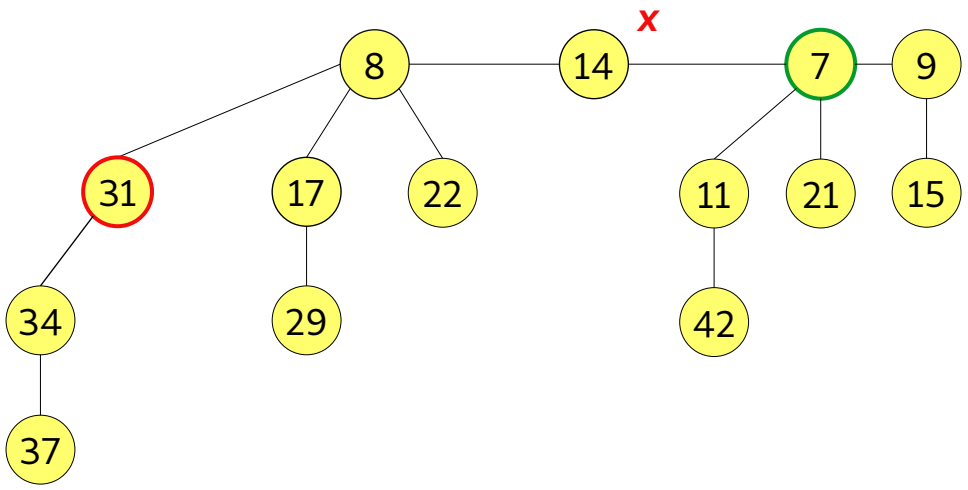
Выполняем **FibHeapCascadingCut(<31>)**

- Изменяем значение <31>.*mark* с FALSE на **TRUE**



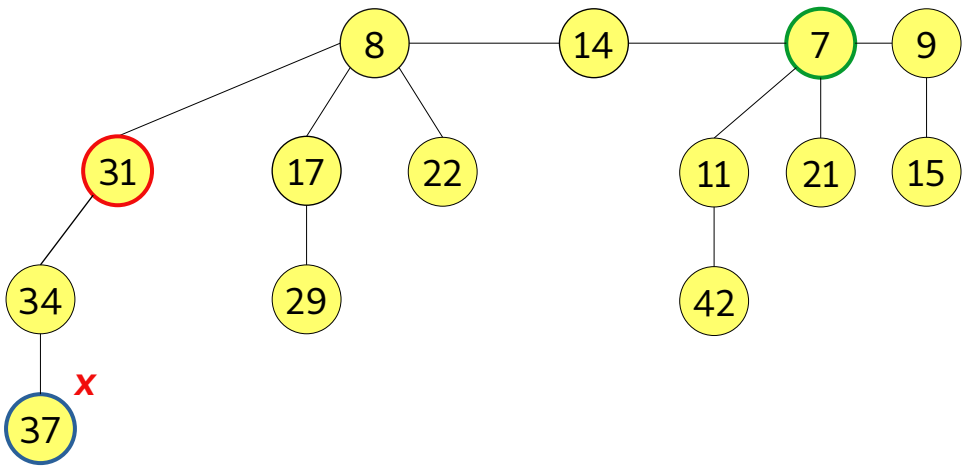
Уменьшение ключа (DecreaseKey)

- Корректируем указатель на минимальный элемент
- 14 > 7, указатель не изменяется



Уменьшение ключа (DecreaseKey)

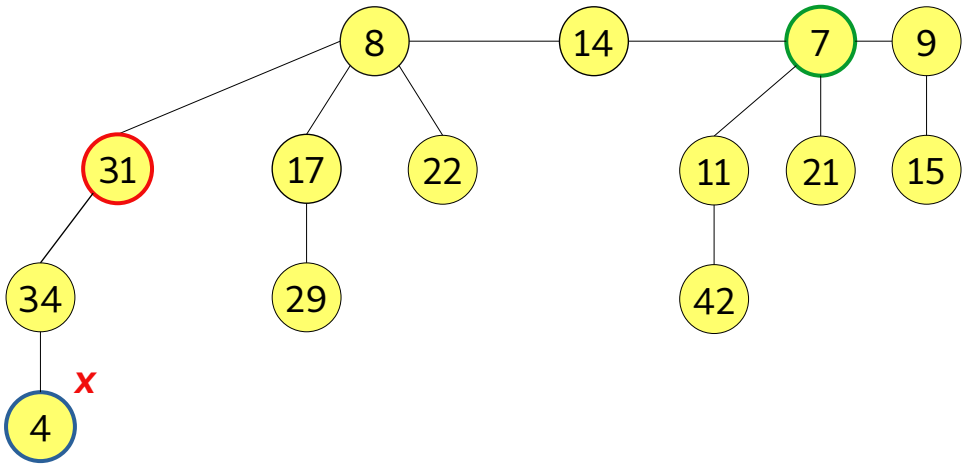
Изменим ключ 37 до значения 4



Уменьшение ключа (DecreaseKey)

Изменим ключ 37 до значения 4

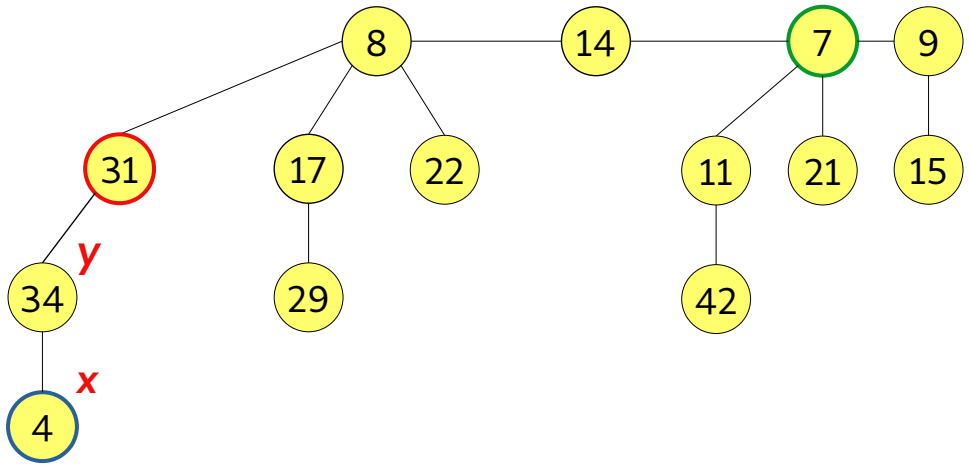
- Устанавливаем в узле 37 новый ключ 4
- Нарушены свойства кучи min-heap: 4 < 34



Уменьшение ключа (DecreaseKey)

Изменим ключ 37 до значения 4

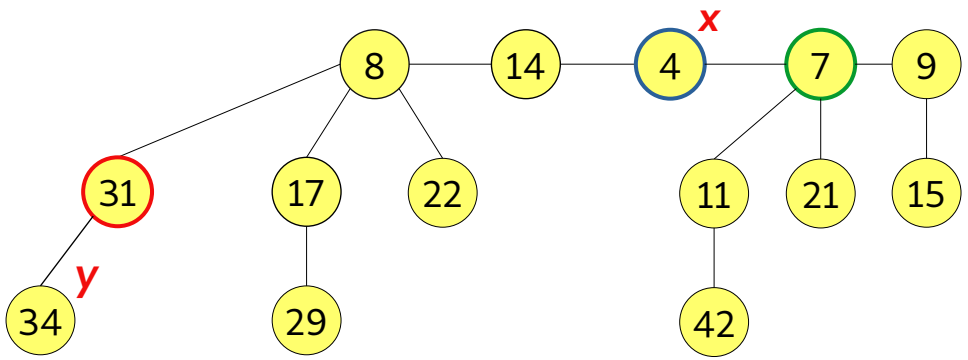
- Устанавливаем в узле 37 новый ключ 4
- Нарушены свойства кучи min-heap: 4 < 34
- Выполняем **FibHeapCut(<4>, <34>), FibHeapCascadingCut(<34>)**



Уменьшение ключа (DecreaseKey)

Выполняем **FibHeapCut(<4>, <34>)**

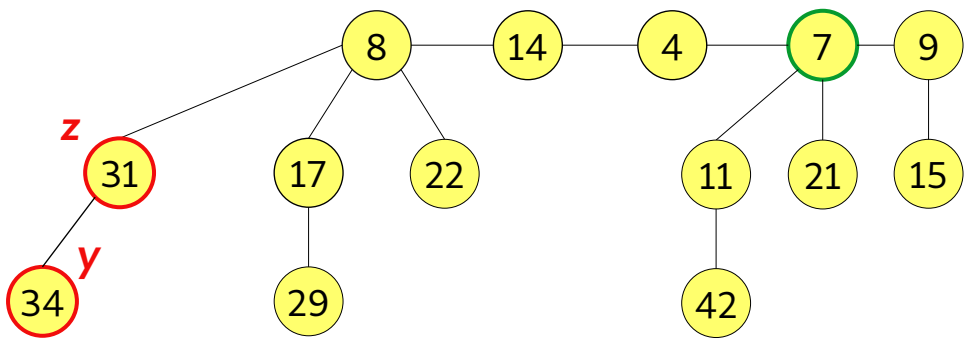
- Переносим <4> в список корней кучи
- Устанавливаем <4>.*mark* = FALSE



Уменьшение ключа (DecreaseKey)

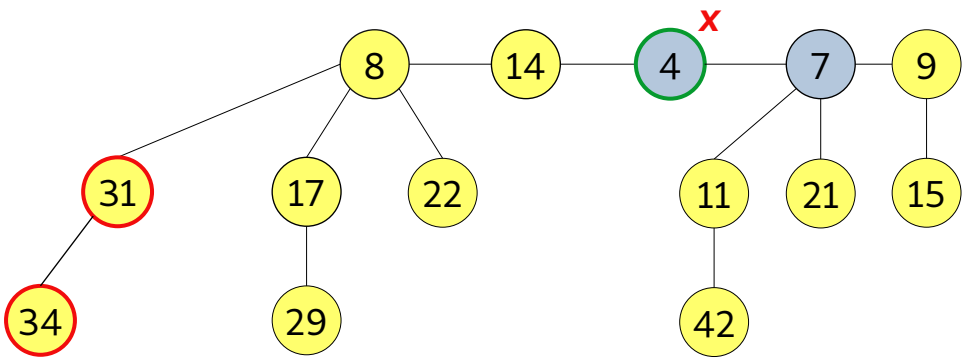
Выполняем **FibHeapCascadingCut(<34>)**

Изменяем значение <34>.*mark* с FALSE на **TRUE**



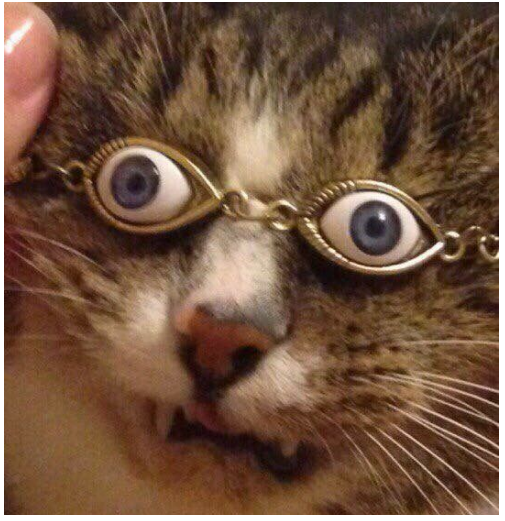
Уменьшение ключа (DecreaseKey)

- Корректируем указатель на минимальный элемент
- **4 < 7, указатель изменился**



Удаление заданного узла (Delete)

```
function FibHeapDelete(heap, x)
    FibHeapDecreaseKey(heap, x, -Infinity)
    FibHeapDeleteMin(heap)
end function
```



Очередь с приоритетом (priority queue)

- В таблице приведены трудоёмкости операций различных очередей с приоритетом в худшем случае (*worst case*)
- Символом "*" отмечена амортизированная сложность операций

Операция	Binary heap	Binomial heap	Fibonacci heap	Pairing heap	Brodal heap
FindMin	$\Theta(1)$	$O(\log n)$	$\Theta(1)^*$	$\Theta(1)^*$	$\Theta(1)$
DeleteMin	$\Theta(\log n)$	$\Theta(\log n)$	$O(\log n)^*$	$O(\log n)^*$	$O(\log n)$
Insert	$\Theta(\log n)$	$O(\log n)$	$\Theta(1)^*$	$\Theta(1)^*$	$\Theta(1)$
DecreaseKey	$\Theta(\log n)$	$\Theta(\log n)$	$\Theta(1)^*$	$O(\log n)^*$	$\Theta(1)$
Merge / Union	$\Theta(n)$	$\Omega(\log n)$	$\Theta(1)$	$\Theta(1)^*$	$\Theta(1)$

Дальнейшее чтение

- Прочитать в **CLRS 3ed.**
  § 19.2 «Операции над объединяемыми пирамидами»
- Разобрать доказательство амортизированной вычислительной сложности операций (метод потенциалов)
- Прочитать в **CLRS 3ed.**
  § 19.4 «Оценка максимальной степени»
- Изучить распространённость и области применения фибоначчиевых куч и очередей Бродала

ご清聴ありがとうございました!

Даниил Михайлович Берлизов
Старший преподаватель Кафедры вычислительных систем СибГУТИ
E-mail: sillyhat34@gmail.com

Курс «Структуры и алгоритмы обработки данных»
Осенний семестр, 2021 г.