Курс «Алгоритмы и алгоритмические языки»

Лекция 20

Красно-черные деревья

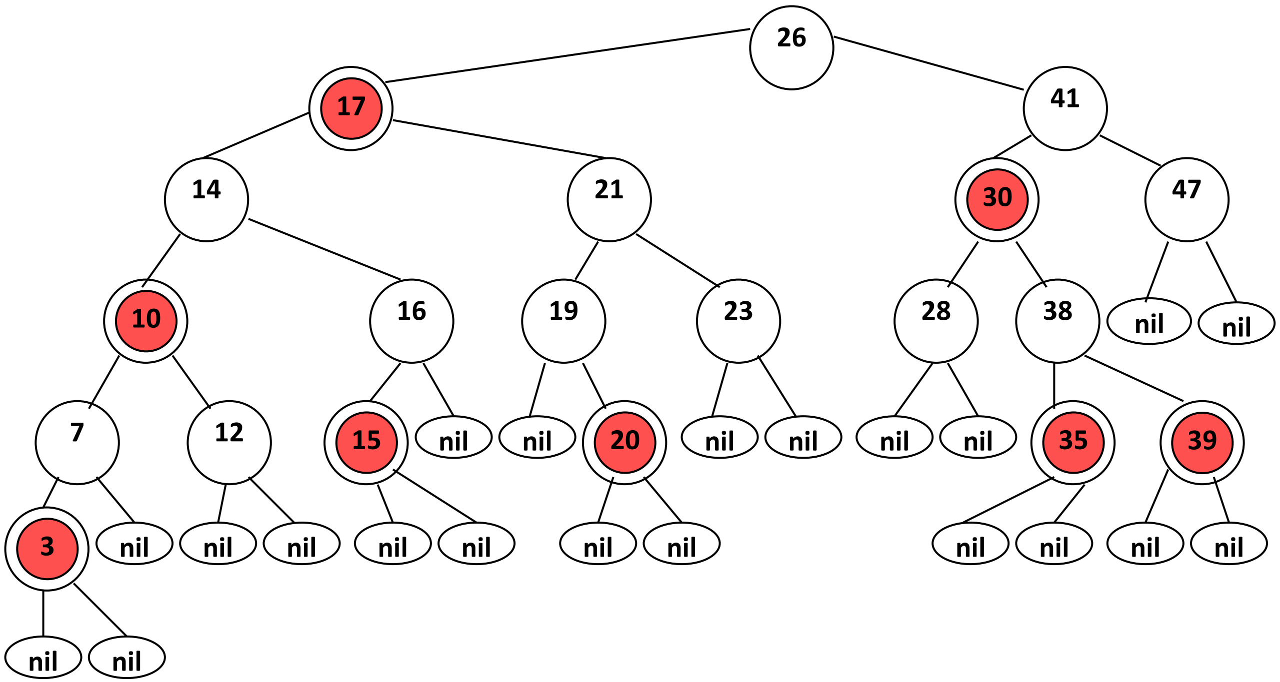
- Красно-черное дерево – двоичное дерево поиска, каждая вершина которого окрашена либо в красный, либо в черный цвет
- Поля – цвет, дети, родители

```
typedef struct rbtree {
  int key;
  char color;
  struct rbtree *left, *right, *parent;
} rbtree, *prbtree;
```

- Будем считать, что если `left` или `right` равны `NULL`, то это “указатели” на фиктивные листы, т.е. все вершины внутренние

Красно-черные деревья

- Свойства красно-черных деревьев:
  1. Каждая вершина либо красная, либо черная.
  2. Каждый лист (фиктивный) – черный.
  3. Если вершина красная, то оба ее сына – черные.
  4. Все пути, идущие от корня к любому листу, содержат одинаковое количество черных вершин



Красно-черные деревья

- Обозначим $bh(x)$ – "черную" высоту поддерева с корнем x (саму вершину в число не включаем), т.е. количество черных вершин от x до листа
- Черная высота дерева – черная высота его корня
- *Лемма*: Красно-черное дерево с n внутренними вершинами (без фиктивных листьев) имеет высоту не более $2log_2(n+1)$.

  (1) Покажем вначале, что поддерево x содержит не меньше $2^{bh(x)} – 1$ внутренних вершин

  (1а) Индукция. Для листьев $bh = 0$, т.е. $2^{bh(x)} – 1 = 2^0 – 1 = 0$.

  (1б) Пусть теперь x – не лист и имеет черную высоту k. Тогда каждый ее сын имеет черную высоту не меньше $k – 1$ (красный сын имеет высоту k, черный – $k – 1$).

  (1в) По предположению индукции каждый сын имеет не меньше $2^{k-1} – 1$ вершин. Поэтому поддерево x имеет не меньше $2^{k-1} – 1 + 2^{k-1} – 1 + 1 = 2^k – 1$.

Красно-черные деревья

- *Лемма*: Красно-черное дерево с n внутренними вершинами (без фиктивных листьев) имеет высоту не более $2log_2(n+1)$.

  (2) Теперь пусть высота дерева равна h.

  (2а) По свойству 3 черные вершины составляют не меньше половины всех вершин на пути от корня к листу. Поэтому черная высота дерева bh не меньше $h/2$.

  (2б) Тогда $n \geq 2^{h/2} - 1$ и $h \leq 2log_2(n + 1)$. Лемма доказана.

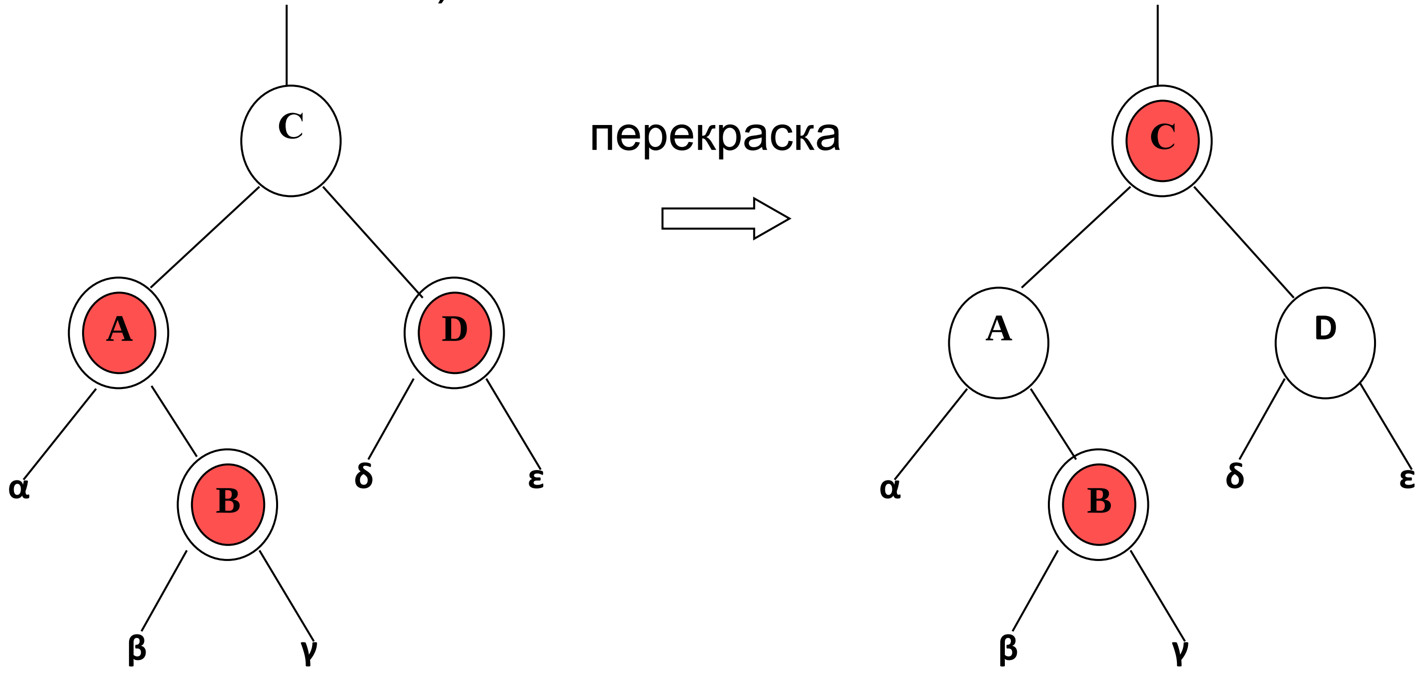
- Следовательно, поиск по красно-черному дереву имеет сложность $O\ (log_2 n)$.

Красно-черные деревья: вставка вершины

- Сначала мы используем обычную процедуру занесения новой вершины в двоичное дерево поиска:
  - красим новую вершину в красный цвет.
- Если дерево было пустым, то красим новый корень в черный цвет
- Свойство 4 при вставке изначально не нарушено, т.к. новая вершина красная
- Если родитель новой вершины черный (новая – красная), то свойство 3 также не нарушено
- Иначе (родитель красный) свойство 3 нарушено

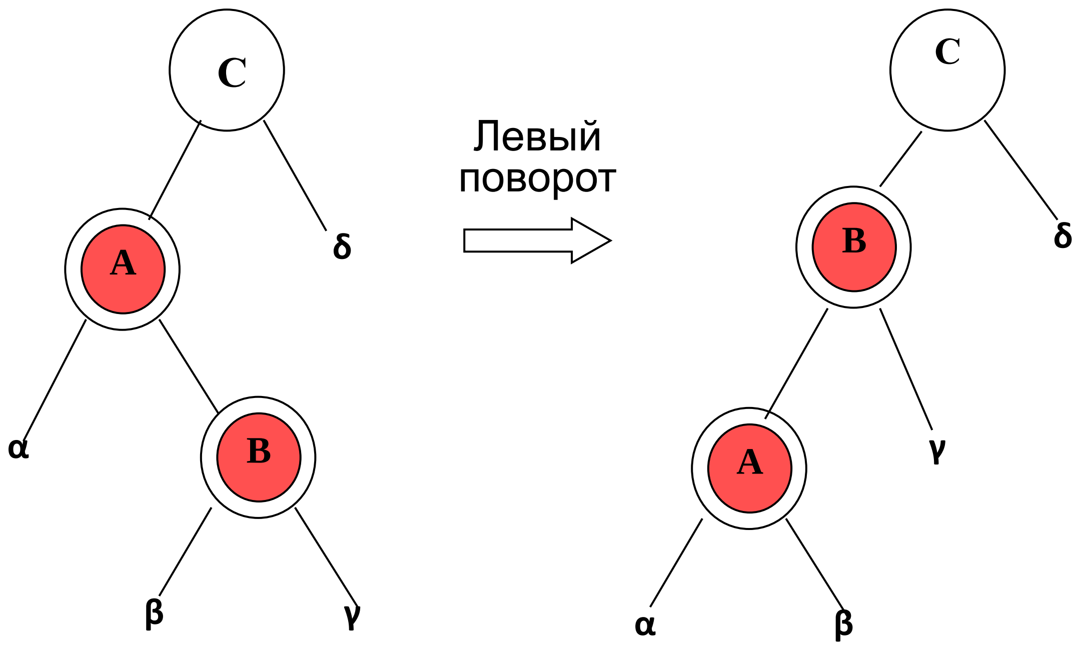
Красно-черные деревья: вставка вершины

- Случай 1: “дядя” (второй сын родителя родителя текущей вершины) тоже красный (как текущая вершина и родитель)
  - Возможно выполнить перекраску:
    родителя и дядю (вершины A и D) – в черный цвет,
    деда – (вершина C) – в красный цвет
  - Свойство 4 не нарушено (черные высоты поддеревьев совпадают)



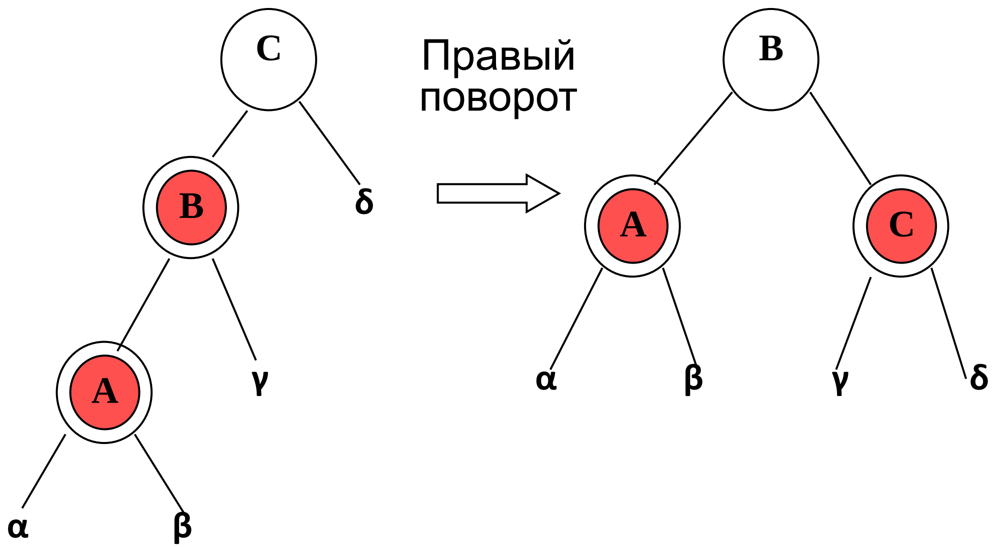
Красно-черные деревья: вставка вершины

- Случай 2: “дядя” (второй сын родителя родителя текущей вершины) черный
  - Шаг 1: Необходимо выполнить левый поворот текущей вершины (вершины B)



Красно-черные деревья: вставка вершины

- Случай 2: “дядя” (второй сын родителя родителя текущей вершины) черный
  - Шаг 2: Необходимо выполнить правый поворот вершины C, после чего перекрасить вершины B и C
  - Все поддеревья имеют черные корни и одинаковую черную высоту, поэтому свойства 3 и 4 верны



Пирамидальная сортировка (heapsort)

- Можно использовать дерево поиска для сортировки
- Например, последовательный поиск минимального элемента, удаление его и вставка в отсортированный массив
  - Сложность такого алгоритма есть $O(nh)$, где h – высота дерева
- Недостатки:
  - Требуется дополнительная память для дерева
  - Требуется построить само дерево (с минимальной высотой)
- Можно ли построить похожий алгоритм без требований к дополнительной памяти?

Пирамидальная сортировка: пирамида (двоичная куча)

- Рассматриваем массив **`a`** как двоичное дерево:
  - Элемент **`a[i]`** является узлом дерева
  - Элемент **`a[i/2]`** является родителем узла **`a[i]`**
  - Элементы **`a[2*i]`** и **`a[2*i+1]`** являются детьми узла **`a[i]`**

- Для всех элементов пирамиды выполняется соотношение (основное свойство кучи):

  `a[i] >= a[2*i]` и **`a[i] >= a[2*i+1]`**

  или

  `a[i/2] <= a[i]`

  - Сравнение может быть как в большую, так и в меньшую сторону

- **Замечание**. Определение предполагает нумерацию элементов массива от 1 до n
  - Для нумерации от 0 до n-1:

  `a[i] >= a[2*i+1]` и **`a[i] >= a[2*i+2]`**

Пирамидальная сортировка: пирамида (двоичная куча)

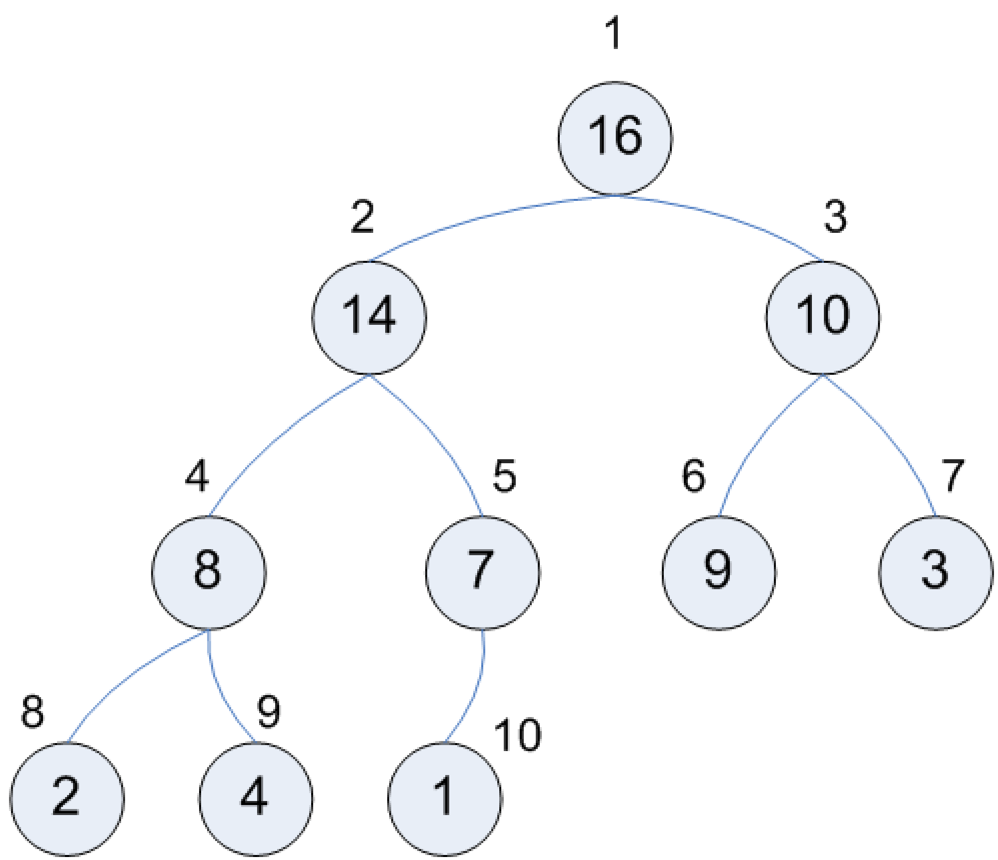
- Для всех элементов пирамиды выполняется соотношение:

  `a[i] >= a[2*i]` и **`a[i] >= a[2*i+1]`**

  или

  `a[i/2] <= a[i]`

  - Сравнение может быть как в большую, так и в меньшую сторону



Пирамидальная сортировка: просеивание элемента

- Как добавить элемент в уже существующую пирамиду?
- Алгоритм:
  - Поместим новый элемент в корень пирамиды
  - Если этот элемент меньше одного из сыновей:
    - Элемент меньше наибольшего сына
    - Обменяем элемент с наибольшим сыном (это позволит сохранить свойство пирамиды для другого сына)
    - Повторим процедуру для обмененного сына

Пирамидальная сортировка: просеивание элемента

```
static void sift (int *a, int l, int r) {
  int i, j, x;

  i = l; j = 2*l; x = a[l];
  /* j указывает на наибольшего сына */
  if (j < r && a[j] < a[j + 1])
    j++;
  /* i указывает на отца */
  while (j <= r && x < a[j]) {
    /* обмен с наибольшим сыном: a[i] == x */
    a[i] = a[j]; a[j] = x;
    /* продвижение индексов к следующему сыну */
    i = j; j = 2*j;
    /* выбор наибольшего сына */
    if (j < r && a[j] < a[j + 1])
      j++;
  }
}
```

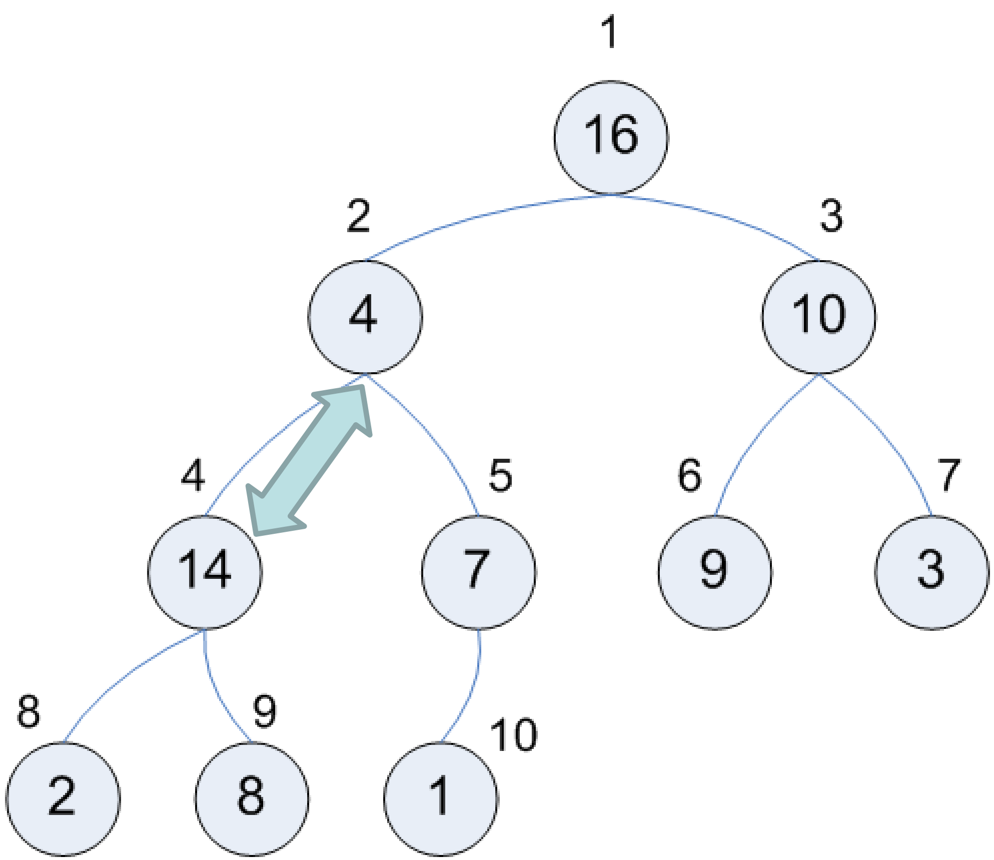
Пирамидальная сортировка: просеивание элемента

```
/* l, r - от 0 до n-1 */
static void sift (int *a, int l, int r) {
  int i, j, x;

  /* Теперь l, r, i, j от 1 до n, а индексы массива
     уменьшаются на 1 при доступе */
  l++, r++;
  i = l; j = 2*l; x = a[l-1];
  /* j указывает на наибольшего сына */
  if (j < r && a[j-1] < a[j])
    j++;
  /* i указывает на отца */
  while (j <= r && x < a[j-1]) {
    /* обмен с наибольшим сыном: a[i-1] == x */
    a[i-1] = a[j-1]; a[j-1] = x;
    /* продвижение индексов к следующему сыну */
    i = j; j = 2*j;
    /* выбор наибольшего сына */
    if (j < r && a[j-1] < a[j])
      j++;
  }
}
```

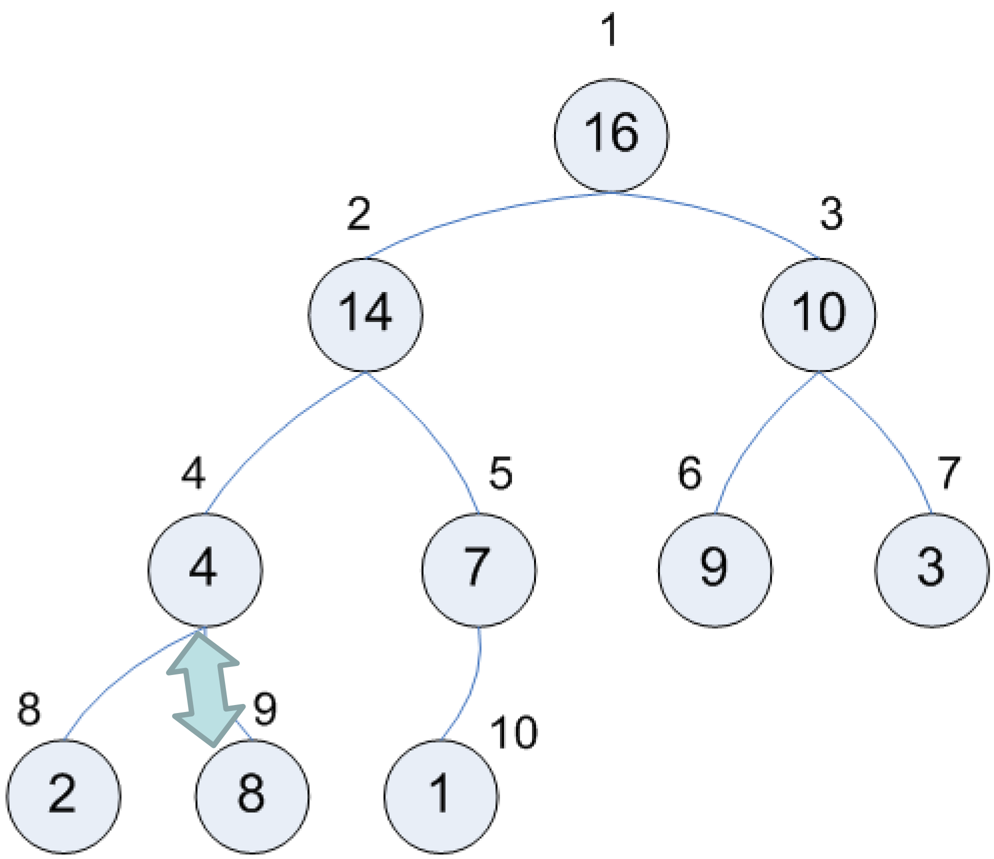
Пирамидальная сортировка: просеивание элемента

- Вызов sift (2, 10) для левого поддерева



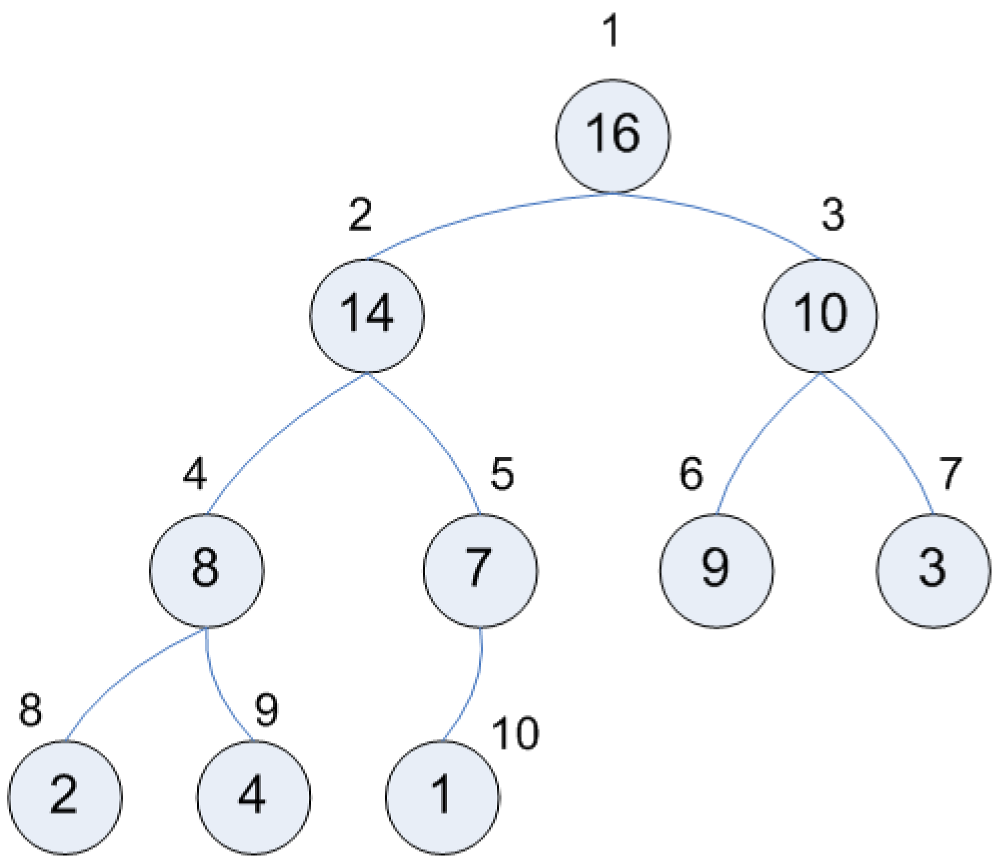
Пирамидальная сортировка: просеивание элемента

- Вызов sift (2, 10) для левого поддерева



Пирамидальная сортировка: просеивание элемента

- Вызов sift (2, 10) для левого поддерева



Пирамидальная сортировка: алгоритм

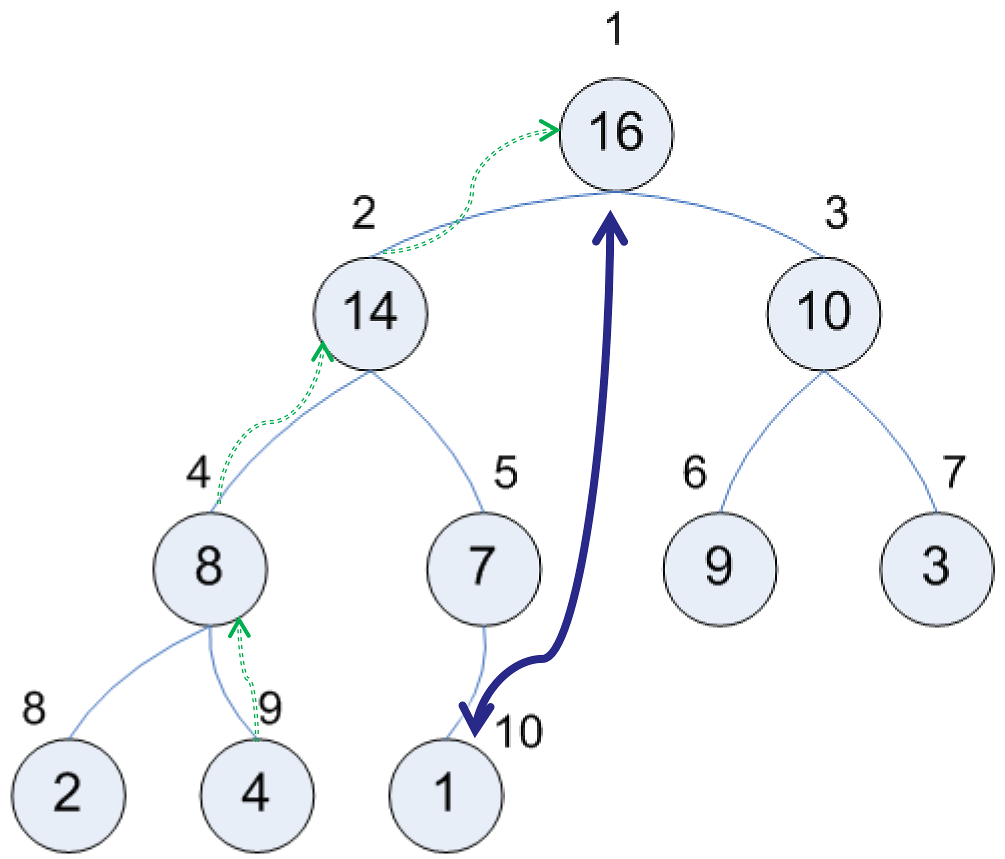
- (1) Построим пирамиду по сортируемому массиву
  - Элементы массива от n/2 до n являются листьями дерева, а следовательно, правильными пирамидами из одного элемента
  - Для остальных элементов в порядке уменьшения индекса просеиваем их через правую часть массива
- (2) Отсортируем массив по пирамиде
  - Первый элемент массива максимален (корень пирамиды)
  - Поменяем первый элемент с последним (таким образом, последний элемент отсортирован)
  - Теперь для первого элемента свойство кучи нарушено: повторим просеивание первого элемента в пирамиде от первого до предпоследнего
  - Снова поменяем первый и предпоследний элемент и т.п.

Пирамидальная сортировка: программа

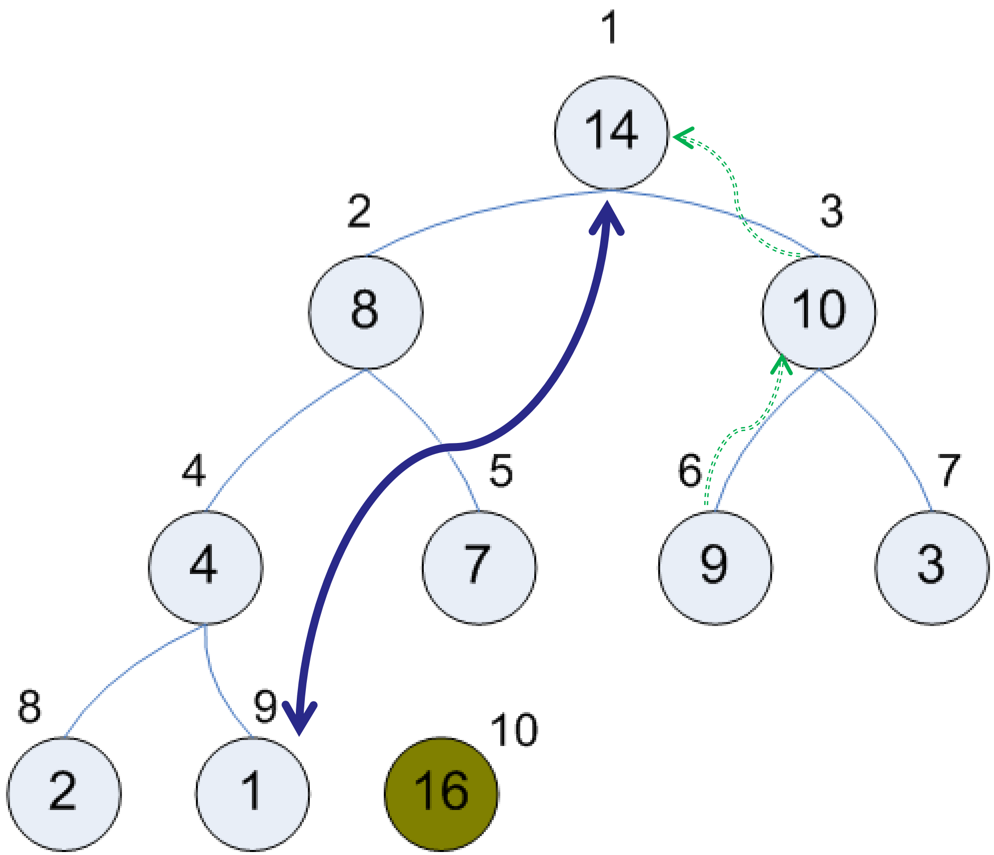
```
void heapsort (int *a, int n) {
  int i, x;

  /* Построим пирамиду по сортируемому массиву */
  /* Элементы нумеруются с 0 -> идем от n/2-1 */
  for (i = n/2 - 1; i >= 0; i--)
    sift (a, i, n - 1);
  for (i = n – 1; i > 0; i--) {
    /* Текущий максимальный элемент в конец */
    x = a[0]; a[0] = a[i]; a[i] = x;
    /* Восстановим пирамиду в оставшемся массиве */
    sift (a, 0, i – 1);
  }
}
```

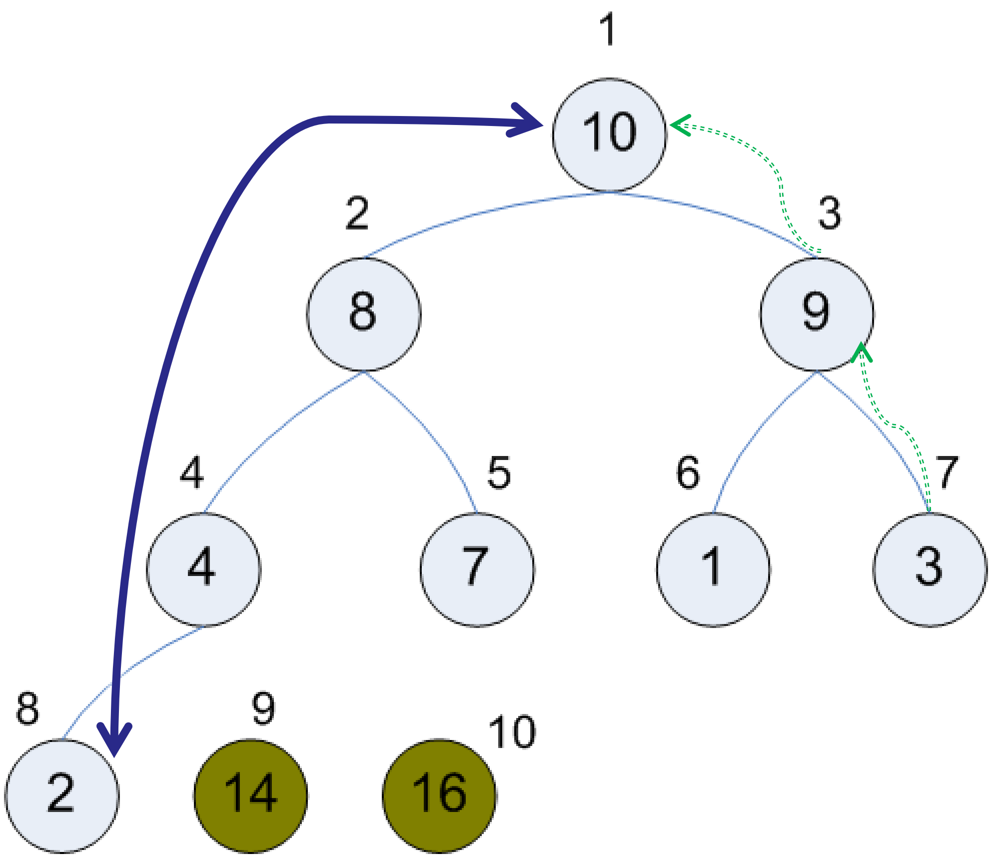
Пирамидальная сортировка: пример



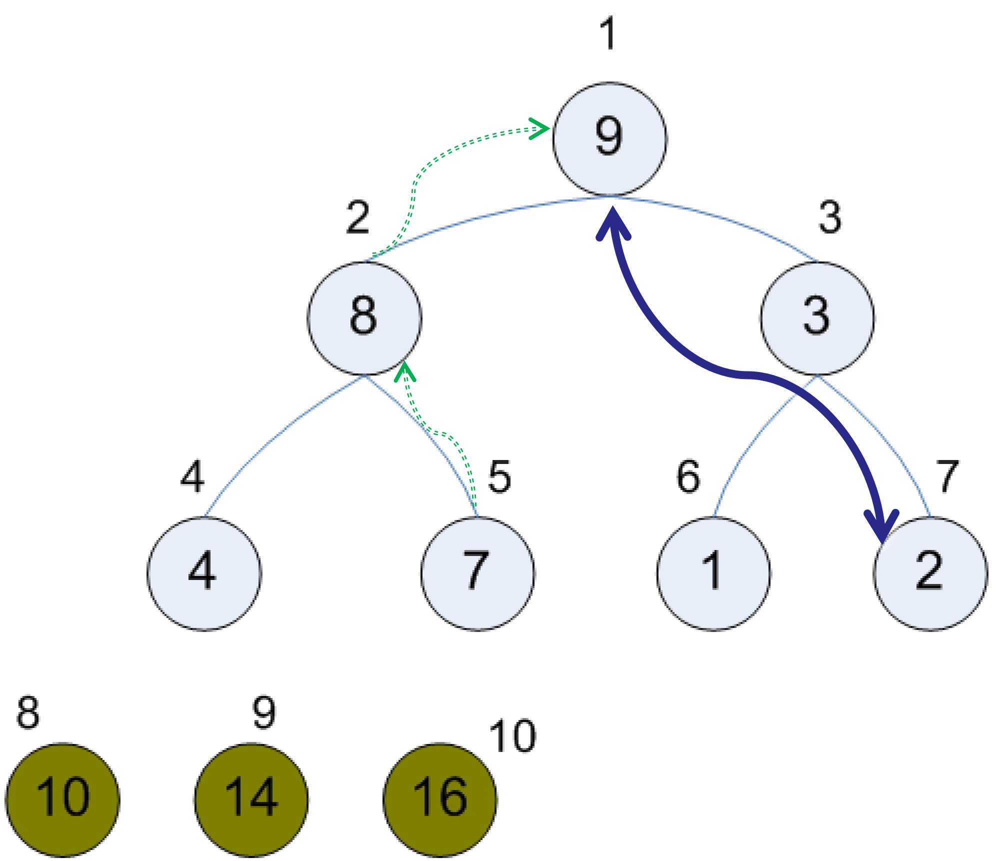
Пирамидальная сортировка: пример



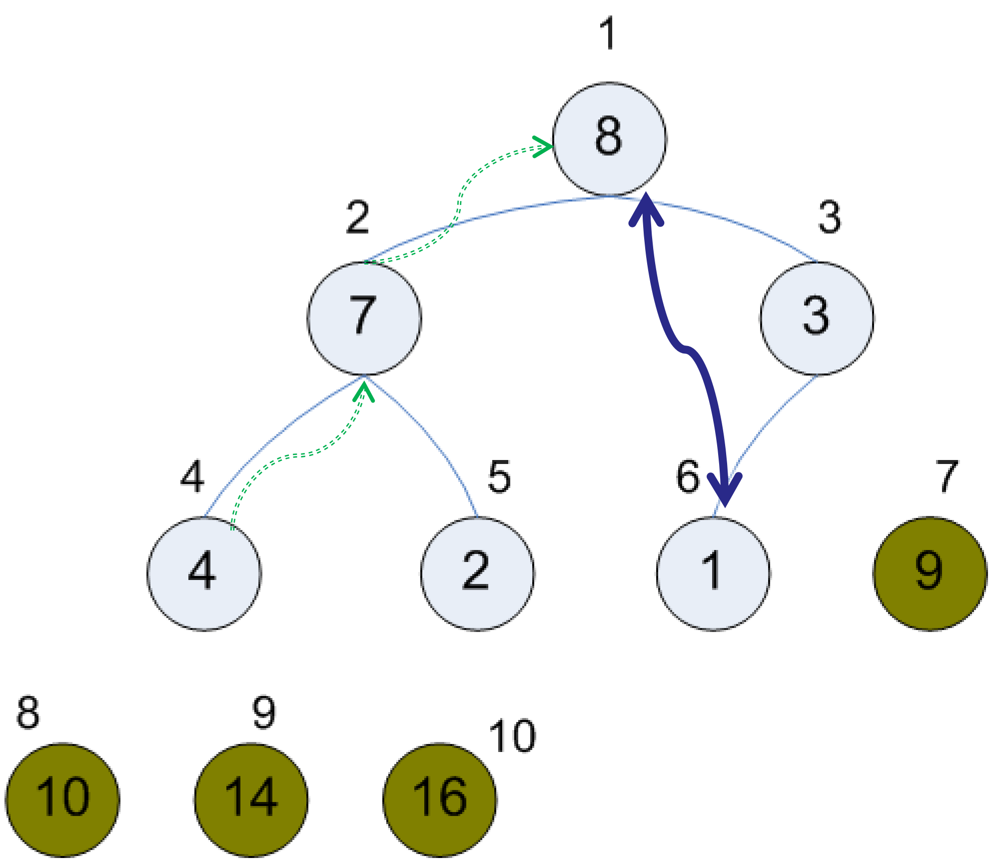
Пирамидальная сортировка: пример



Пирамидальная сортировка: пример



Пирамидальная сортировка: пример



Пирамидальная сортировка: сложность алгоритма

- (1) Построим пирамиду по сортируемому массиву
  - Элементы массива от n/2 до n являются листьями дерева, а следовательно, правильными пирамидами из 1 элемента
  - Для остальных элементов в порядке уменьшения индекса просеиваем их через правую часть массива
- (2) Отсортируем массив по пирамиде
  - Первый элемент массива максимален (корень пирамиды)
  - Поменяем первый элемент с последним (таким образом, последний элемент отсортирован)
  - Теперь для первого элемента свойство кучи нарушено: повторим просеивание первого элемента в пирамиде от первого до предпоследнего
  - Снова поменяем первый и предпоследний элемент и т.п.

- Сложность этапа построения пирамиды есть $O(n)$
- Сложность этапа сортировки есть $O(n \log n)$
- Сложность в худшем случае также $O(n \log n)$
- Среднее количество обменов – $n/2 * \log n$