

# Лекция 6

## Фибоначчиевы кучи (Fibonacci heap)

**Курносов Михаил Георгиевич**

E-mail: [mkurnosov@gmail.com](mailto:mkurnosov@gmail.com)

WWW: [www.mkurnosov.net](http://www.mkurnosov.net)

Курс «Структуры и алгоритмы обработки данных»

Сибирский государственный университет телекоммуникаций и информатики (Новосибирск)

Осенний семестр, 2015

# Очередь с приоритетом (priority queue)

---

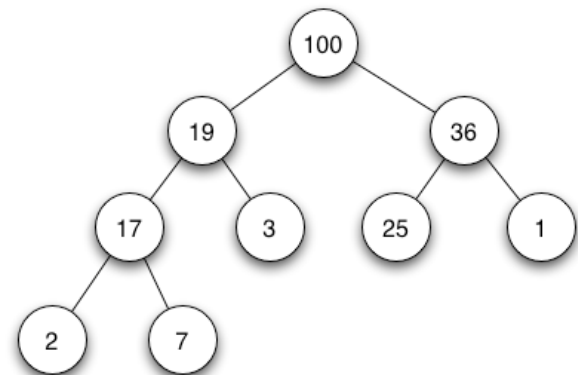
- **Очередь с приоритетом (priority queue)** – это очередь, в которой элементы имеют приоритет (вес)
- **Поддерживаемые операции:**
  - **Insert(*key*, *value*)** – добавляет в очередь значение *value* с приоритетом (весом, ключом) *key*
  - **DeleteMin/DeleteMax** – удаляет из очереди элемент с мин./макс. приоритетом (ключом)
  - **Min/Max** – возвращает элемент с мин./макс. ключом
  - **DecreaseKey** – изменяет значение ключа заданного элемента
  - **Merge( $q_1$ ,  $q_2$ )** – сливает две очереди в одну

Значение (Value)	Приоритет (Priority)
Слон	3
Кит	1
Лев	15

# Бинарная куча (binary heap)

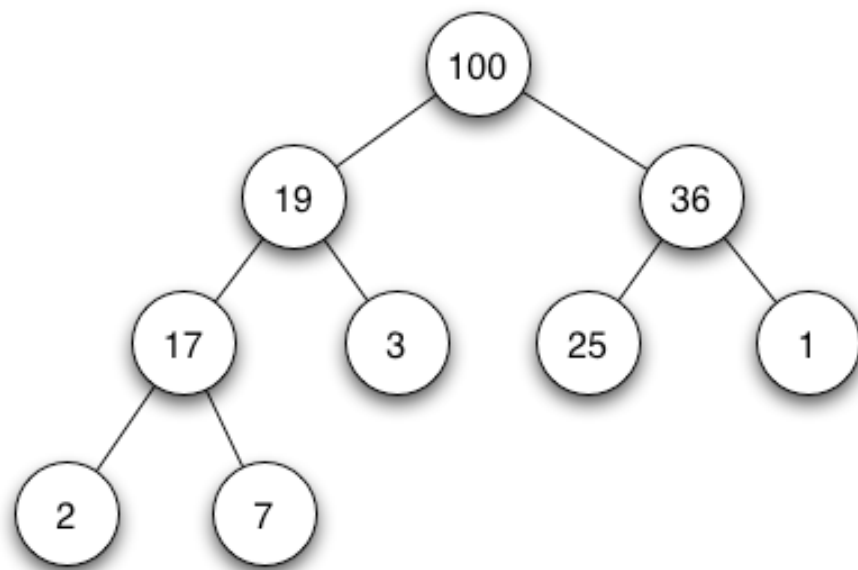
---

- **Бинарная куча (пирамида, сортирующее дерево, binary heap)** – это бинарное дерево, удовлетворяющее следующим условиям:
  - а) приоритет (ключ) любой вершины не меньше ( $\geq$ ), приоритета её потомков
  - б) дерево является завершённым бинарным деревом (complete binary tree) – все уровни заполнены слева направо, возможно за исключением последнего



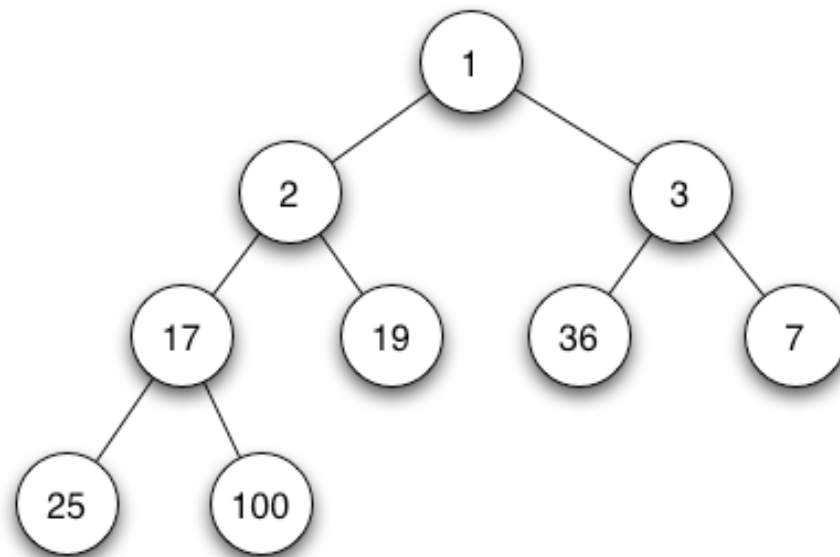
# Бинарная куча (binary heap)

---



**max-heap**

Приоритет любой вершины  
**не меньше ( $\geq$ ),**  
приоритета потомков



**min-heap**

Приоритет любой вершины  
**не больше ( $\leq$ ),**  
приоритета потомков

# Очередь с приоритетом (priority queue)

---

- В таблице приведены трудоемкости операций очереди 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)

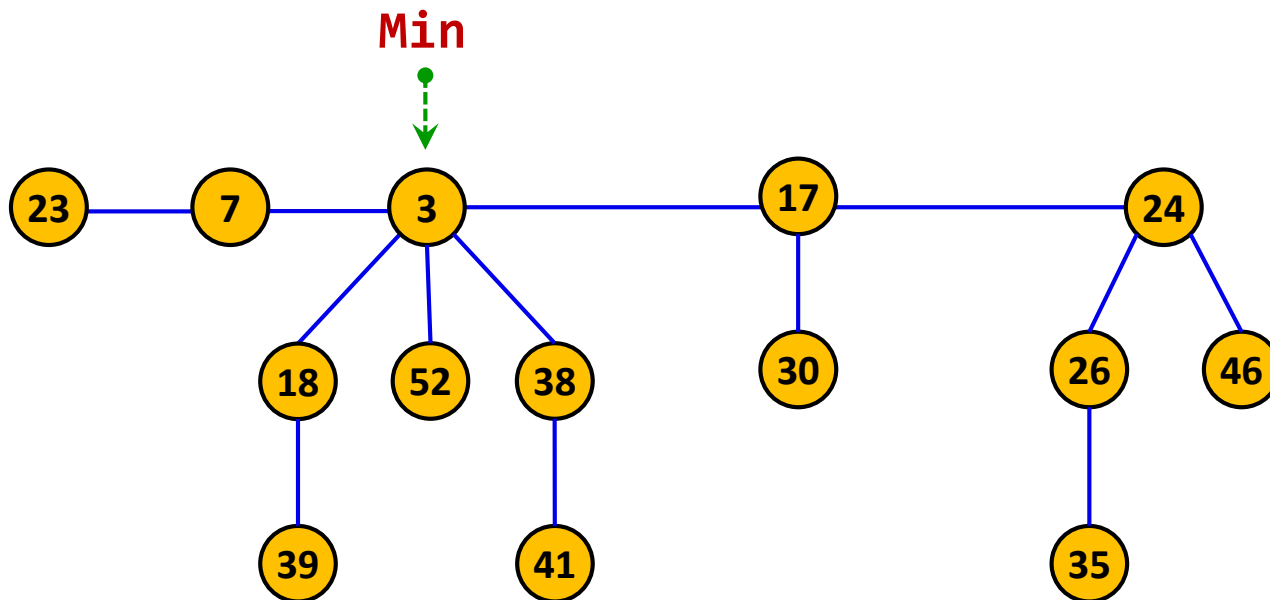
---

- **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 минимального остовного дерева, поиск кратчайшего пути в графе)
- Например, в алгоритме Дейкстры фибоначчиевы кучи (min-heap) можно использовать для хранения вершин и быстрого уменьшения текущего кратчайшего пути до них (DecreaseKey)
- **Авторы:** Michael L. Fredman, Robert E. Tarjan, 1984
- Fredman M.L., Tarjan R.E. **Fibonacci heaps and their uses in improved network optimization algorithms.** – Journal of the Association for Computing Machinery. – 1987, 34 (3). – pp. 596-615  
([http://www.cl.cam.ac.uk/~sos22/supervise/dsaa/fib\\_heaps.pdf](http://www.cl.cam.ac.uk/~sos22/supervise/dsaa/fib_heaps.pdf))

# 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
- Максимальная степень  $D(n)$  узла в фибоначчиевой куче из  $n$  элементов:

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

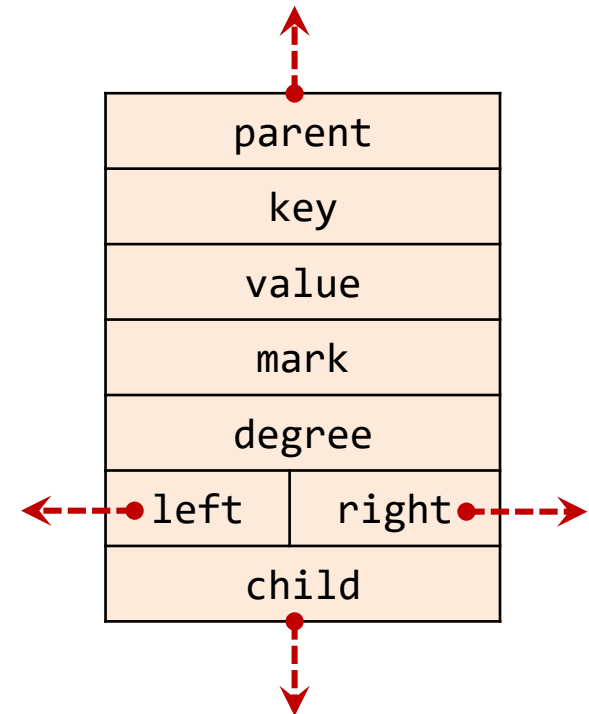


min-heap (5 деревьев, 14 узлов)

# Узел фибоначчиевой кучи

---

- Дочерние узлы (включая корни) объединены в циклический дважды связный список (doubly linked list)
- Каждый узел фибоначчиевой кучи (дерева) 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** – были ли потери узлом дочерних узлов



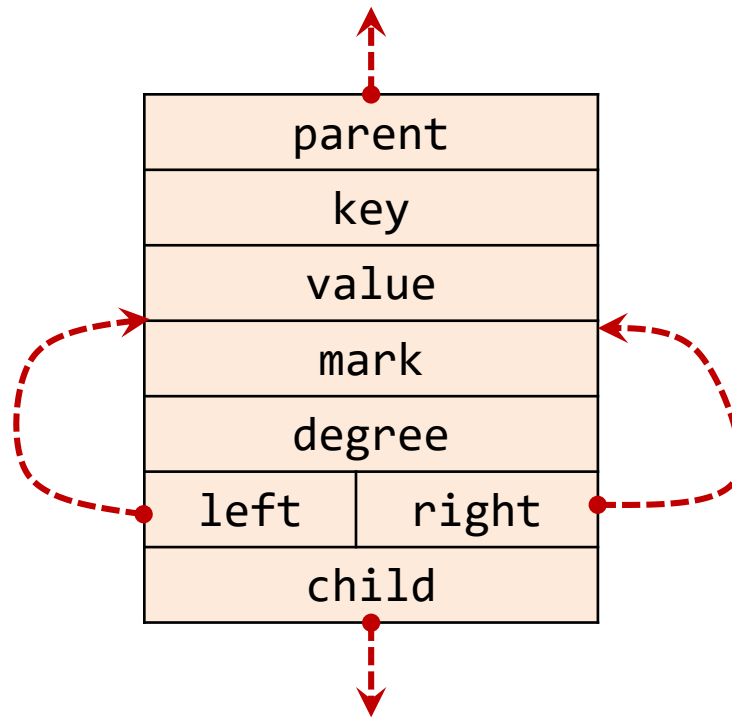


# Узел фибоначчиевой кучи

---

- Если узел является единственным дочерним узлом

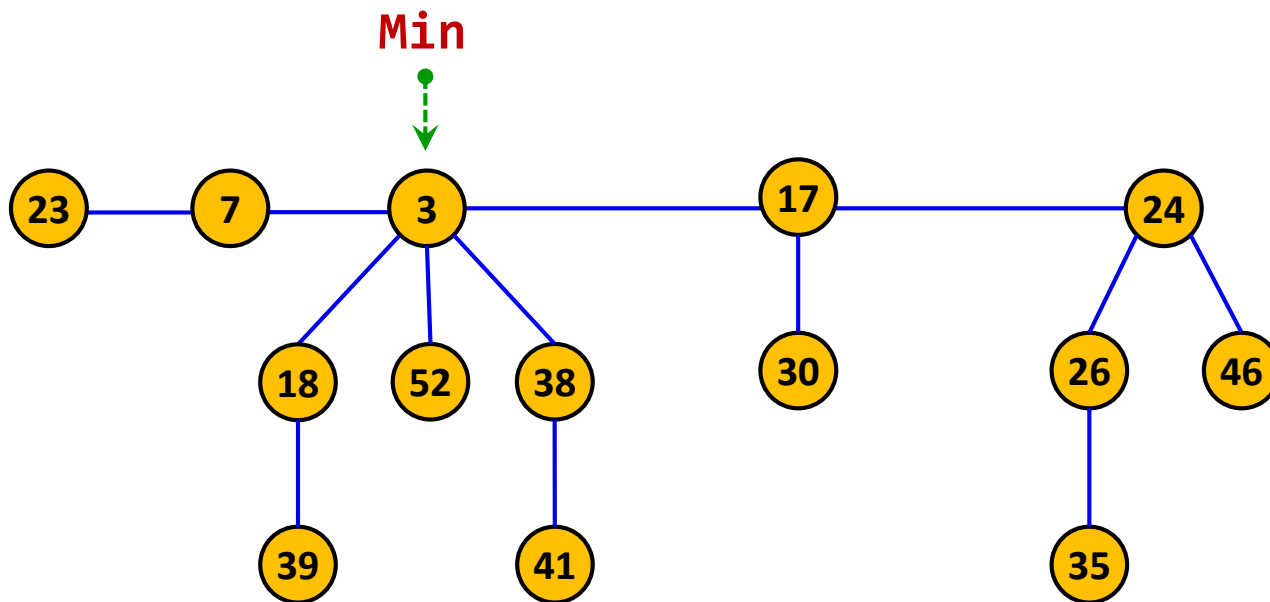
$$x.left = x.right = x$$



# Фибоначчиевы кучи (Fibonacci heaps)

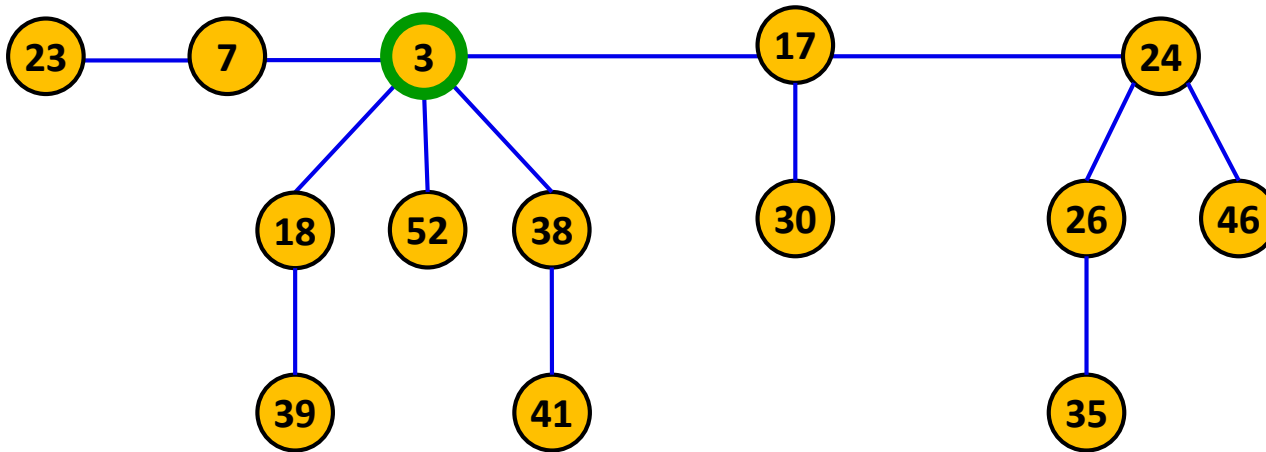
---

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



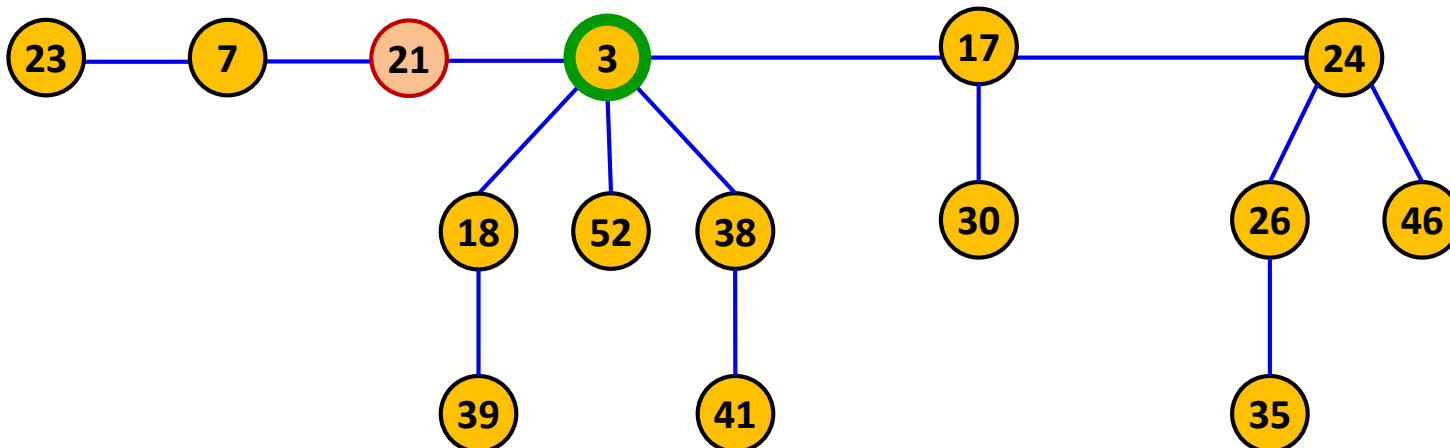
**min-heap** (5 деревьев, 14 узлов)

# Добавление узла в кучу (Insert)



## Добавление узла с ключом 21

Создаем в памяти новый узел (21) и помещаем его слева от узла с минимальным ключом (3)



# Добавление узла в кучу (Insert)

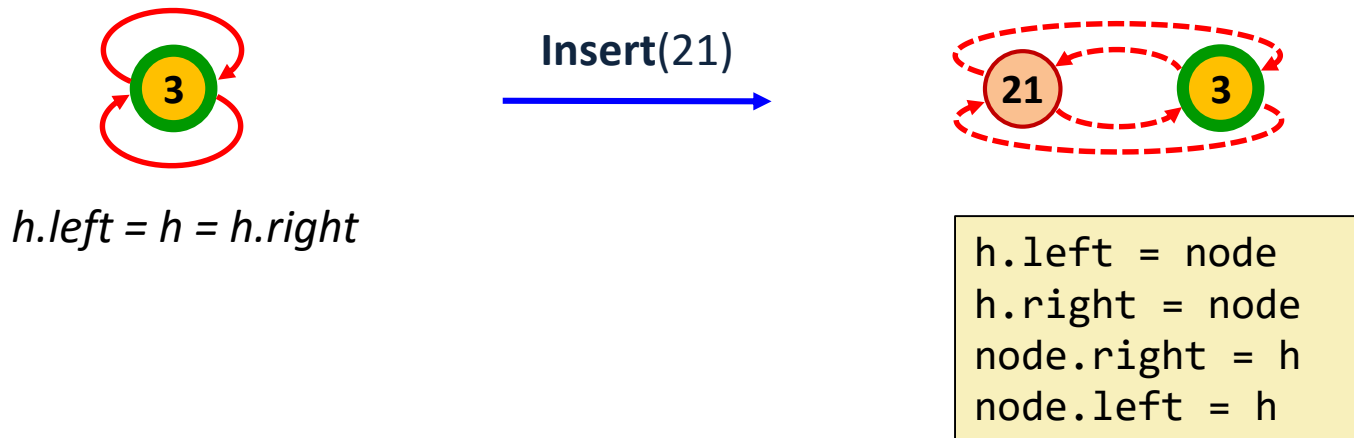
```
function FibHeapInsert(heap, key, value)
    node = AllocateMemory();
    node.key = key
    node.value = value
    node.degree = 0
    node.mark = false
    node.parent = NULL
    node.child = NULL
    node.left = node
    node.right = node
    /* Добавляем node в список корней heap */
    FibHeapAddNodeToRootList(node,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

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

# Добавление узла в список корней

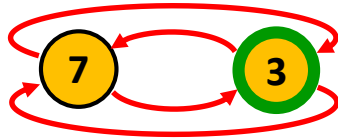
---

- 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

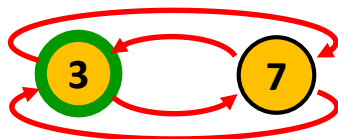


# Добавление узла в список корней

- 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

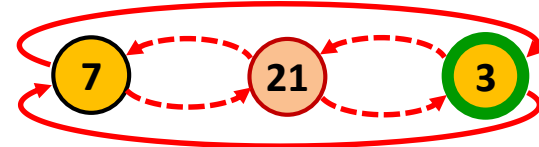


или



$h.left \neq h \neq h.right$

Insert(21)



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

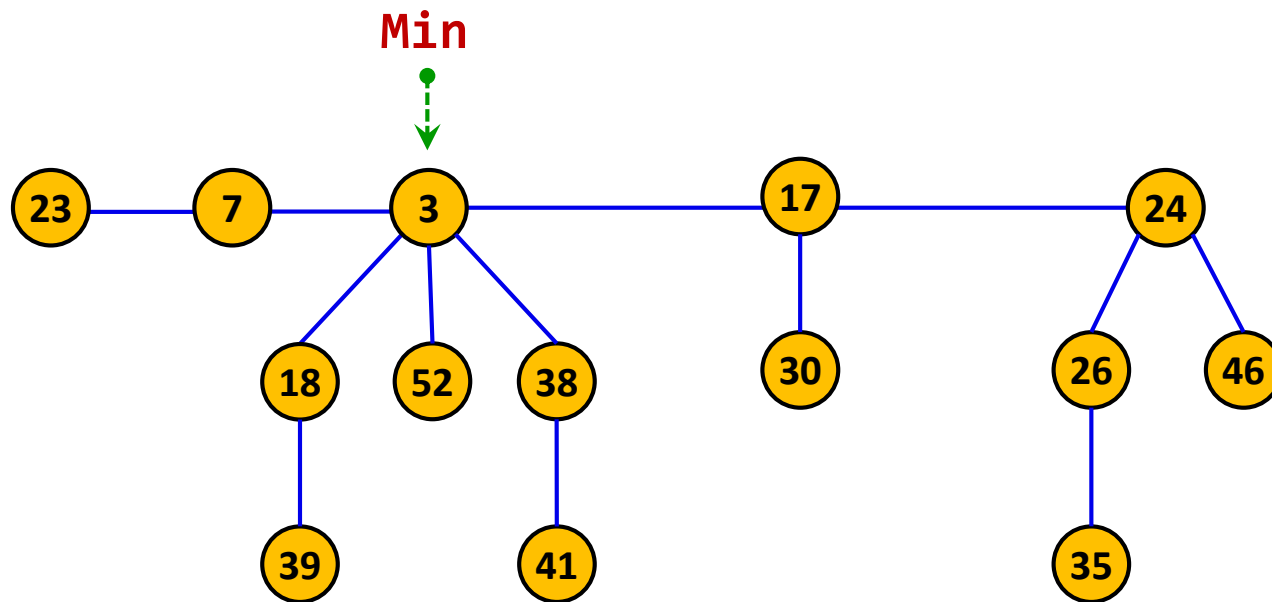
# Добавление узла в список корней

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

$$T_{\text{AddNodeToRootList}} = O(1)$$

# 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)

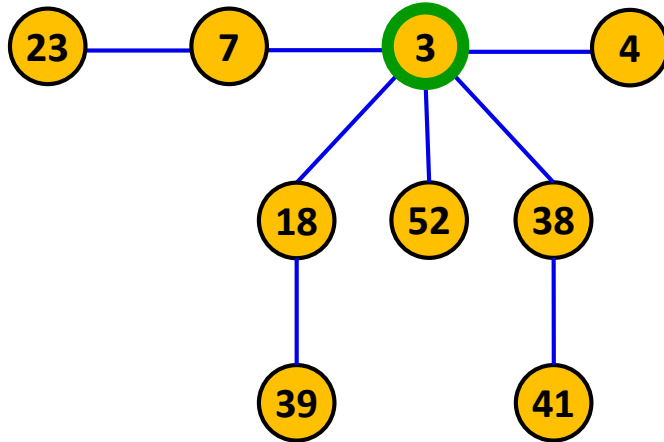
---

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

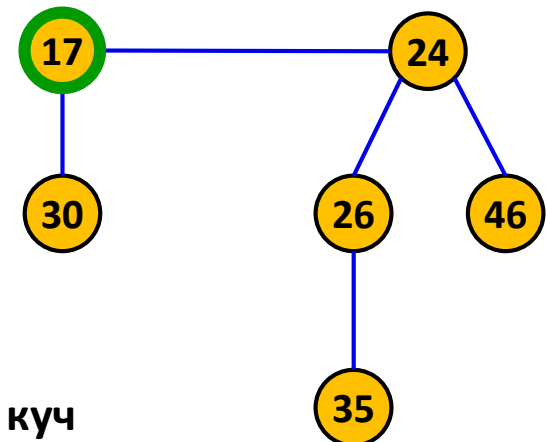
$$T_{Min} = O(1)$$

# Слияние фибоначчиевых куч (Union)

Heap 1

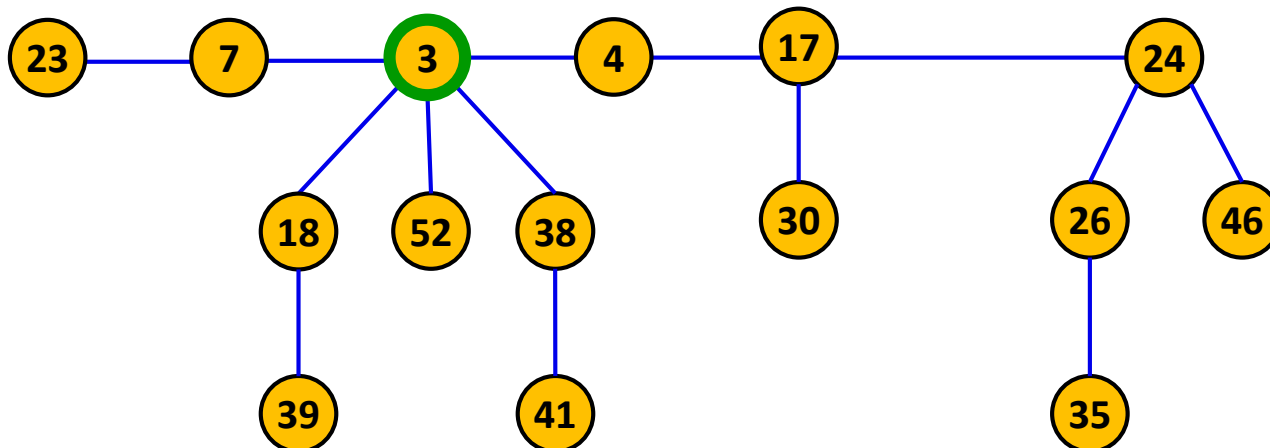


Heap 2



**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 < heap1.min.key)
  then
    heap.min = heap2.min
  end if
  heap.nnodes = heap1.nnodes + heap2.nnodes
  FreeMemory(heap1)
  FreeMemory(heap2)
  return heap
end function
```

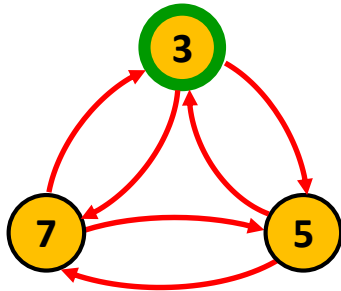
$$T_{Union} = O(1)$$

# Слияние пары списков корней

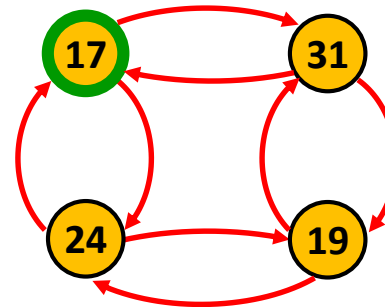
---

- 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



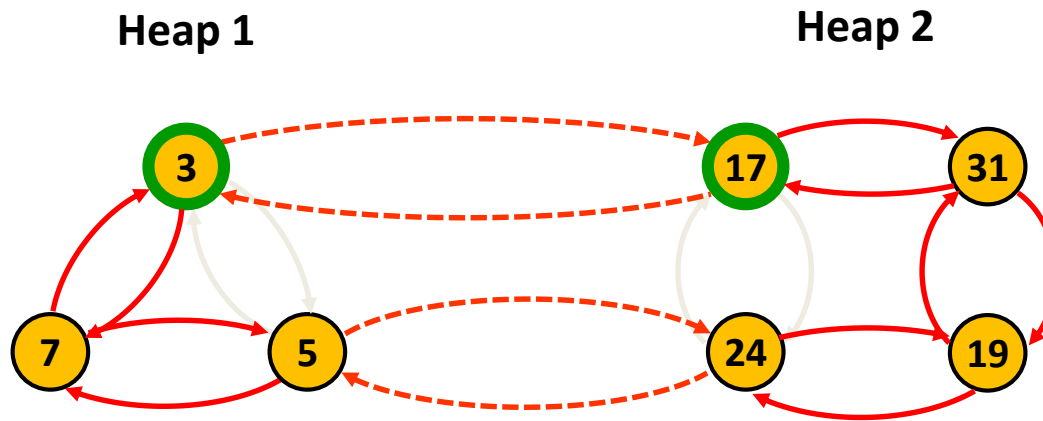
Heap 2



# 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

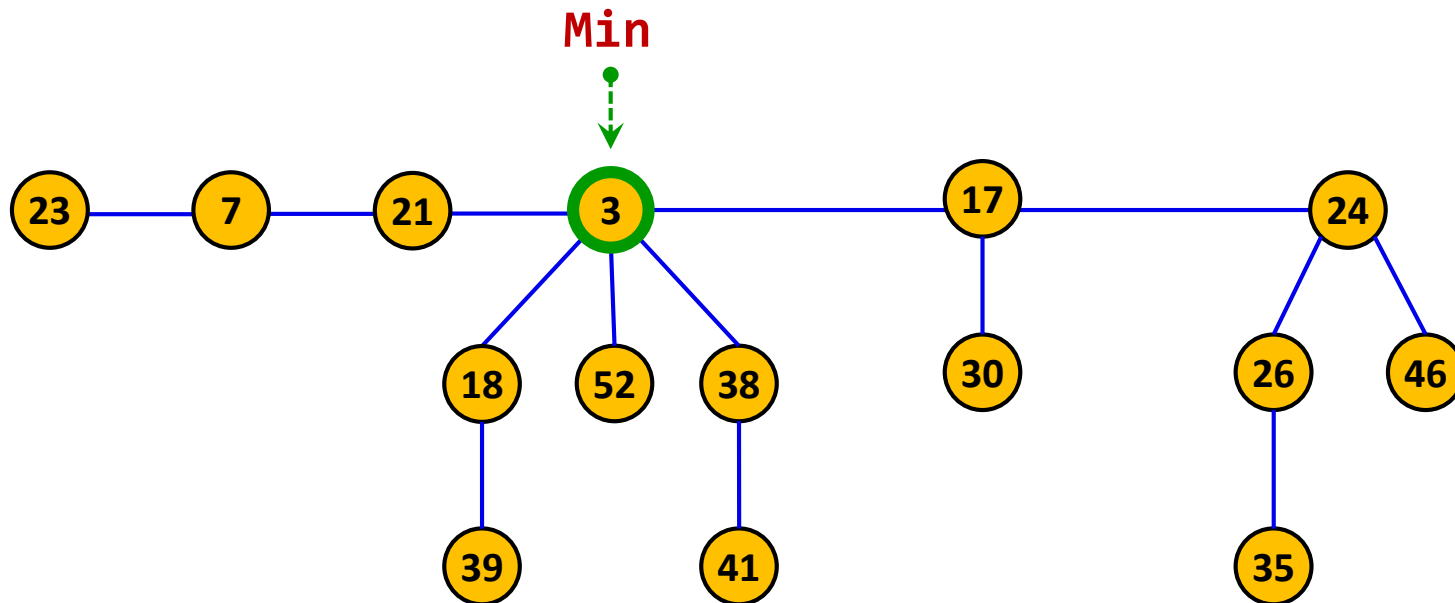
  left1 = heap1.left
  left2 = heap2.left
  left1.right = heap2
  heap2.left = left1
  heap1.left = left2
  left2.right = heap1
  return heap1
end function
```

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

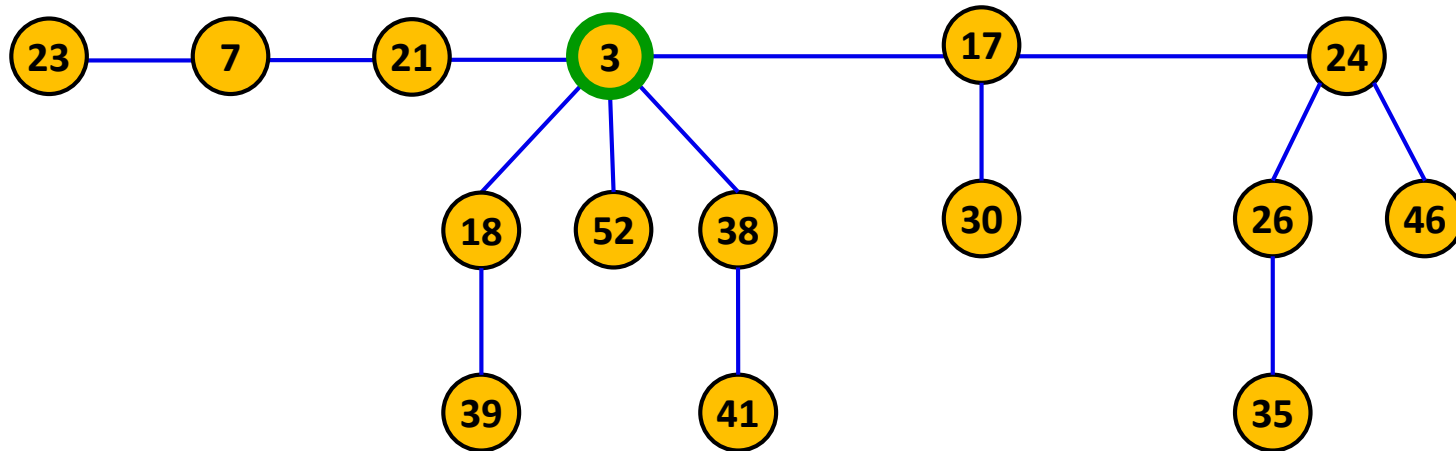
# Удаление минимального узла (DeleteMin)

---

- 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

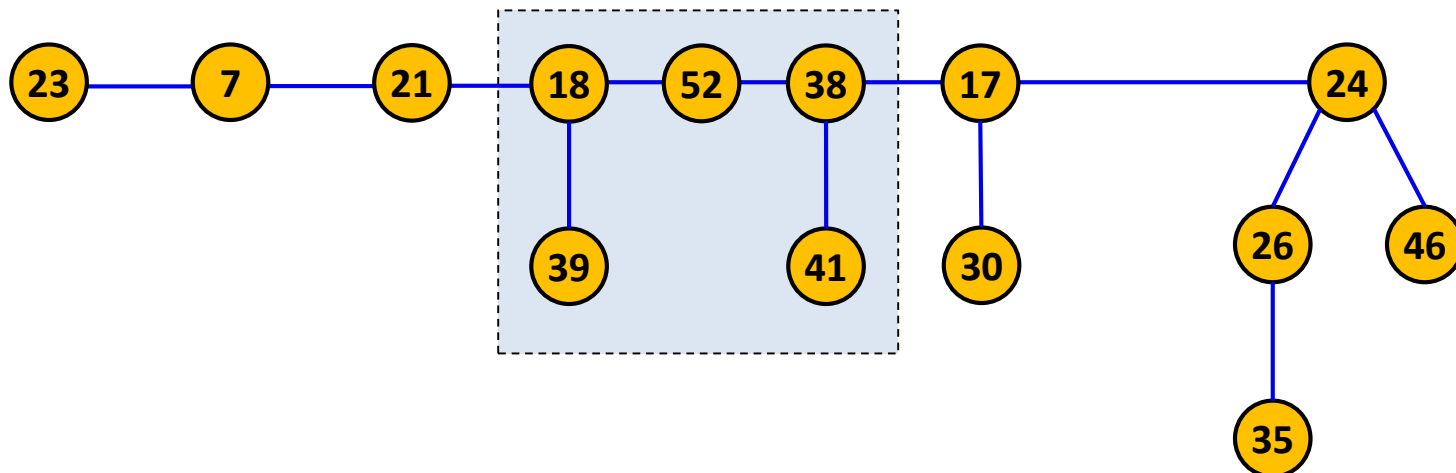


# Удаление минимального узла (DeleteMin)



**Удаление минимального элемента**

Поднимаем в список корней дочерние элементы минимального узла (3)





# Слияние пары списков корней

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

$O(\log n)$

# 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  $x.key \leq y.key$
  2. Делаем дочерним узлом  $x$  (и наоборот в случае  $\text{max-heap}$ ); поле  $x.degree$  увеличивается, а пометка  $y.mark$  снимается (если была установлена)
- Процедура Consolidate использует вспомогательный массив указателей  $A[0, 1, \dots, D(n)]$ ,  $D(n) \leq \lceil \log n \rceil$
- 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
  /* Цикл по всем узлам списка корней */
  for each w in heap.min do
    x = w
    d = x.degree
    while A[d] != NULL do
      y = A[d]      /* Степень y совпадает со степенью x */
      if x.key > y.key then
        FibHeapSwap(x, y)          /* Обмениваем x и y */
        FibHeapLink(heap, y, x)
      A[d] = NULL
      d = d + 1
    end while
    A[d] = x
  end for
```

}  $O(\log n)$

# Уплотнение списка корней (Consolidate)

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

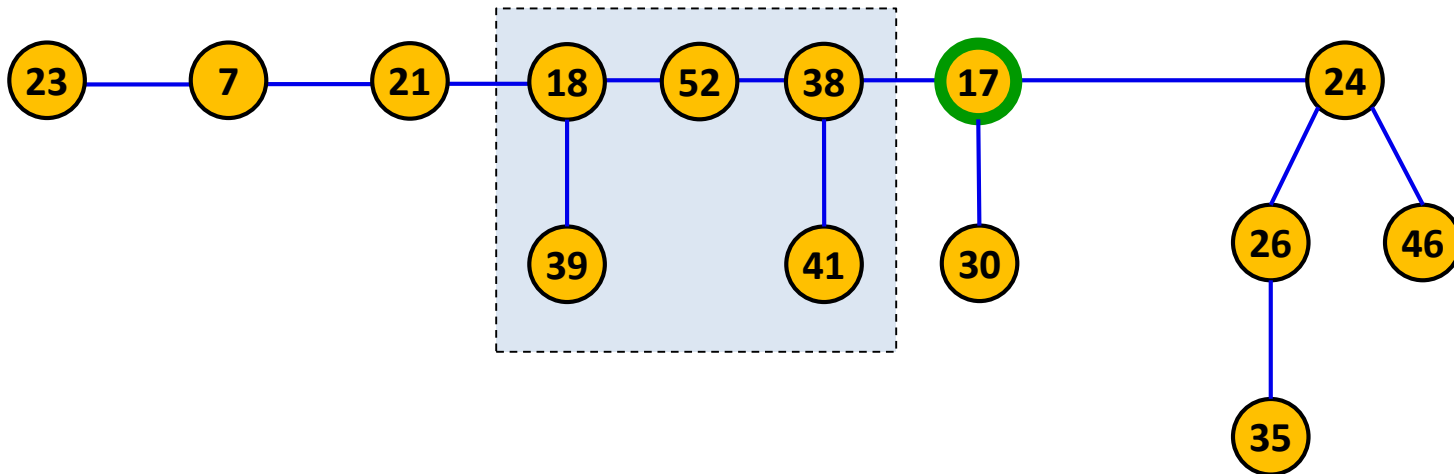
---

```
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 (3), минимальный узел из списка удалили (см. DeleteMin)
- Текущим минимальным узлом стал узел (17) – правый сосед узла (3):  $heap.min = z.right$  (причем,  $z.right$  возможно не минимальный узел)



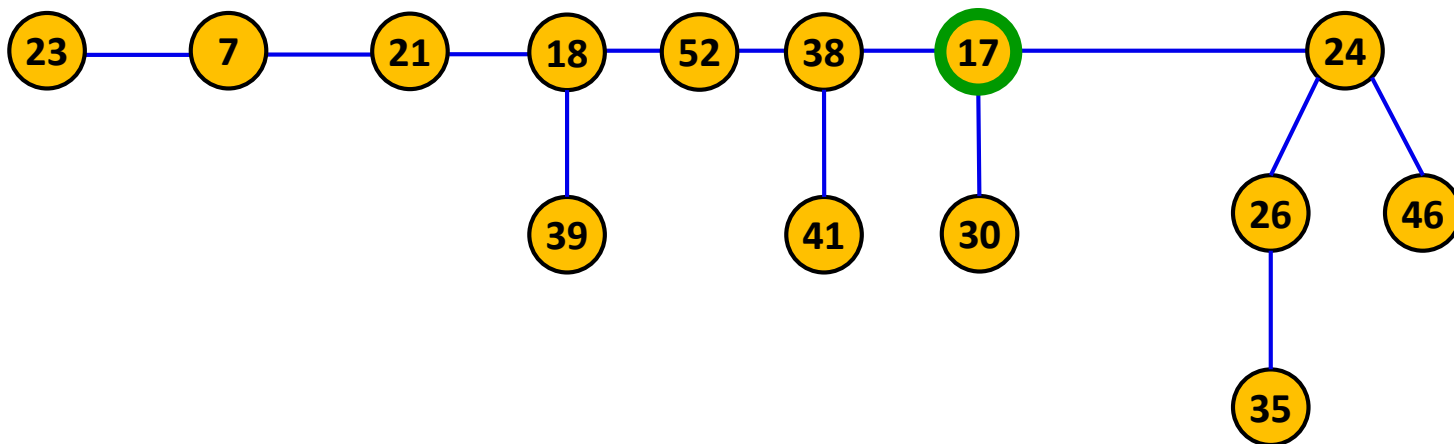
# Уплотнение списка корней (Consolidate)

---

- В куче  $n = 15$  узлов
- Максимальная степень корня дерева  $D(n) \leq \lfloor \log n \rfloor = 3$
- Массив  $A[0, 1, \dots, D(n)] = A[0, \dots, 3]$
- $A[i] = \text{NULL}$

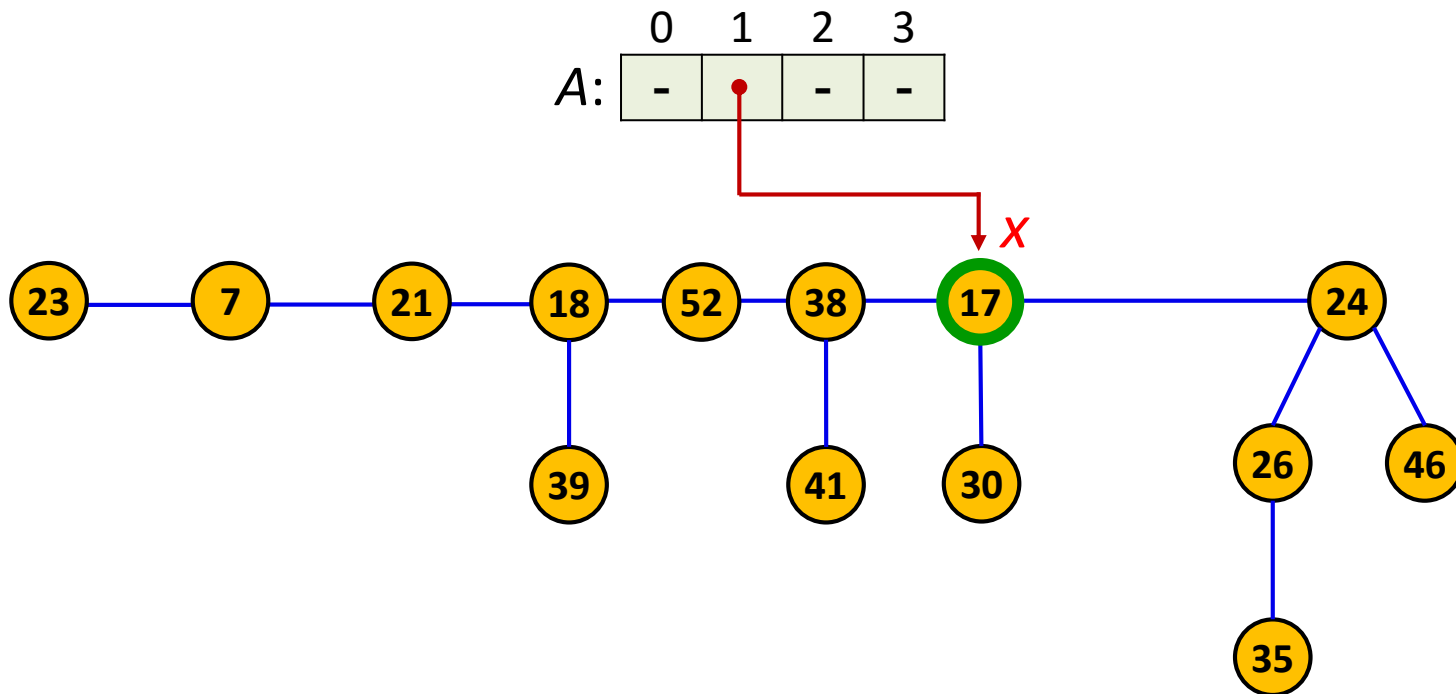
A:

0	1	2	3
-	-	-	-



# Уплотнение списка корней (Consolidate)

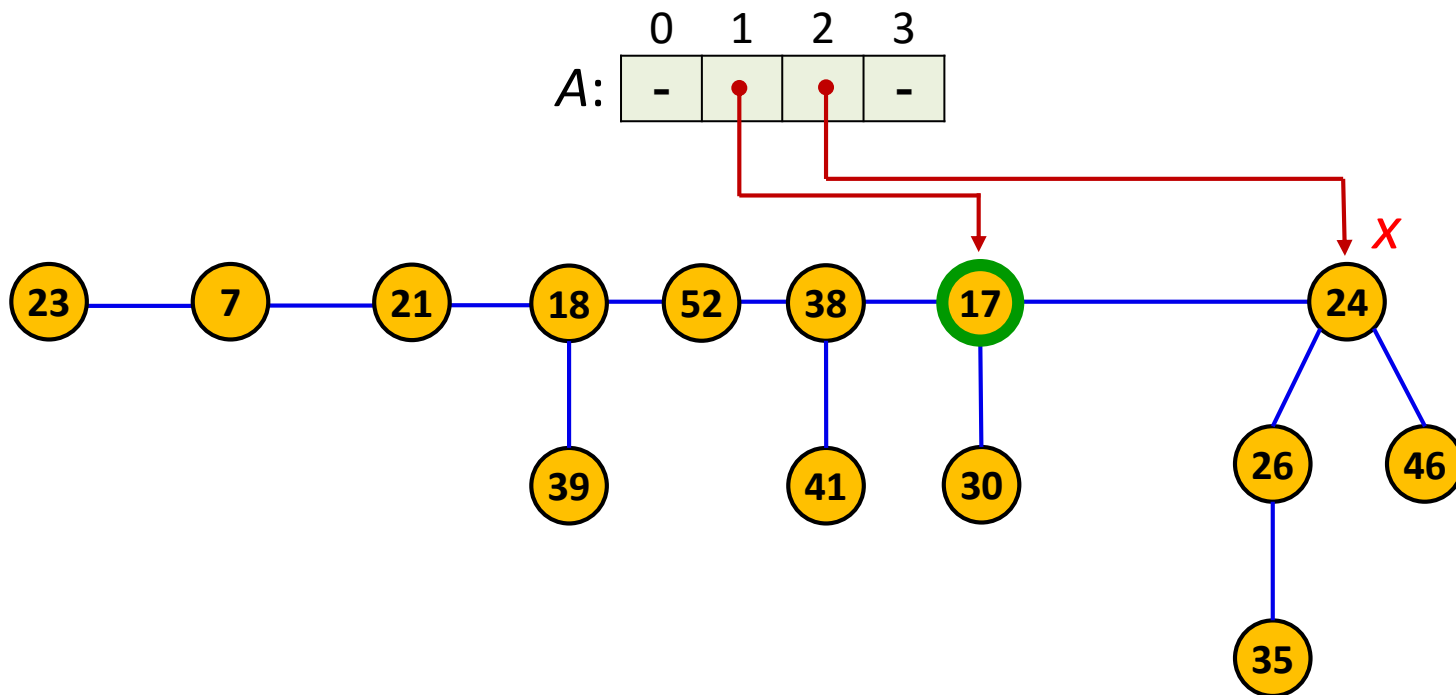
- Обход списка корней начинаем с узла `heap.min` (узел **17**)
- Степень узла 17 равна 1, ищем корни степени 1
- $A[1] = \text{NULL}$ : в списке нет других корней степени 1





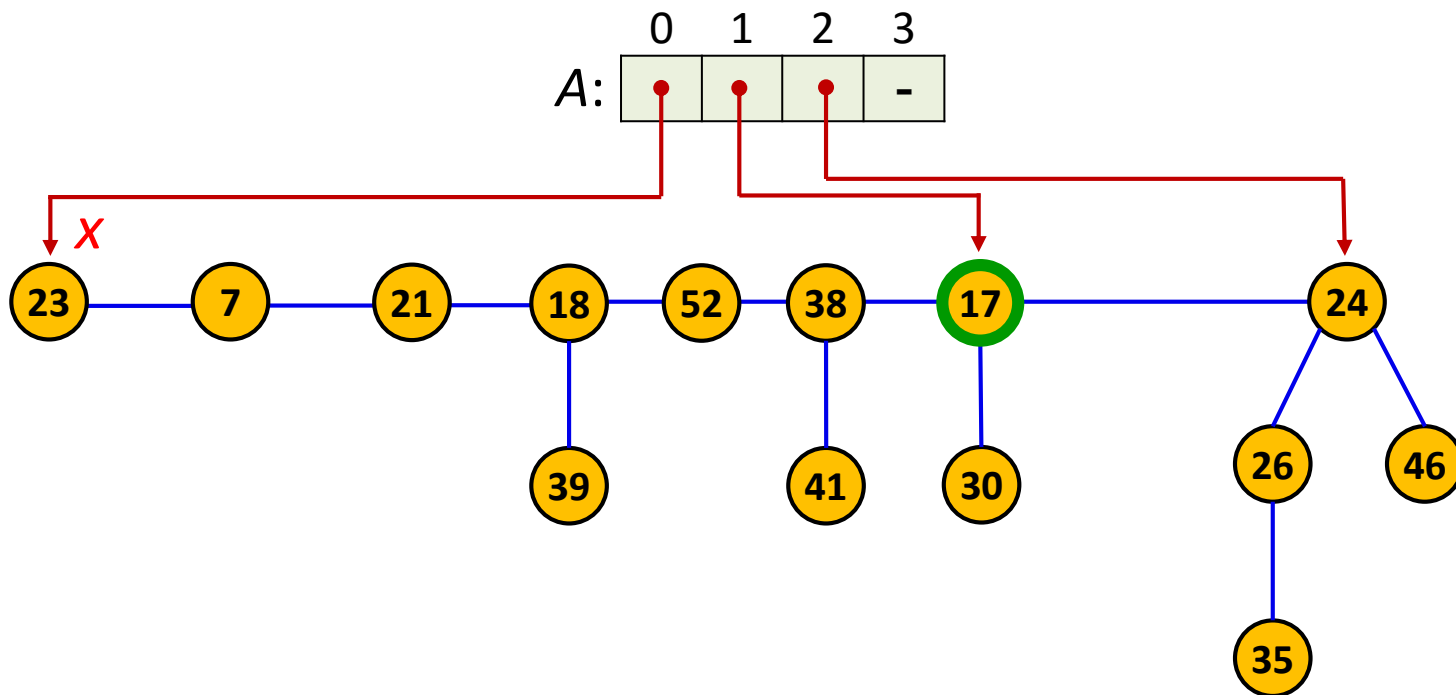
# Уплотнение списка корней (Consolidate)

- Переходим к следующему элементу – узлу **24**
- Степень узла 24 равна 2, ищем корни степени 2
- $A[2] = \text{NULL}$ : в списке нет других корней степени 2



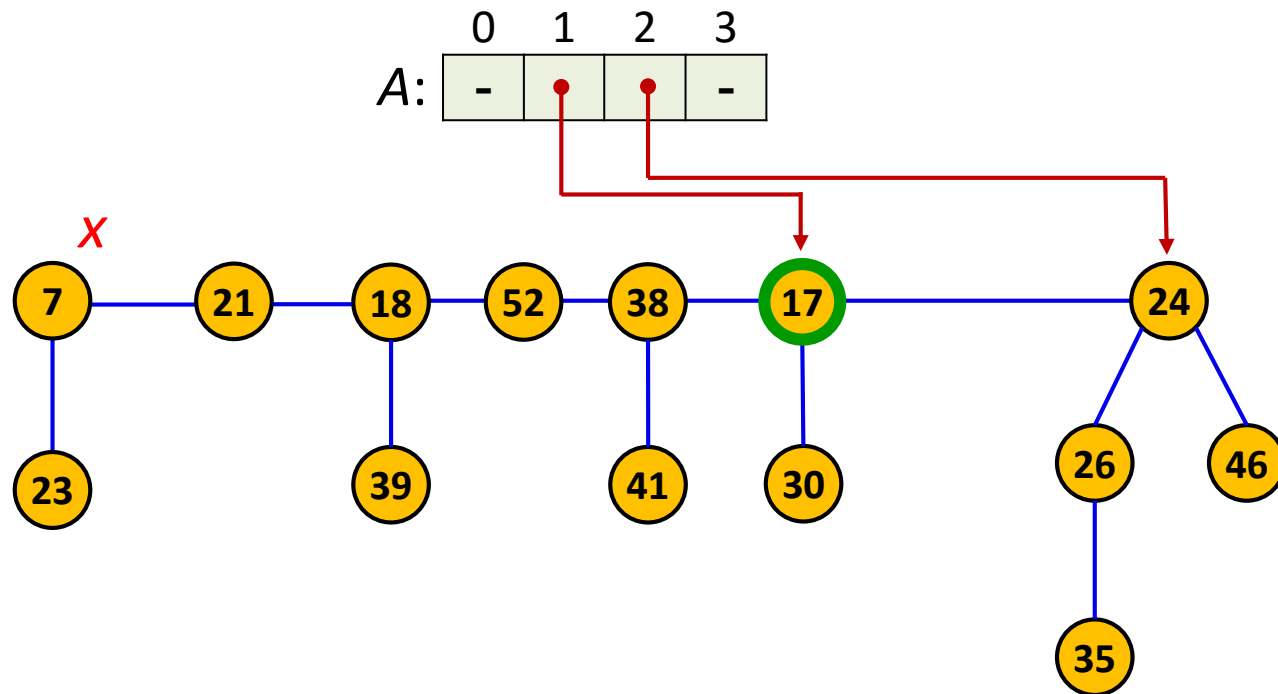
# Уплотнение списка корней (Consolidate)

- Переходим к следующему элементу – узлу **23**
- Степень узла 23 равна 0, ищем корни степени 0
- $A[0] = \text{NULL}$ : в списке нет других корней степени 0



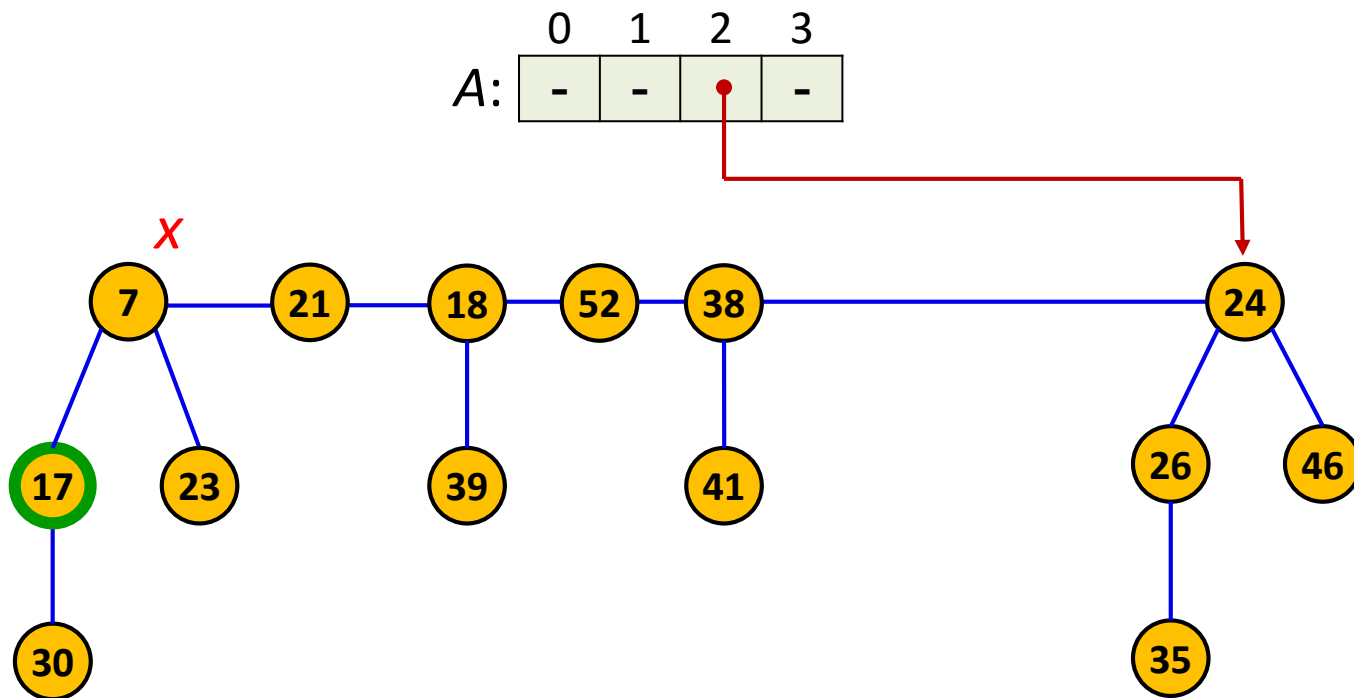
# Уплотнение списка корней (Consolidate)

- Переходим к следующему элементу – узлу **7**
- Степень узла 7 равна 0, ищем корни степени 0
- $A[0] = (23)$ : узел 23 имеет также степень 0
- $7 < 23$ : делаем 23 дочерним узлом элемента 7



# Уплотнение списка корней (Consolidate)

- Степень узла **7** увеличена до 1, ищем корни степени 1
- $A[1] = (17)$ : узел 17 имеет также степень 1
- $7 < 17$ : делаем 17 дочерним узлом элемента 7

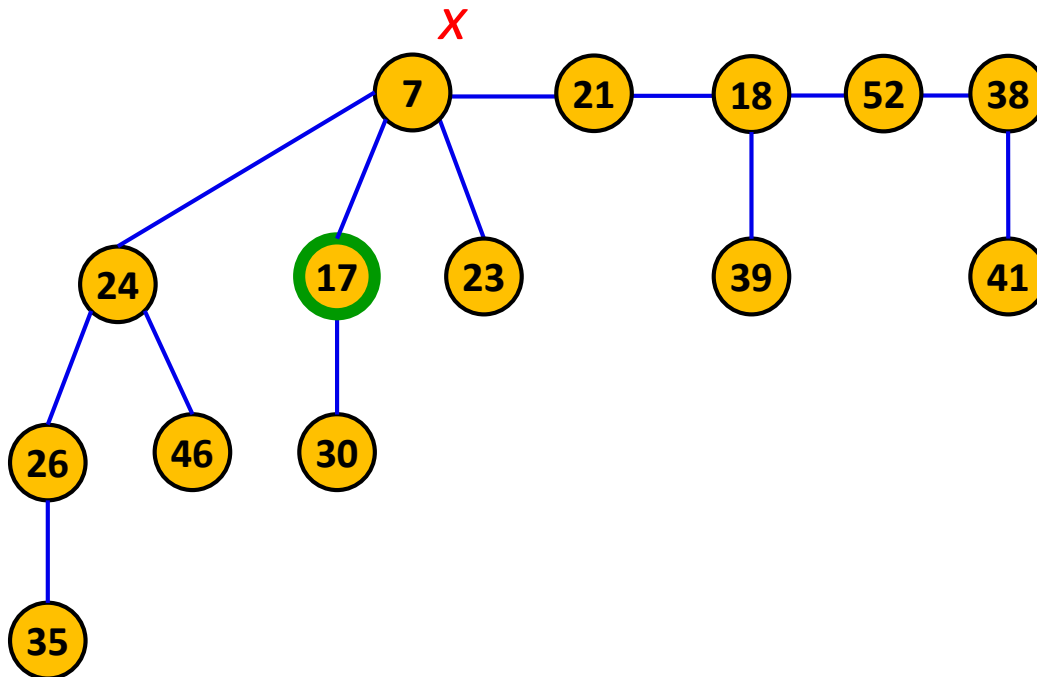


# Уплотнение списка корней (Consolidate)

- Степень узла **7** увеличена до 2, ищем корни степени 2
- $A[2] = (24)$ : узел 24 имеет также степень 2
- $7 < 24$ : делаем 24 дочерним узлом элемента 7

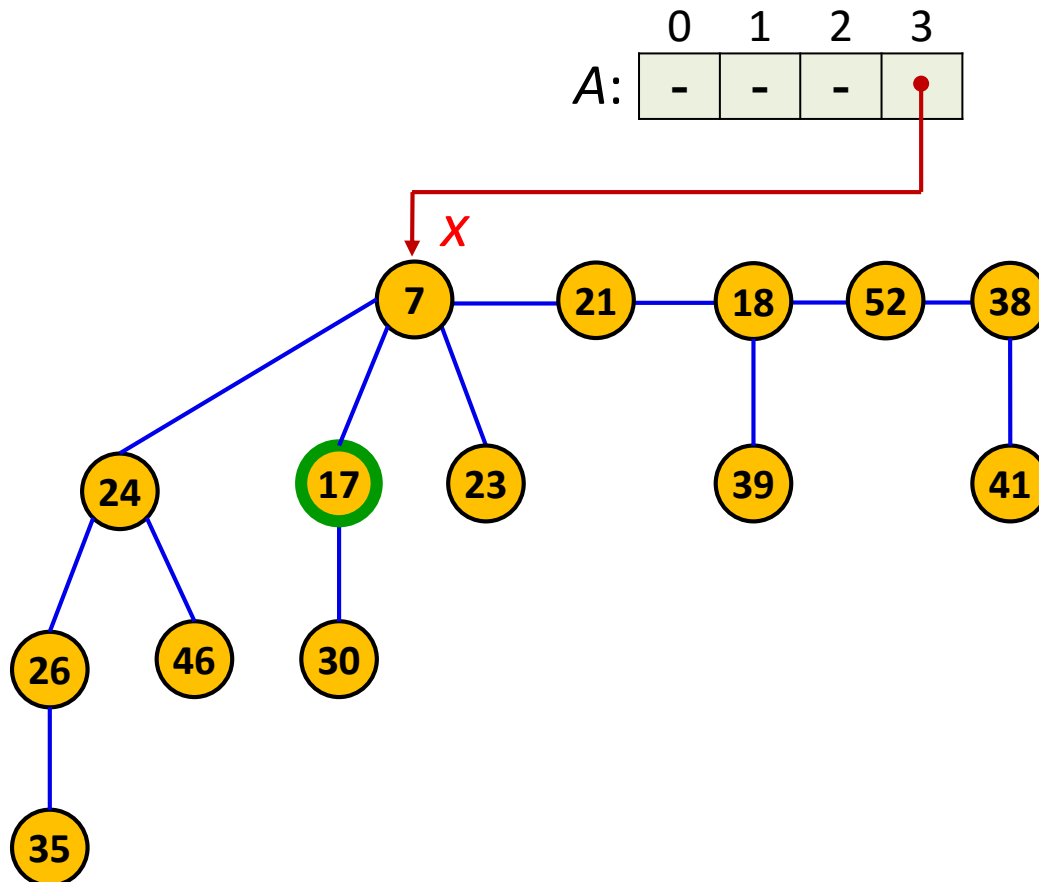
A:

0	1	2	3
-	-	-	-



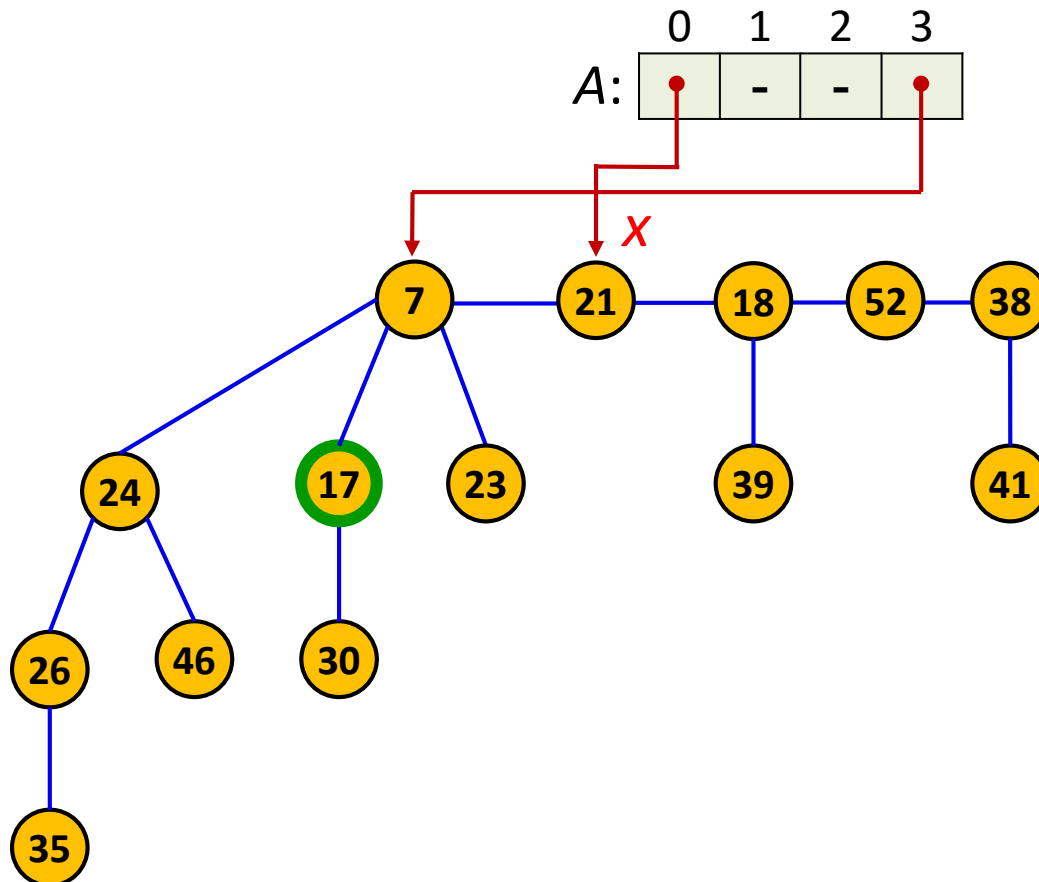
# Уплотнение списка корней (Consolidate)

- Степень узла **7** увеличена до 3, ищем корни степени 3
- $A[3] = \text{NULL}$
- Устанавливаем указатель  $A[3]$  на узел 7



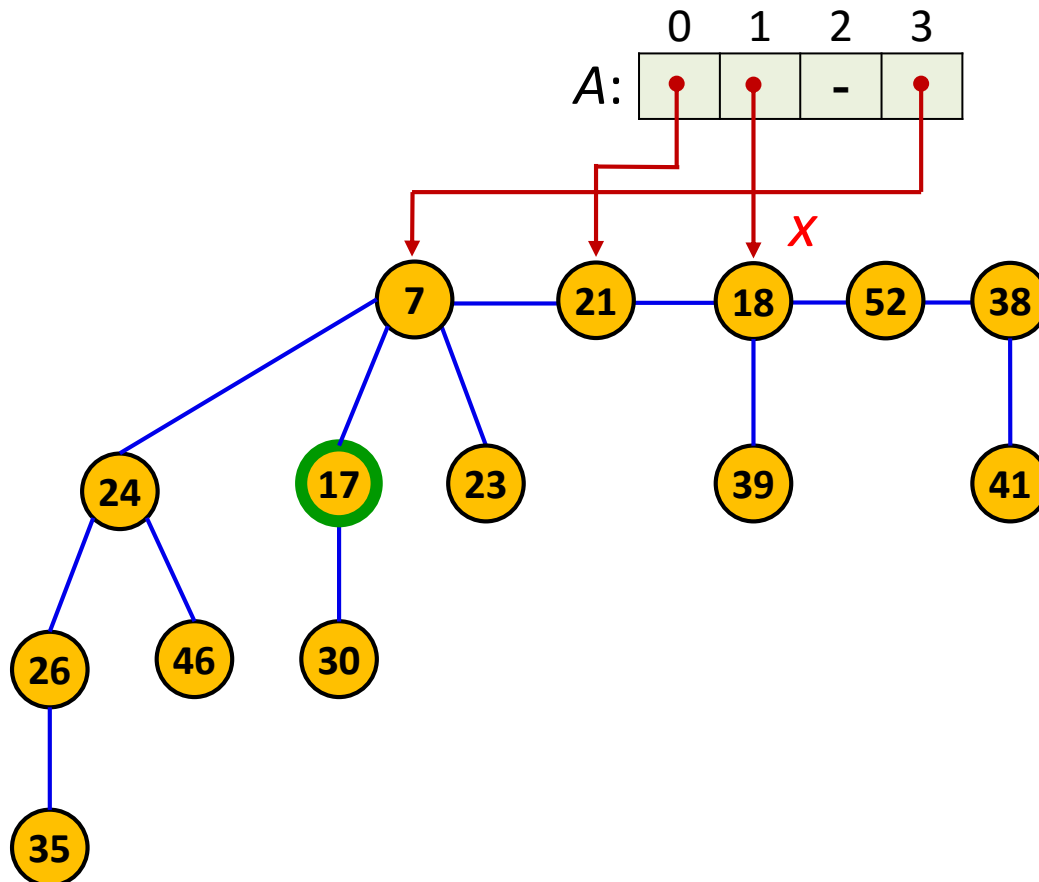
# Уплотнение списка корней (Consolidate)

- Переходим к следующему элементу – узлу 21
- Степень узла 21 равна 0, ищем корни степени 0
- $A[0] = \text{NULL}$ , устанавливаем  $A[0] = (21)$



# Уплотнение списка корней (Consolidate)

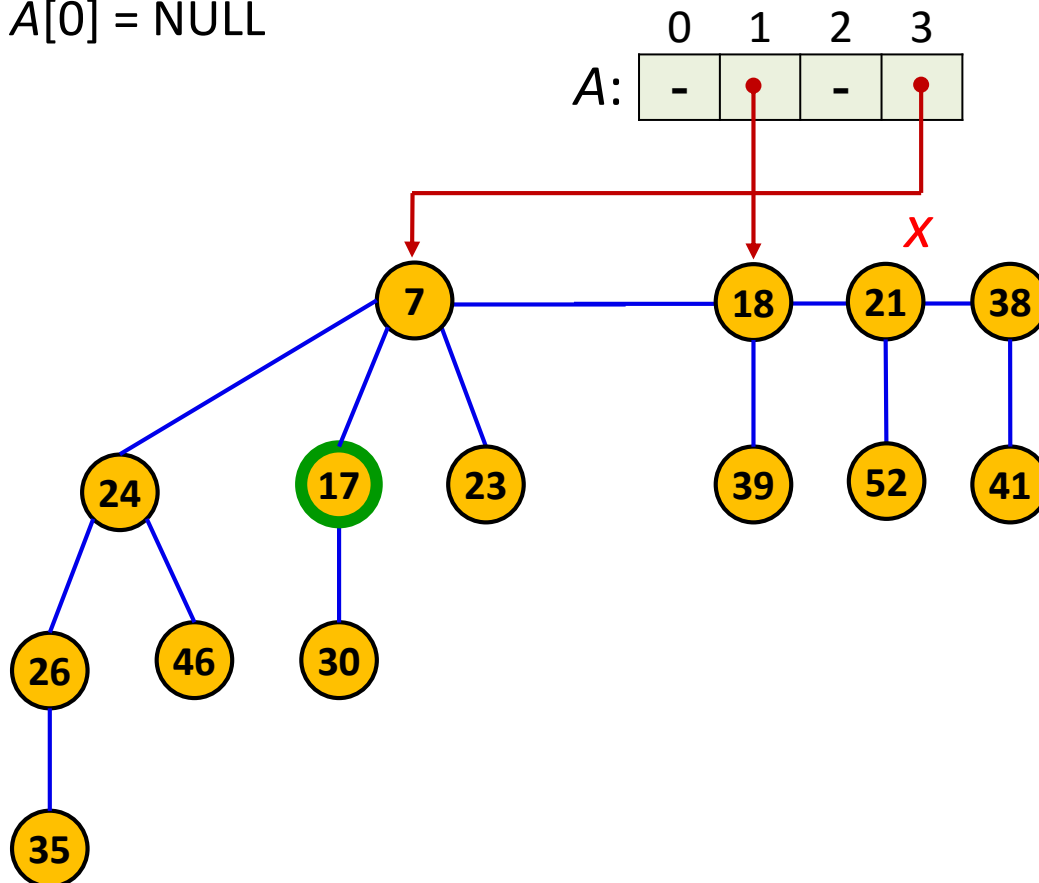
- Переходим к следующему элементу – узлу 18
- Степень узла 18 равна 1, ищем корни степени 1
- $A[1] = \text{NULL}$ , устанавливаем  $A[1] = (18)$





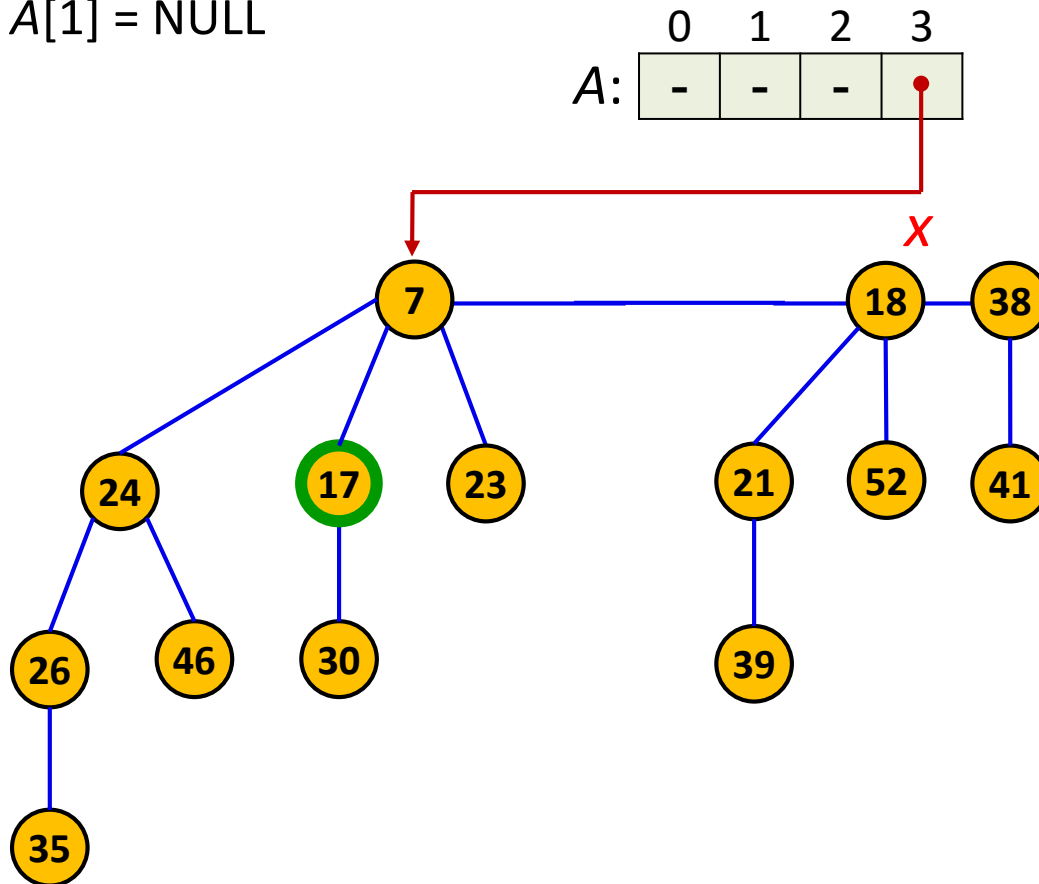
# Уплотнение списка корней (Consolidate)

- Переходим к следующему элементу – узлу 52
- Степень узла 52 равна 0, ищем корни степени 0:  $A[0] = (21)$
- $52 > 21$ : обмениваем узлы 21 и 52, делаем 52 дочерним узлом эл-та 21
- $A[0] = \text{NULL}$



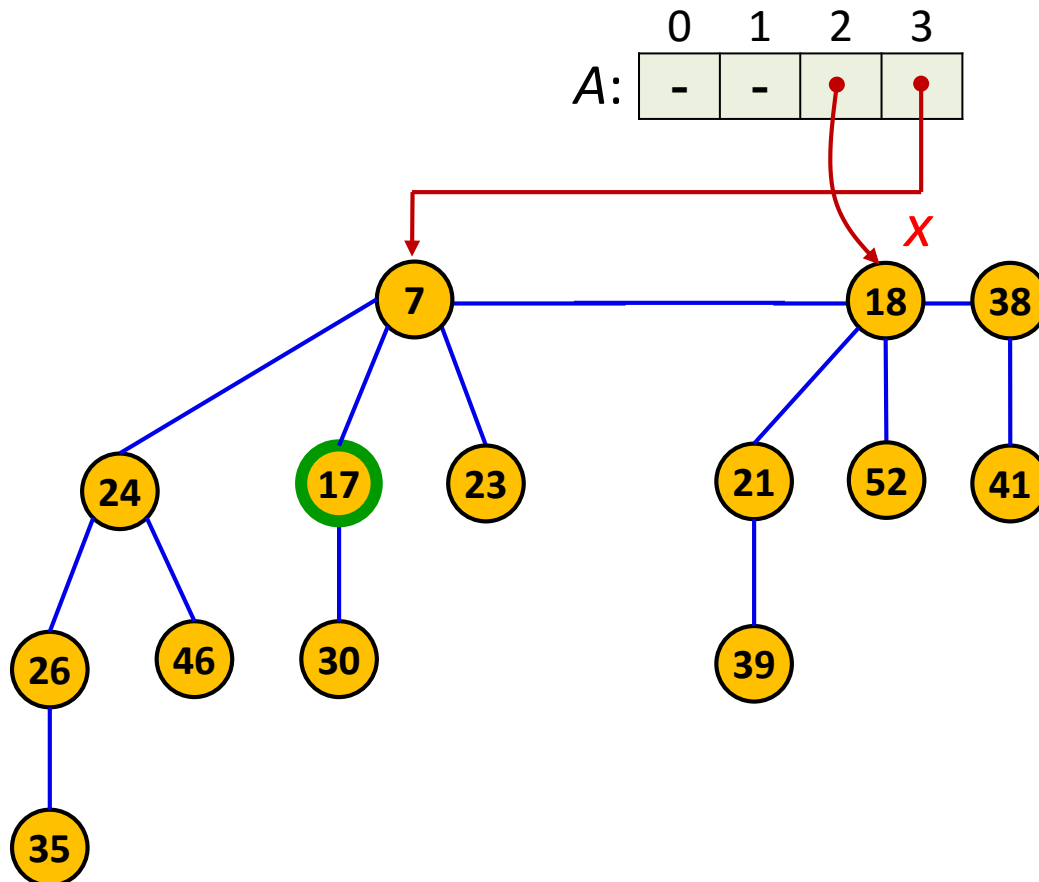
# Уплотнение списка корней (Consolidate)

- Степень узла **21** увеличена до 1, ищем корни степени 1
- $A[1] = (18)$
- $21 > 18$ : обмениваем узлы 21 и 18, делаем 21 дочерним узлом эл-та 18
- $A[1] = \text{NULL}$



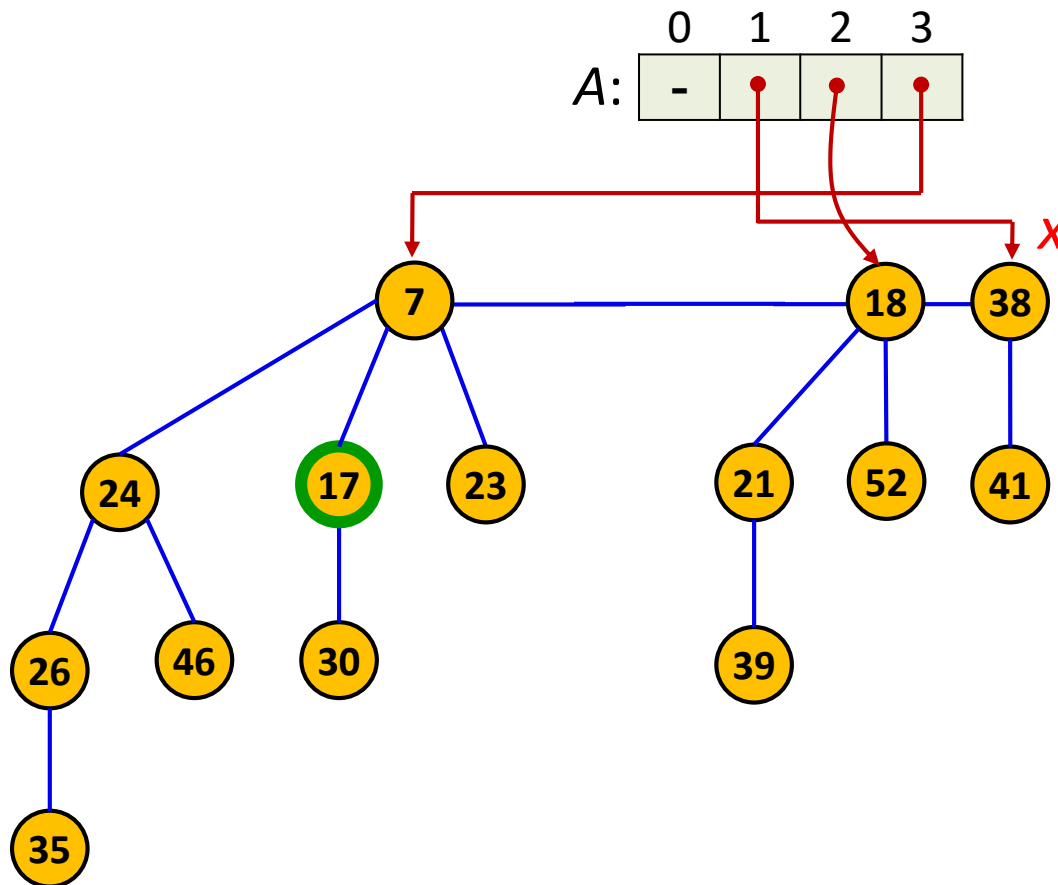
# Уплотнение списка корней (Consolidate)

- Степень узла **18** увеличена до 2, ищем корни степени 2
- $A[2] = \text{NULL}$
- Устанавливаем  $A[2] = (18)$



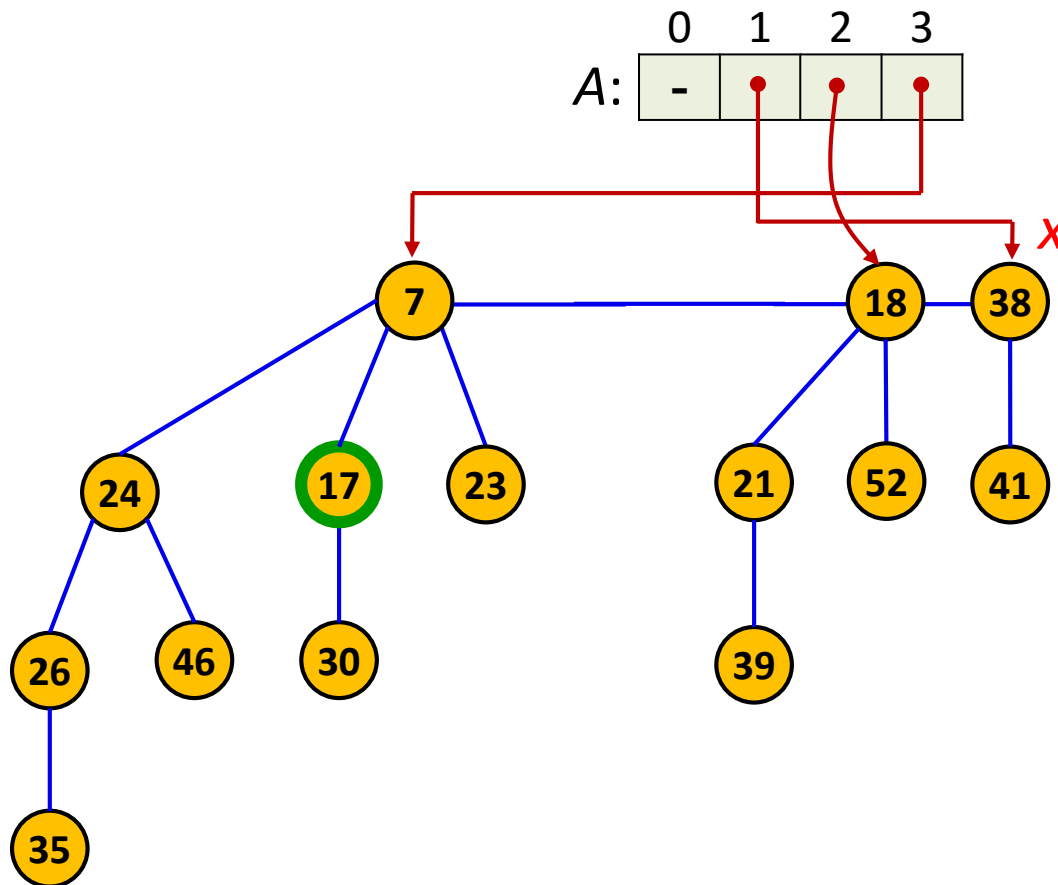
# Уплотнение списка корней (Consolidate)

- Переходим к следующему элементу – узлу **38**
- Степень узла 38 равна 1, ищем корни степени 1:  $A[1] = \text{NULL}$
- Устанавливаем  $A[1] = (38)$



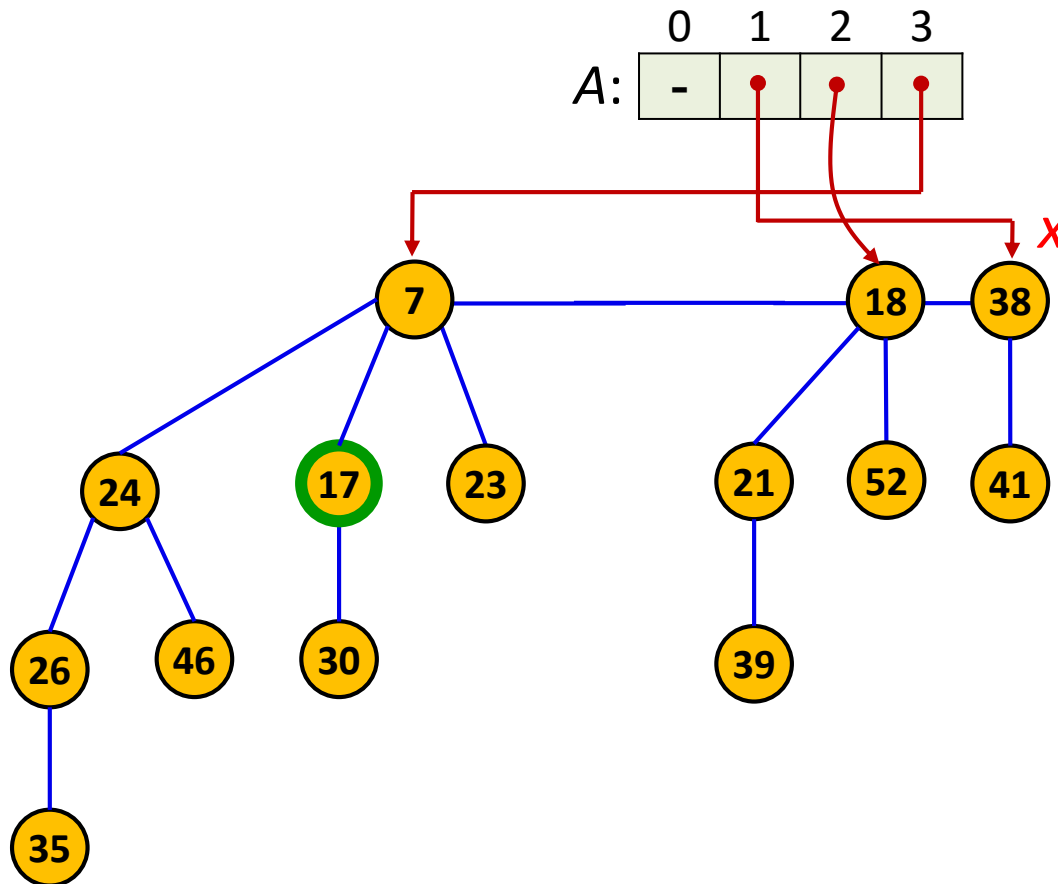
# Уплотнение списка корней (Consolidate)

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



# Уплотнение списка корней (Consolidate)

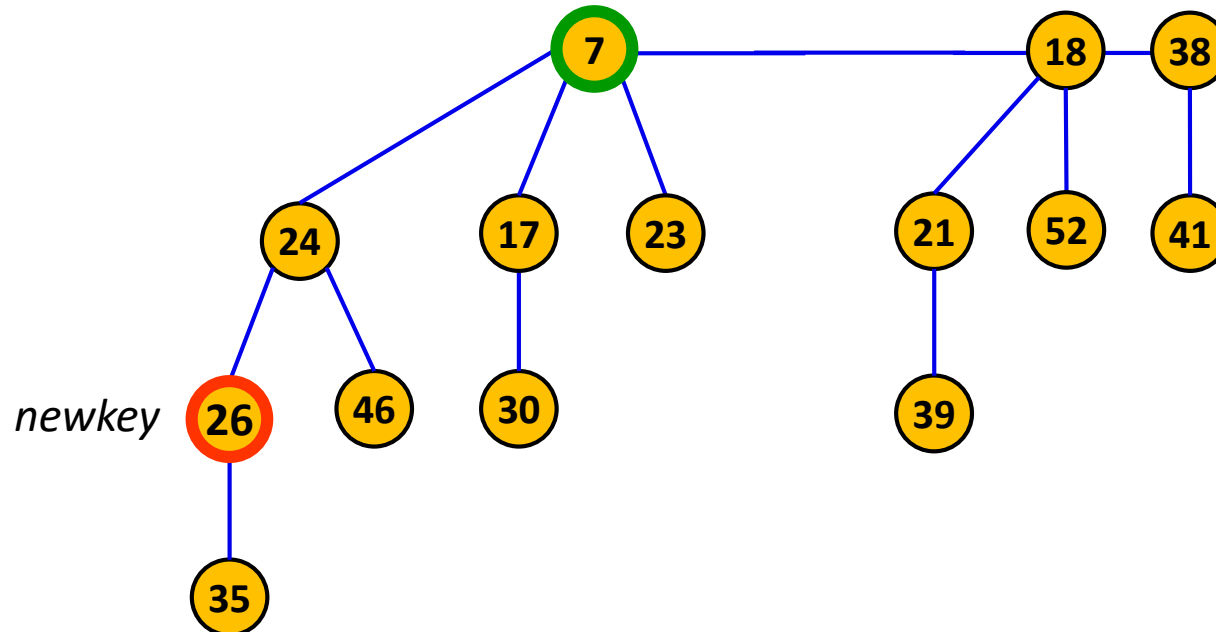
- Ищем новый минимальный узел (устанавливаем `heap.min`)
- Проходим по указателям массива  $A[0, 1, \dots, D(n)] = A[0, \dots, 3]$  и запоминаем указатель на корень с минимальным ключом (требуется  $O(\log n)$  шагов)



# 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)

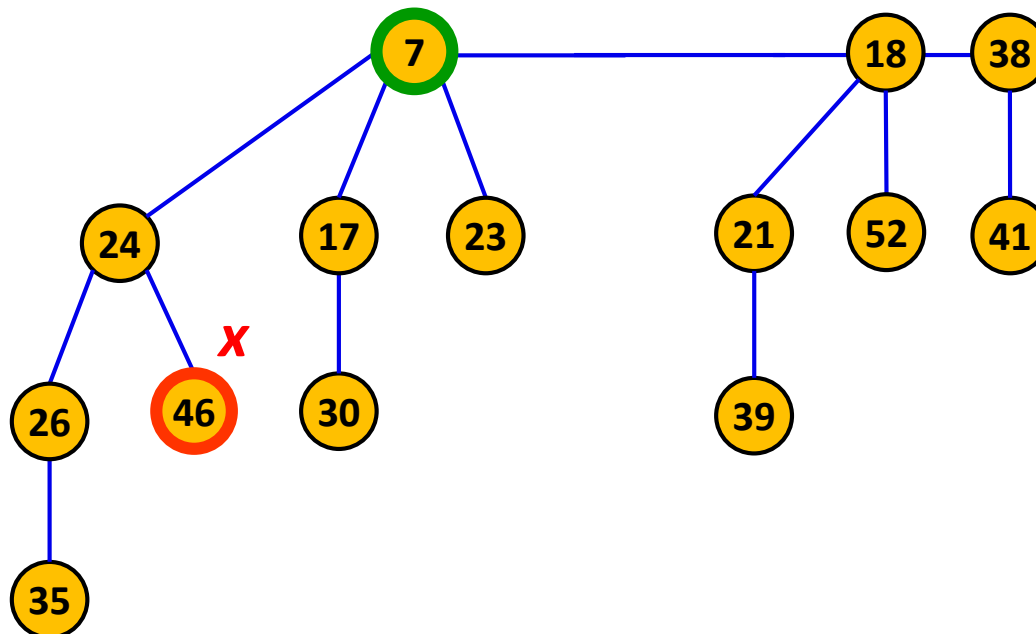
```
function FibHeapCascadingCut(heap, y)
  z = y.parent
  if z = NULL then
    return
  if y.mark = FALSE then
    y.mark = TRUE
  else
    FibHeapCut(heap, y, z)
    FibHeapCascadingCut(heap, z)
  end if
end function
```

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

# Уменьшение ключа (DecreaseKey)

---

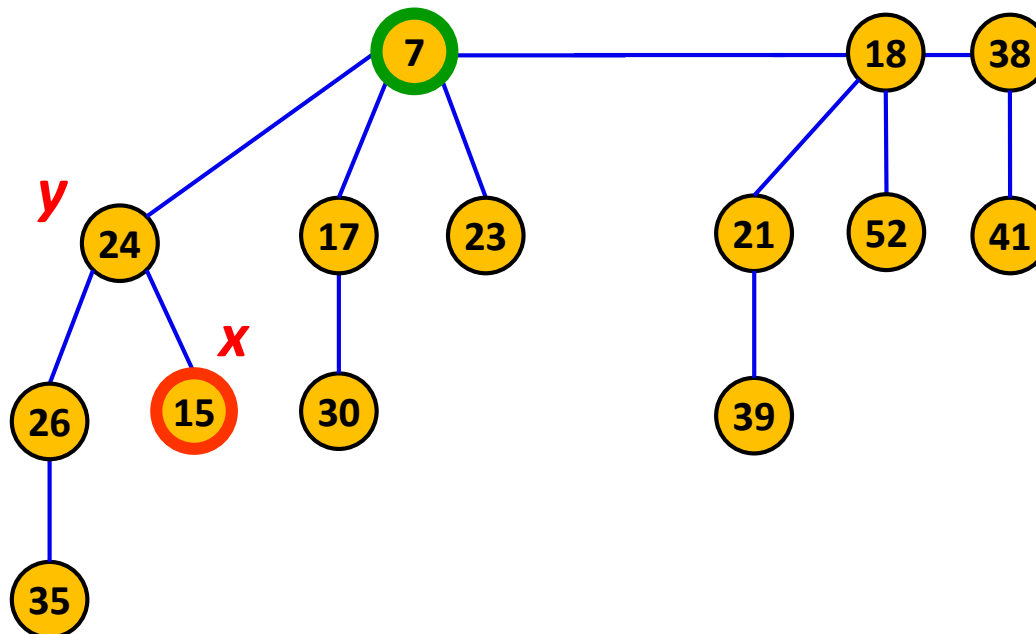
- Изменим ключ 46 до значения 15



# Уменьшение ключа (DecreaseKey)

---

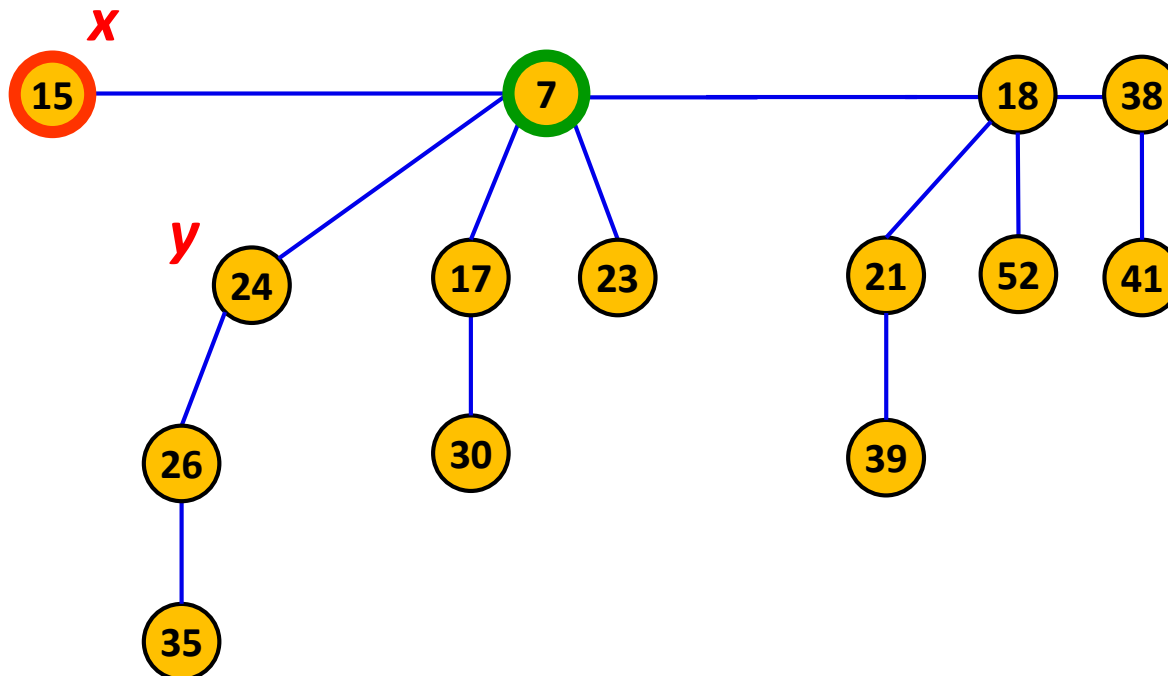
- Изменим ключ 46 до значения 15
- Устанавливаем в узле 46 новый ключ 15
- Нарушены свойства кучи min-heap:  $15 < 24$  ( $x.key < y.key$ )
- Выполняет FibHeapCut(<15>, <24>)
- Выполняем FibHeapCascadingCut(<24>)



# Уменьшение ключа (DecreaseKey)

---

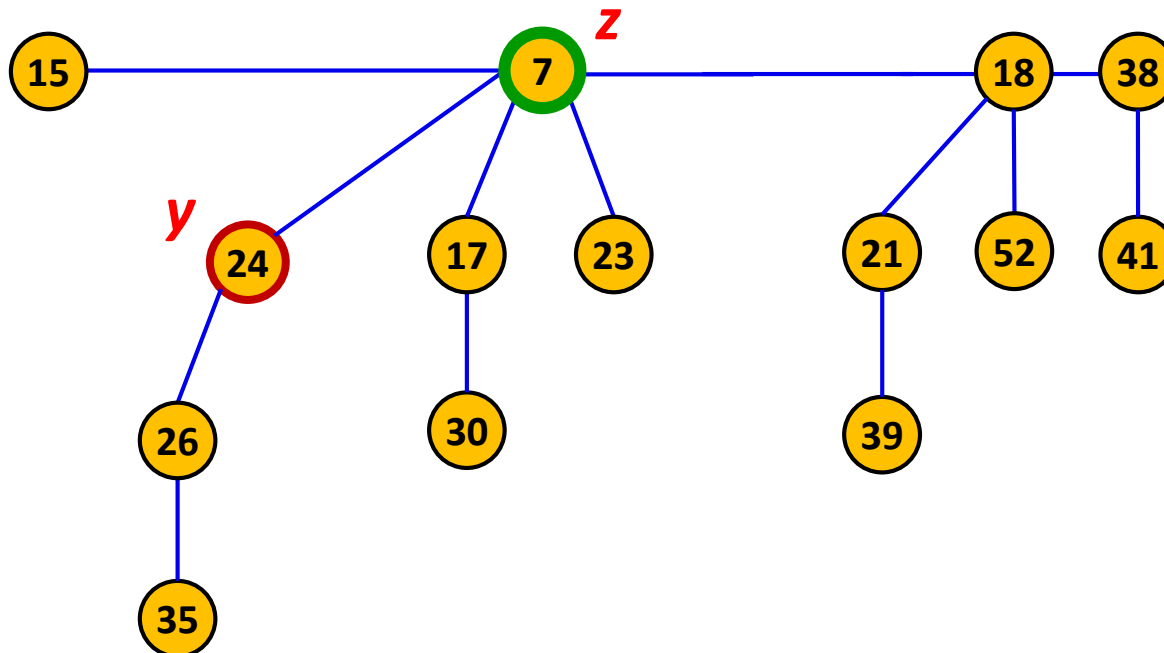
- Выполняет **FibHeapCut**(<15>, <24>)
  - Переносим <15> в список корней кучи
  - Устанавливаем <15>.mark = FALSE



# Уменьшение ключа (DecreaseKey)

---

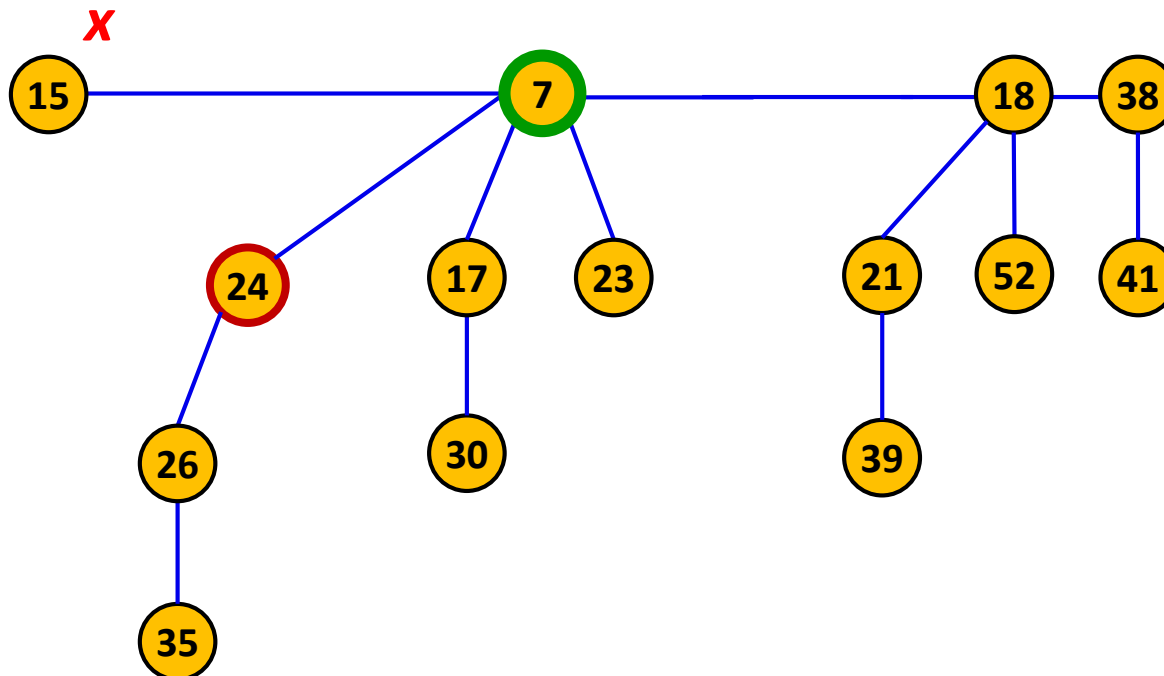
- Выполняем **FibHeapCascadingCut**(<24>)
  - Изменяем значение <24>.mark с FALSE на **TRUE**



# Уменьшение ключа (DecreaseKey)

---

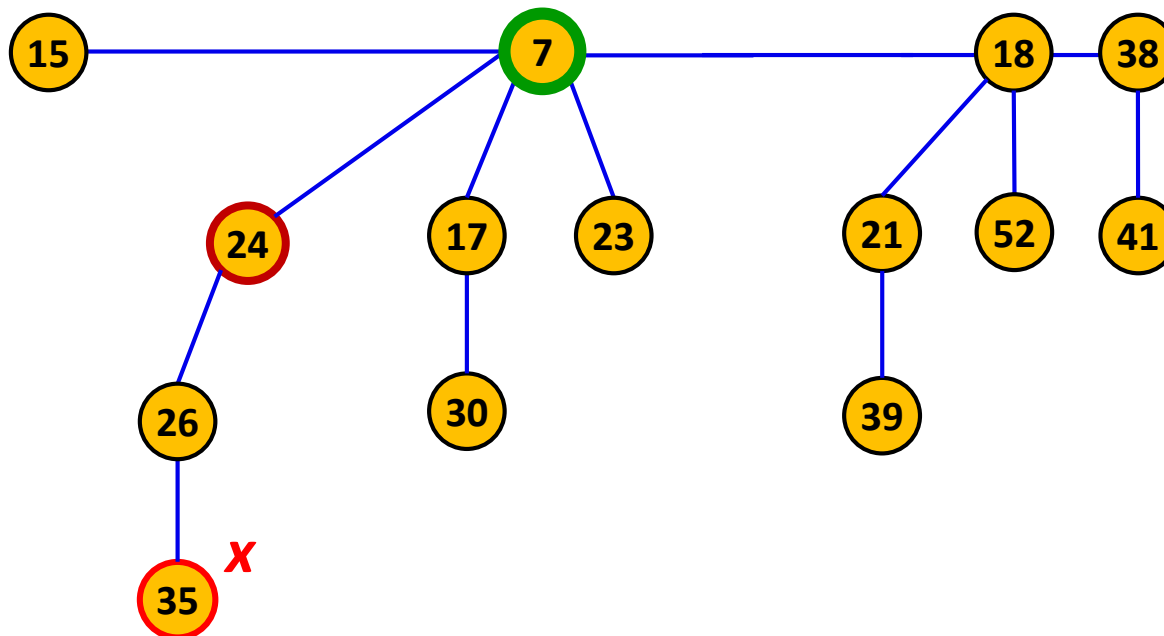
- Корректируем указатель на минимальный элемент
- $15 > 7$ , указатель не изменяется



# Уменьшение ключа (DecreaseKey)

---

- Изменим ключ 35 до значения 5

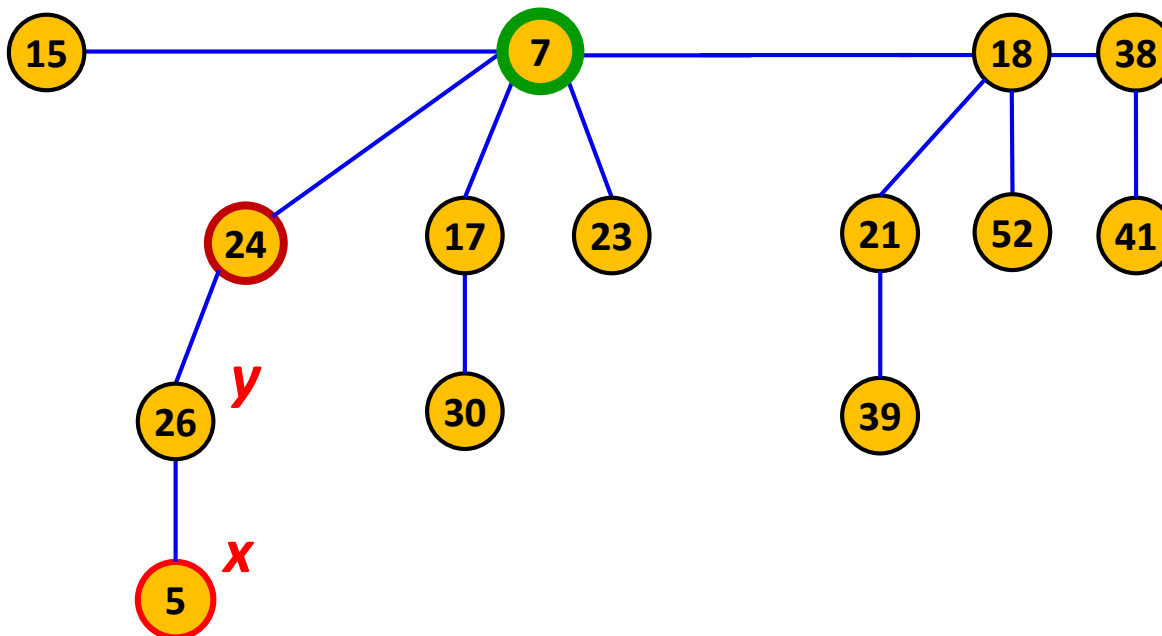




# Уменьшение ключа (DecreaseKey)

---

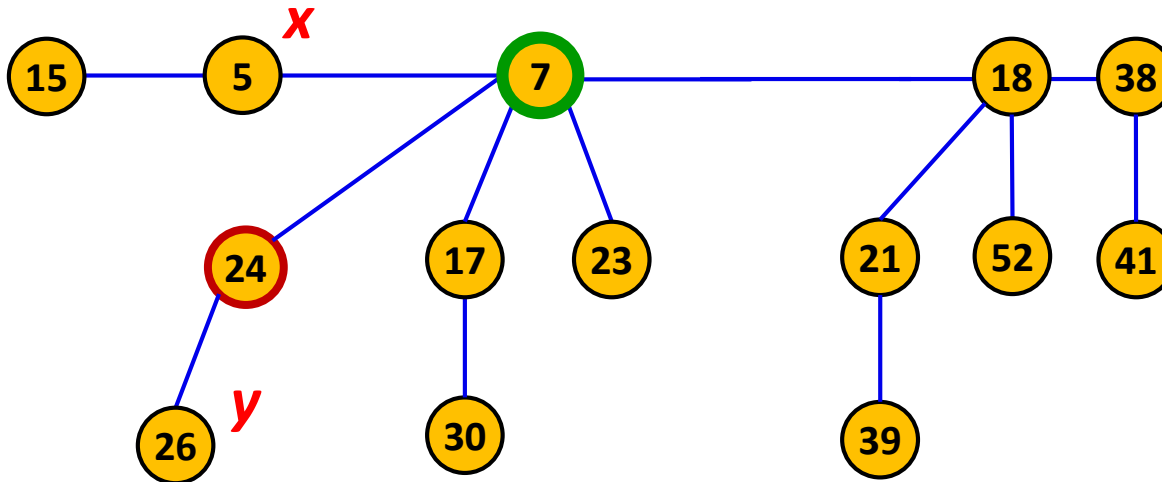
- Изменим ключ 35 до значения 5
- Устанавливаем в узле 35 новый ключ 5
- Нарушены свойства кучи min-heap:  $5 < 25$  ( $x.key < y.key$ )
- Выполняет FibHeapCut(<5>, <26>)
- Выполняем FibHeapCascadingCut(<26>)



# Уменьшение ключа (DecreaseKey)

---

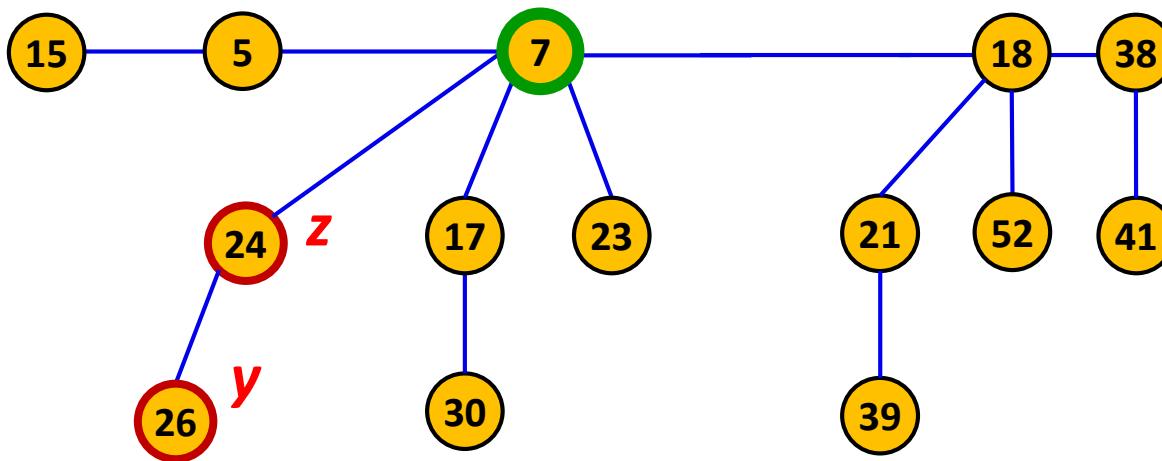
- Выполняет **FibHeapCut**(<15>, <24>)
  - Переносим <15> в список корней кучи
  - Устанавливаем <15>.mark = FALSE



# Уменьшение ключа (DecreaseKey)

---

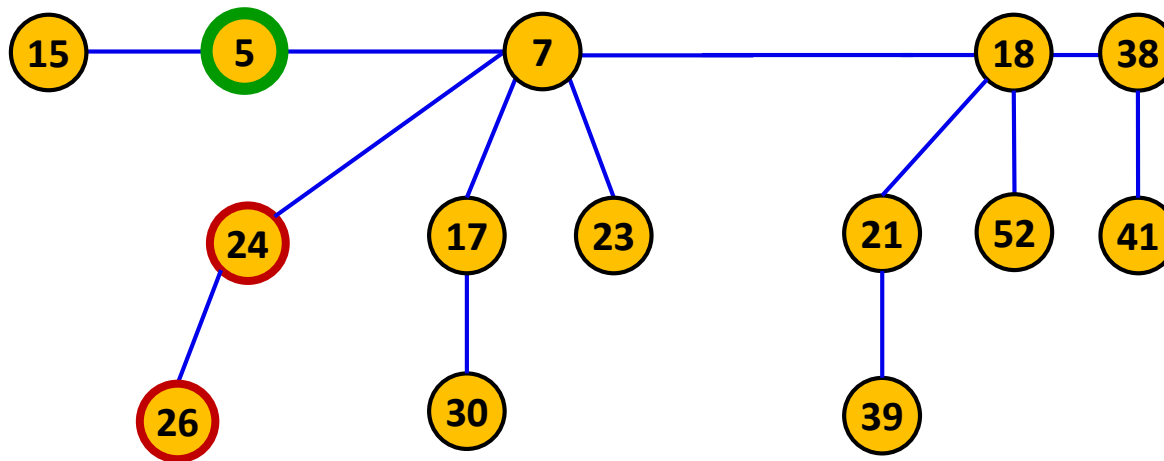
- Выполняем **FibHeapCascadingCut**(<26>)
  - Изменяем значение <26>.mark с FALSE на **TRUE**



# Уменьшение ключа (DecreaseKey)

---

- Корректируем указатель на минимальный элемент
- $5 < 7$ , указатель не изменяется



# Удаление заданного узла (Delete)

---

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

# Очередь с приоритетом (Priority queue)

- В таблице приведены трудоемкости операций очереди 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	$\Theta(n)$	$\Omega(\log n)$	$\Theta(1)$	$\Theta(1)^*$	$\Theta(1)$

# Задания

---

- Прочитать в **CLRS 3ed.**  
§ “19.2 Операции над объединяемыми пирамидами”
- Разобраться с доказательством амортизированной вычислительной сложности операций (метод потенциалов)
- Прочитать в **CLRS 3ed.**  
§ “19.4 Оценка максимальной степени”