Структуры данных
Деревья

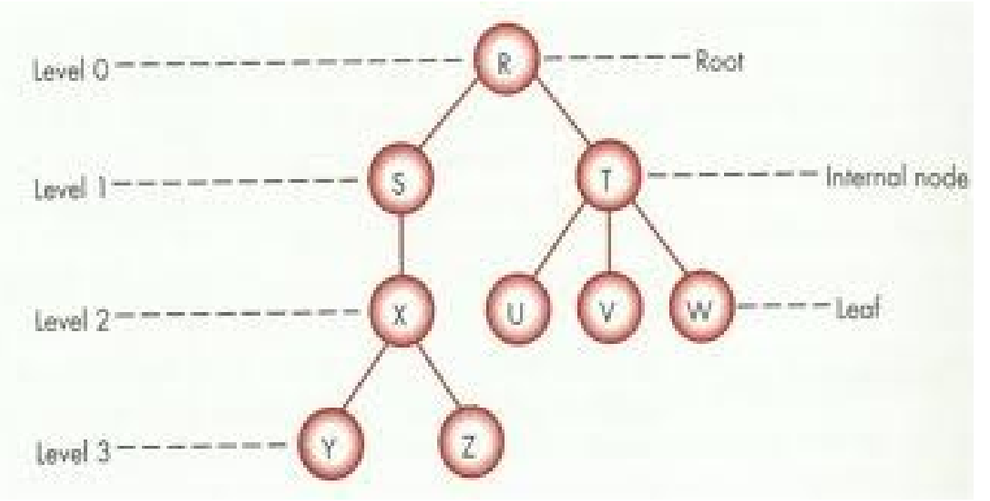
Основы языка C, лекция 14

Упорядоченные структуры

- Есть ли элемент в структуре?
- Список — удобно вставлять O(1),
  неудобно искать O(n)
- Динамический массив — удобно искать O(log n),
  неудобно вставлять O(n)
- Дерево
  - Бинарное дерево поиска

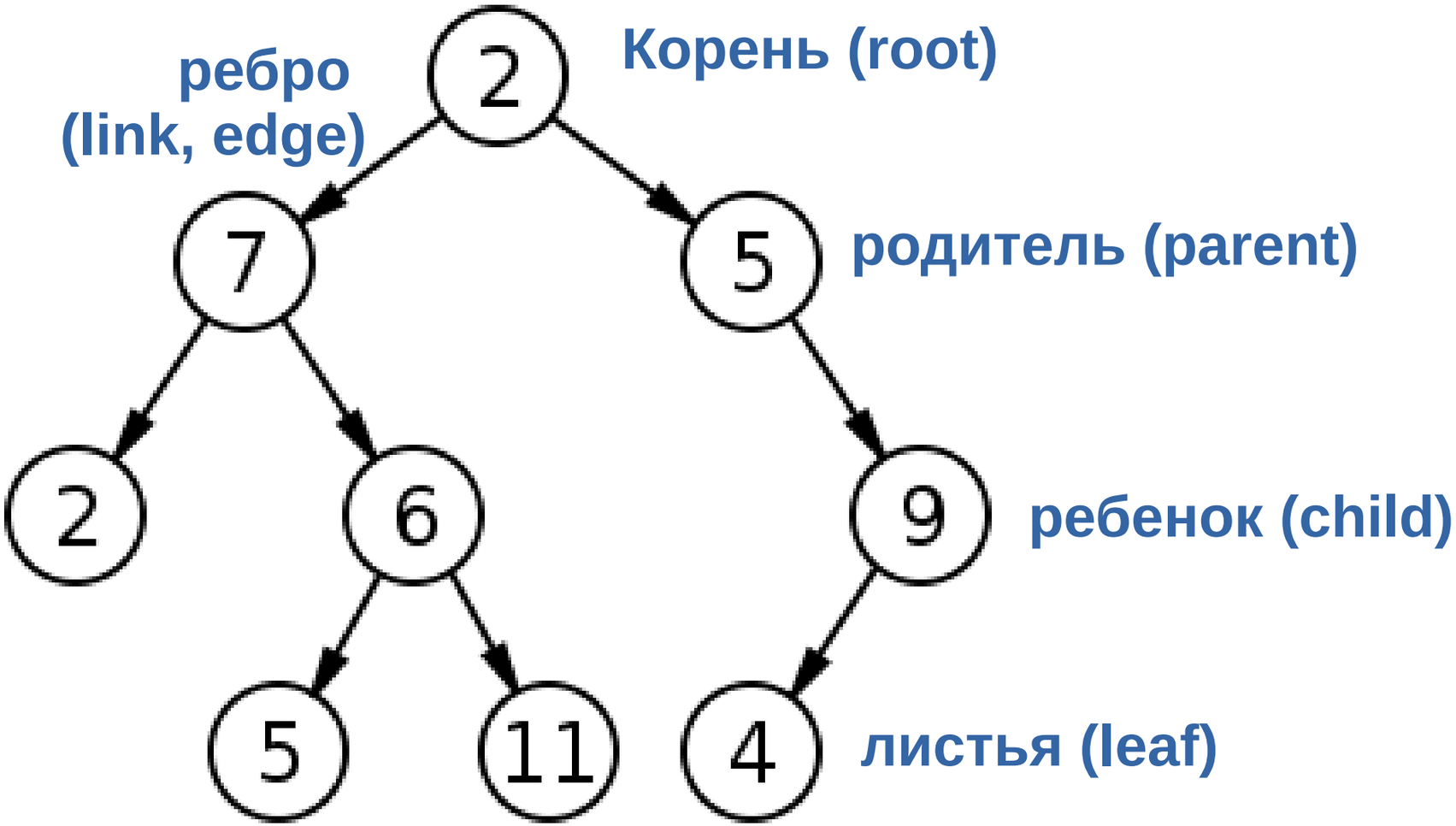
Теория графов

- Дерево — связный ациклический граф
- Вершина (узел) множество V(G)
- Ребро — соединяет две вершины графа
- Корень — выбранная вершина дерева
- Родитель / потомок (ребенок)
- Лист — узел без потомков (5)
- Глубина дерева (3)
- Поддерево
-
- Файловая система



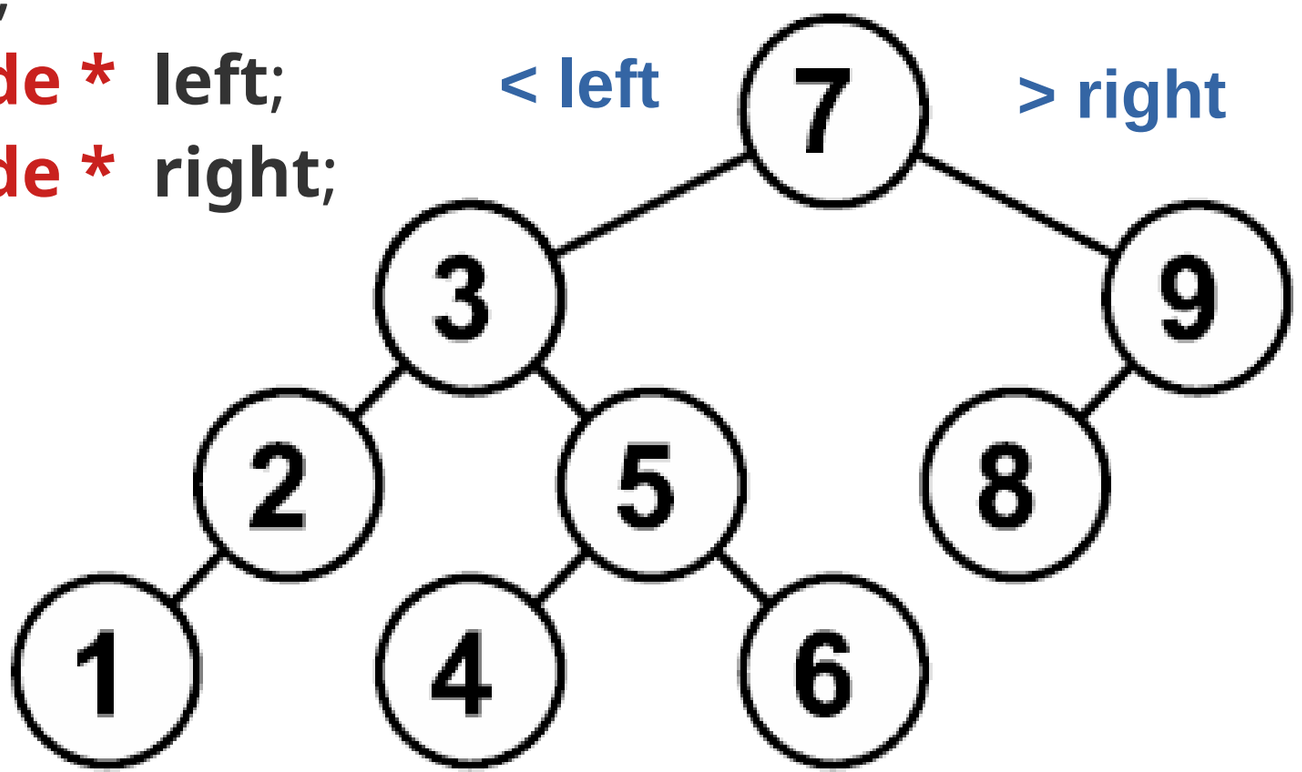
Неупорядоченное бинарное дерево

- бинарные деревья, где каждая вершина имеет 1 входящее ребро и не более 2 выходящих

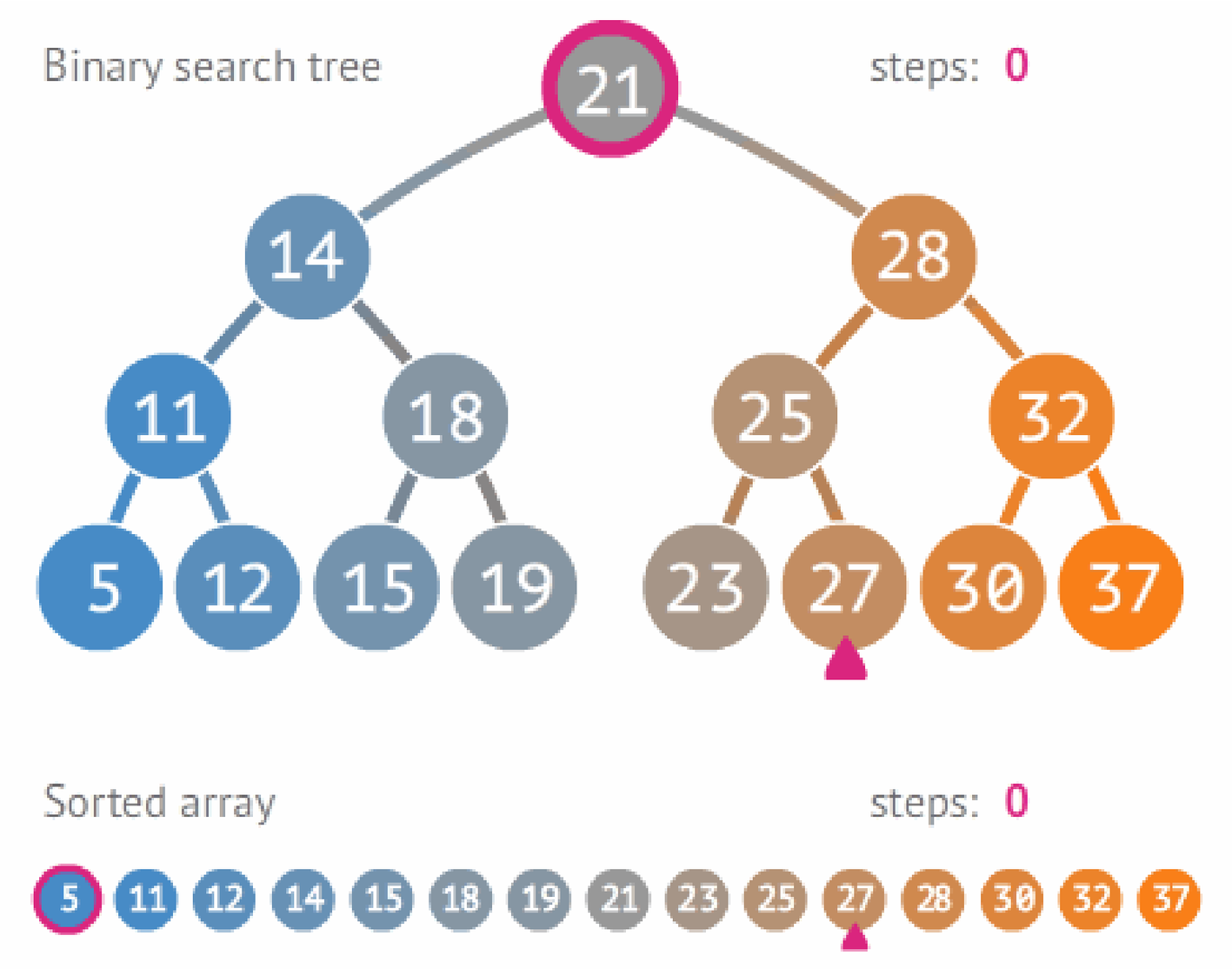


Двоичное дерево поиска BST

- Упорядоченное дерево
- typedef int **Data**;
  typedef struct **Node** {
  Data data;
  struct Node * left;
  struct Node * right;
  } **Node**;



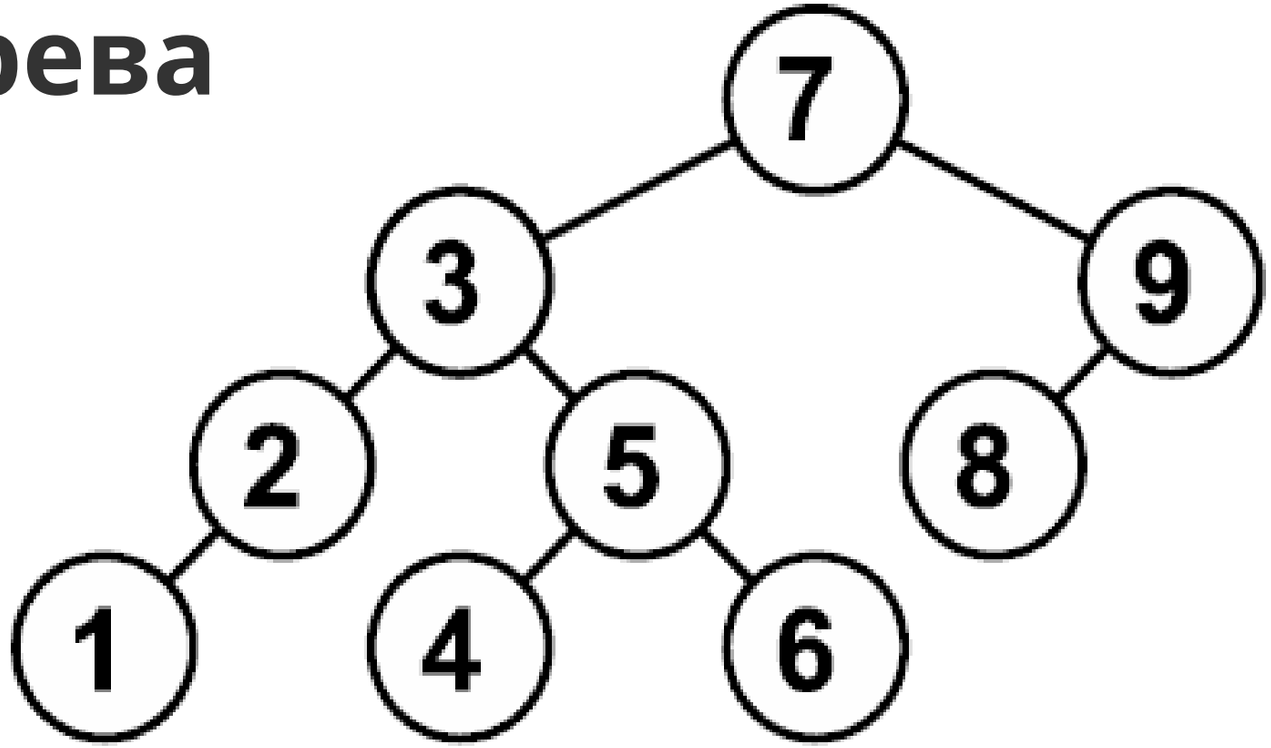
Поиск O(log n) или O(n)



- By A.gholamzade - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=49280824

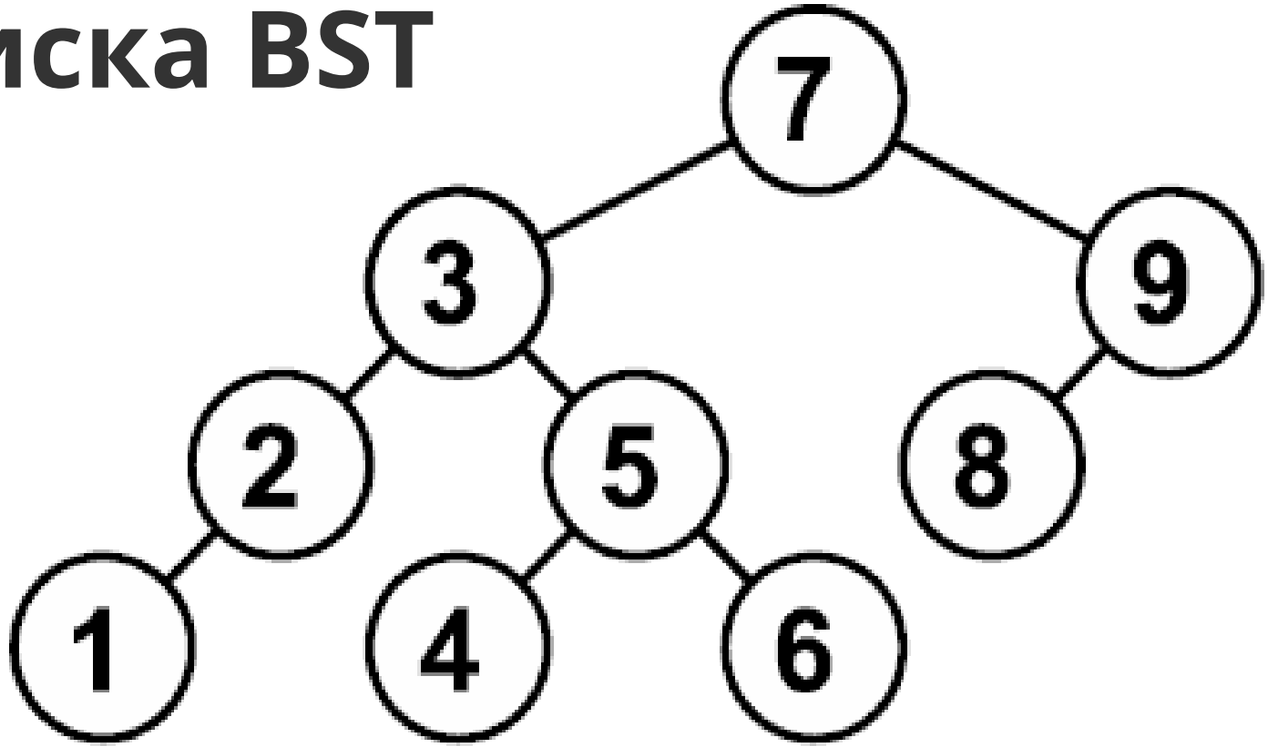
Построение дерева

- **Node** * tree = **NULL**;
- tree = **add** (tree, **7**);
- tree = **add** (tree, **3**);
- tree = **add** (tree, **2**);
- tree = **add** (tree, **1**);
- tree = **add** (tree, **9**);
- tree = **add** (tree, **5**);
- tree = **add** (tree, **4**);
- tree = **add** (tree, **6**);
- tree = **add** (tree, **8**);
- **print** (tree); // 1 2 3 4 5 6 7 8 9



Двоичное дерево поиска BST

- typedef int **Data**;
  typedef struct **Node** {
  Data data; // данные
  struct Node * left;
  struct Node * right;
  } **Node**;
  Node * tree = NULL;



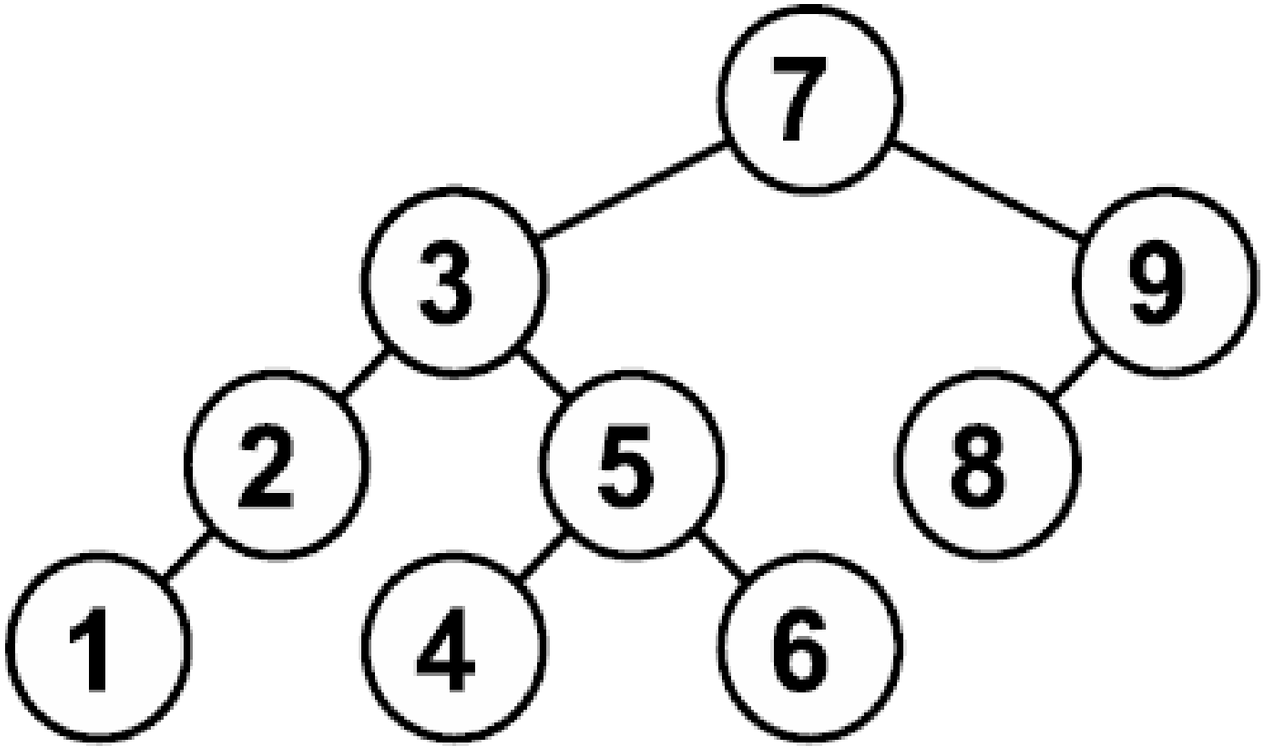
- интерфейс:
  Node * **add** (**Node** * tree, **Data** x);
  Node * **delete** (**Node** * tree, **Data** x);
  Node * **search** (**Node** * tree, **Data** x);
  void **print** (**Node** * tree);

Модель

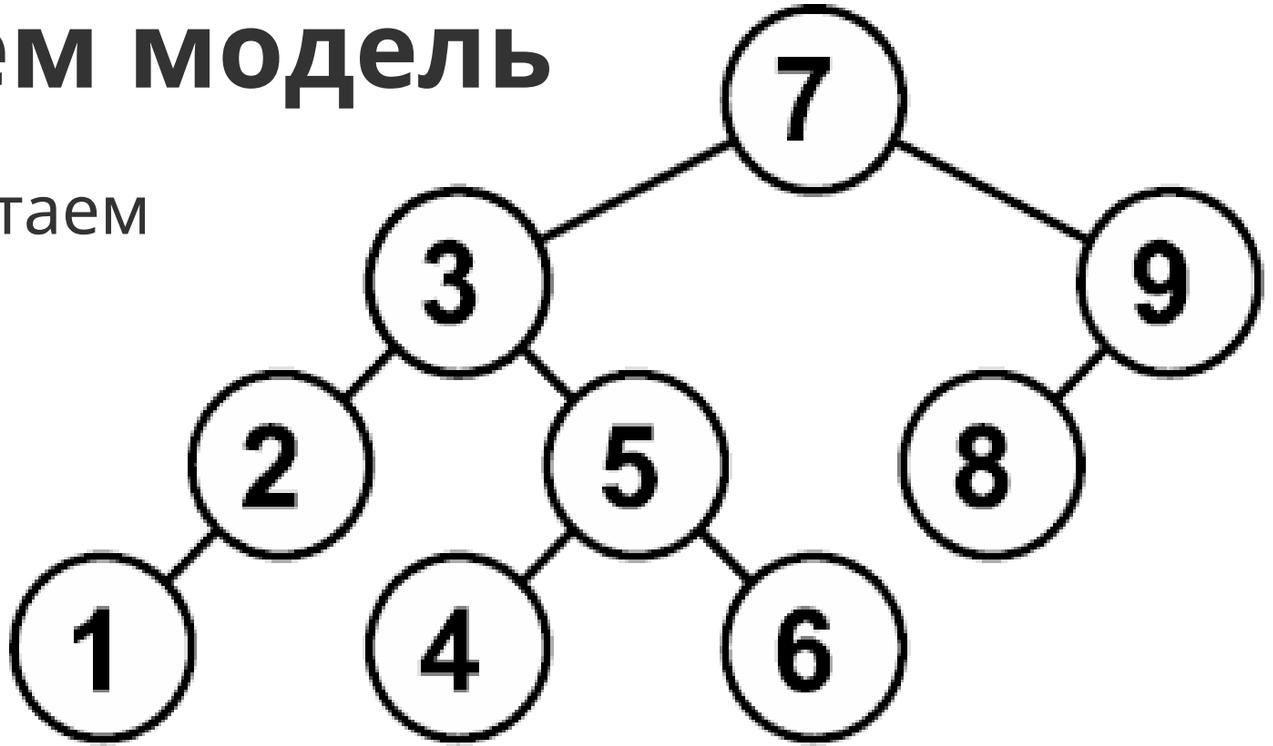
- **Node**

  seven = {7, NULL, NULL},
  nine = {9, NULL, NULL},
  two = {2, NULL, NULL},
  three = {3, NULL, NULL},
  five = {5, NULL, NULL};

- **Node** * tree = NULL;
- tree = & seven;
- seven.left = & three;
- seven.right = & nine;



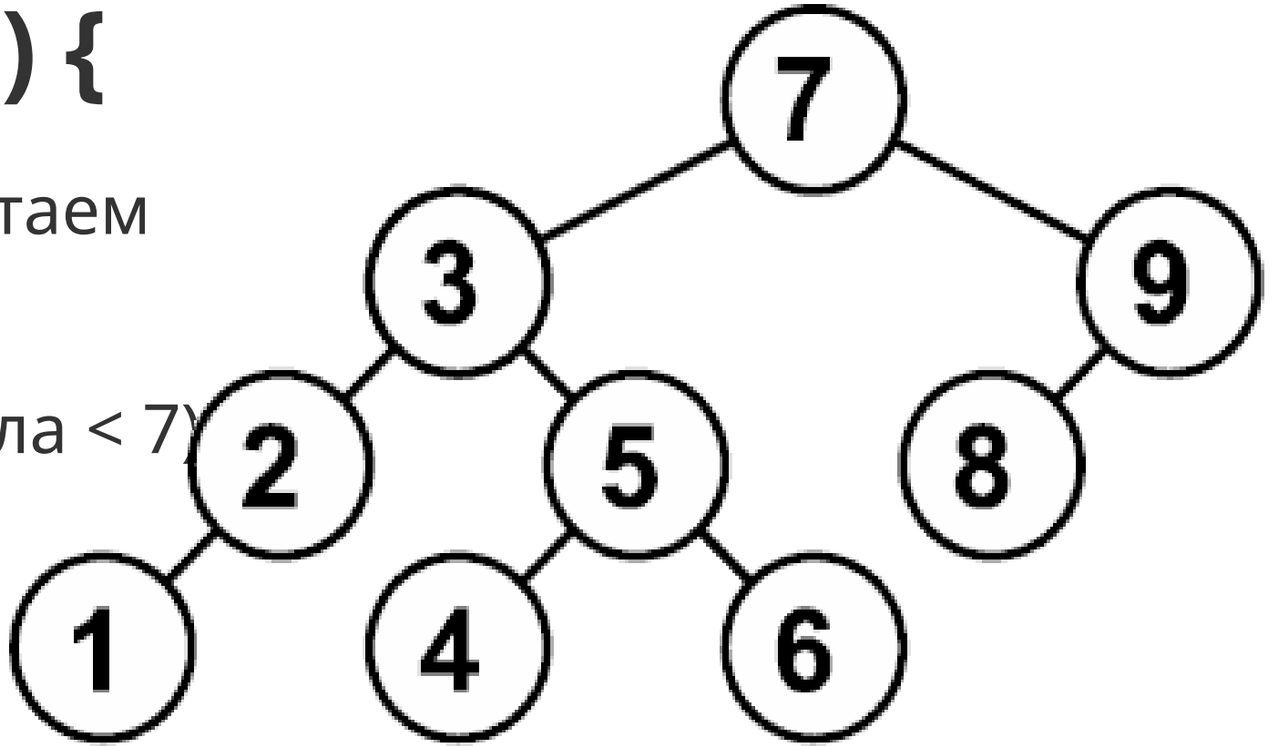
Печать — наращиваем модель



- Пустое дерево — ничего не печатаем
- 1 узел — печатаем его данные
- есть дети — печатаем
  сначала левого ребенка,
  потом правого.
- Дети имеют наследников —
  пусть дети сами с ними разбираются
- Все дети левого ребенка 3 меньше родителя 7
  значит печатаем поддерево левого (числа < 7)
  данные узла 7
  поддерево правого (числа > 7)

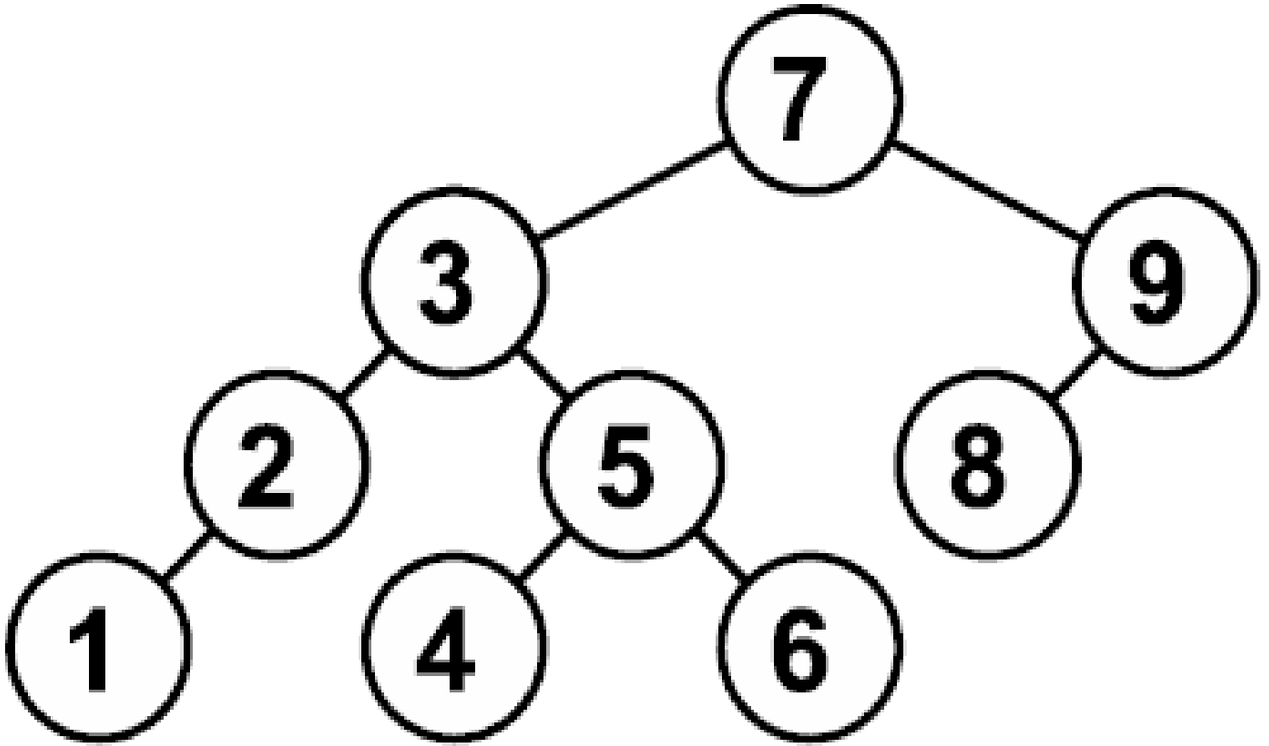
void print (Node * tree) {

- Пустое дерево — ничего не печатаем
  if (tree == NULL) return;
- печатаем поддерево левого (числа < 7)
  print (tree→left);
- данные узла 7
  printf("%d ", tree→data);
- печатаем поддерево правого (числа > 7)
  print (tree→right);
- }



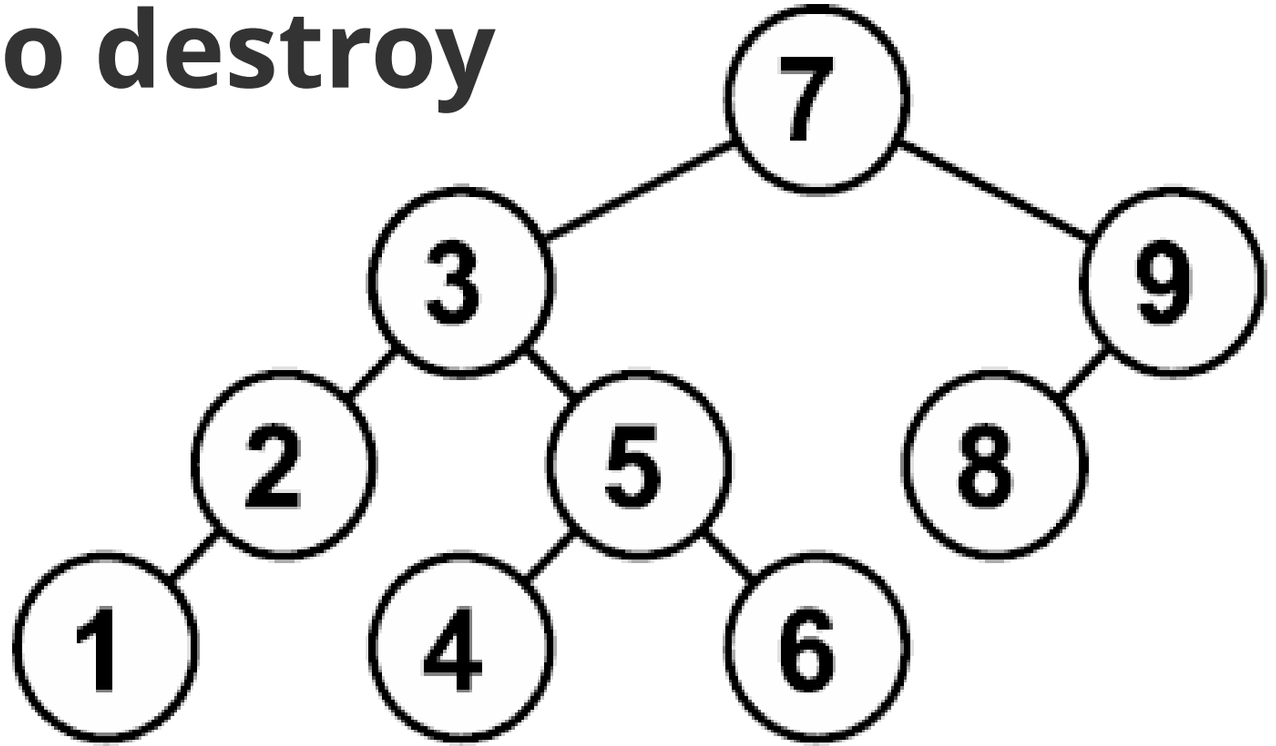
add добавить узел

- Если дерево **пустое**,
  создать первый узел и
  сделать его корнем
  return
-



- Если данные **меньше**, чем в корне,
  добавить в **левое** поддерево
- Если данные **больше**, чем в корне,
  добавить в **правое** поддерево

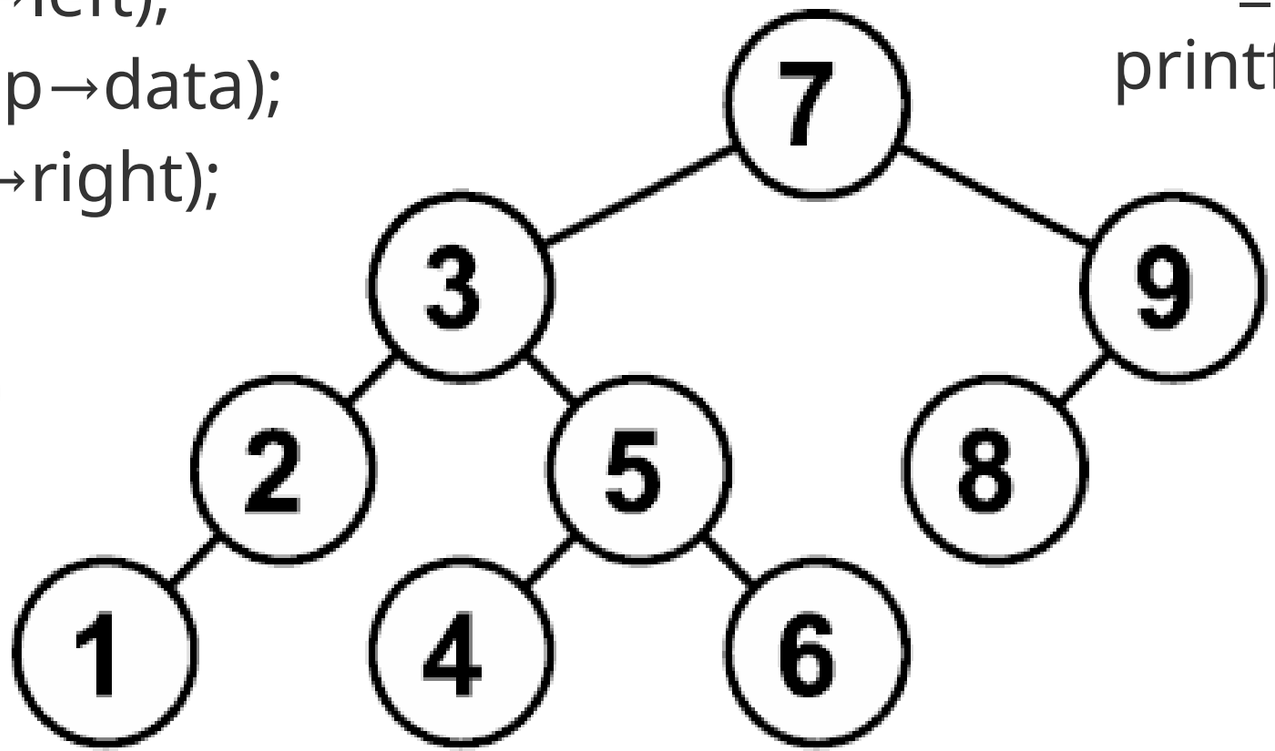
Разрушить все дерево destroy



- Если дерево **пустое**,
  ничего не делать
- разрушить **левое** поддерево
- разрушить **правое** поддерево
- разрушить **себя**
- **tree_destroy**(tree);
  tree_destroy(tree); // Segmentation fault
- tree = **tree_destroy**(tree);
  tree = **tree_destroy**(tree); // free (NULL) - ok

Право или лево?

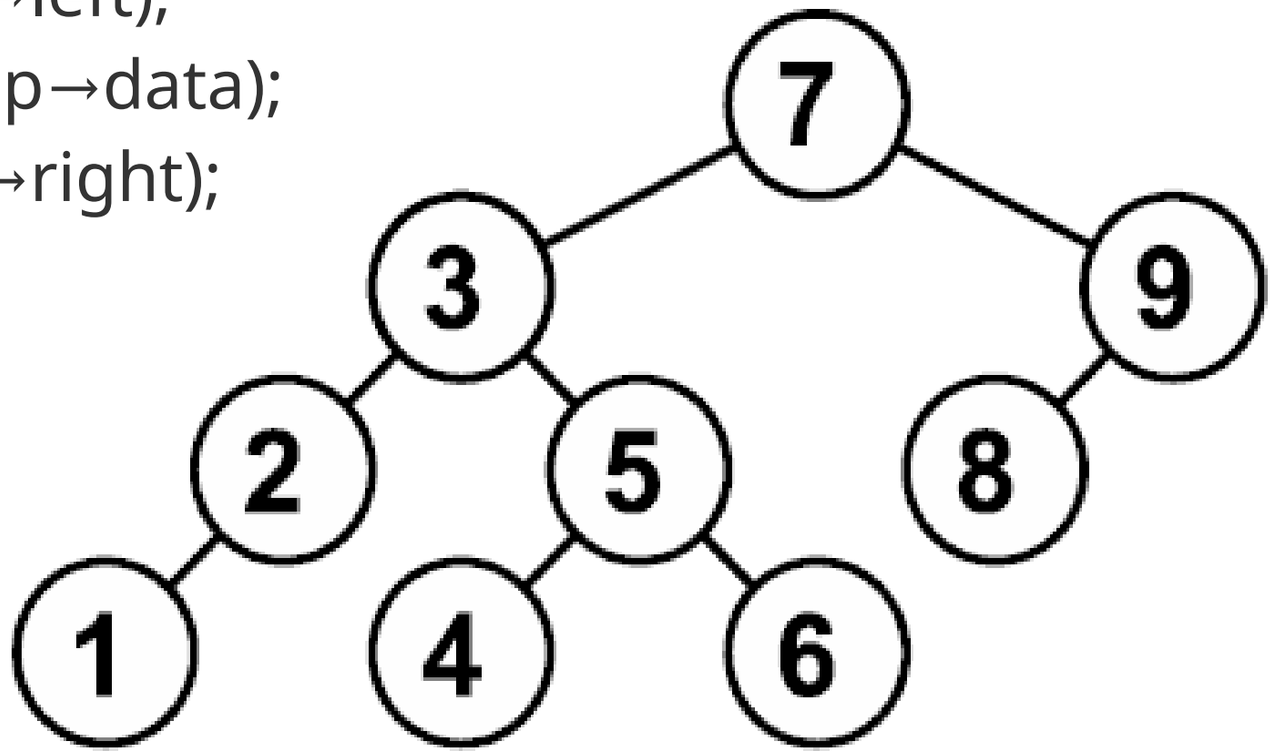
- void tree_print (Node * p) {
  if (tree == NULL)
  return;
  tree_print(p→left);
  printf("%d ", p→data);
  tree_print(p→right);
  }
- 1 2 3 4 5 6 7 8 9



- tree_print(p→right);
  printf("%d ", p→data);
  tree_print(p→left);
- tree_print(p→left);
  tree_print(p→right);
  printf("%d ", p→data);

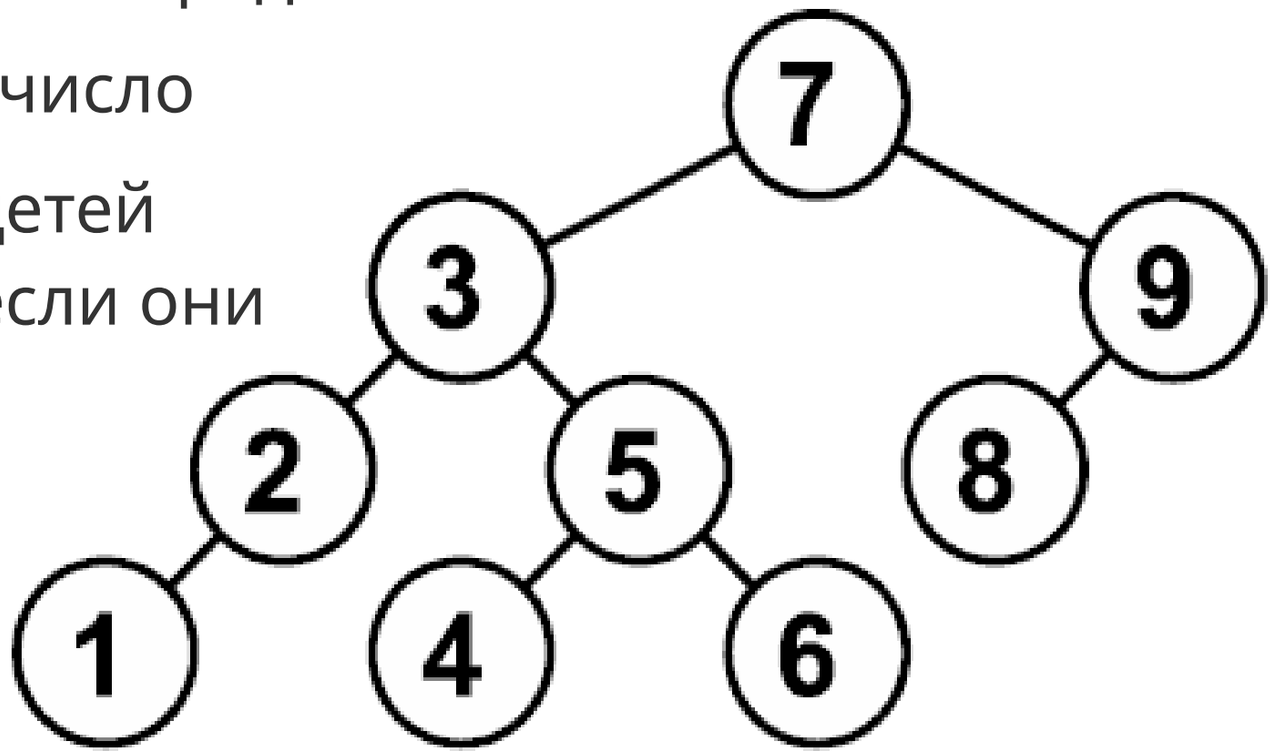
Обход в глубину

- void tree_print (Node * p) {
  if (tree == NULL)
  return;
  tree_print(p→left);
  printf("%d ", p→data);
  tree_print(p→right);
  }
- L7R



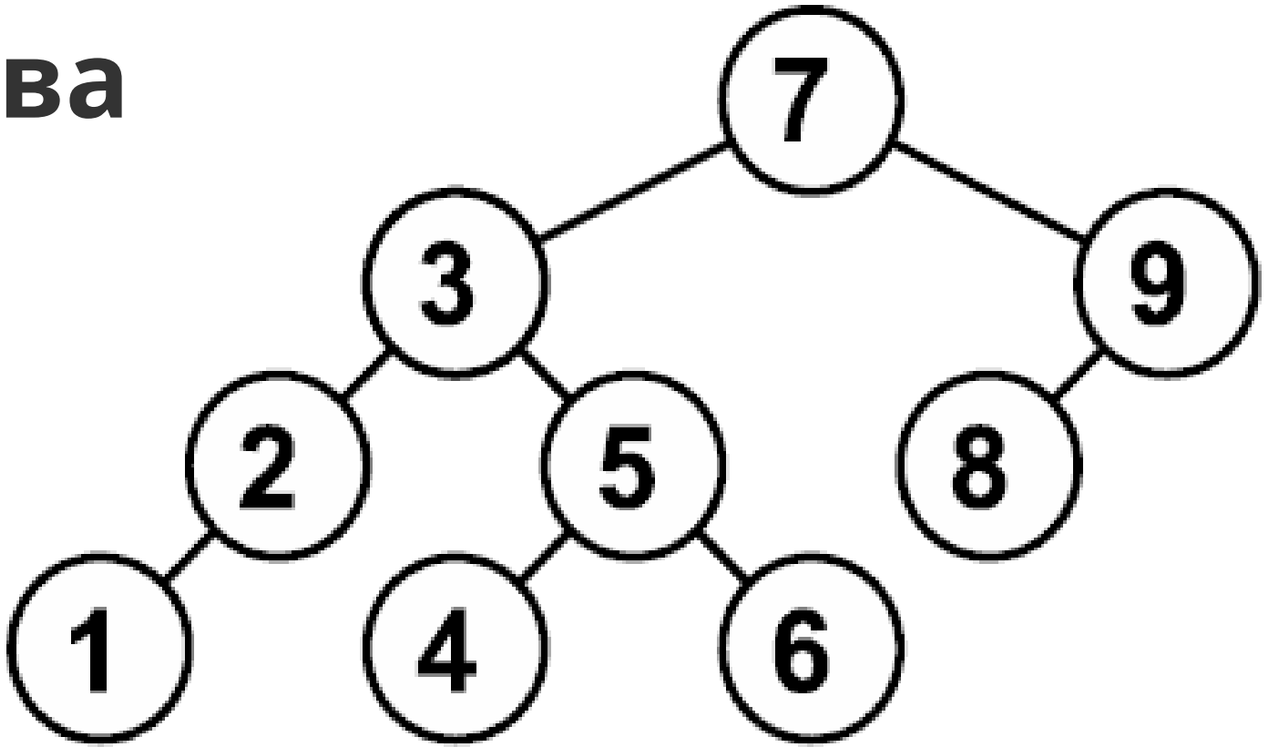
Обход в ширину

- поместим корень в очередь
- пока в очереди есть узлы:
  - взять узел из очереди
  - напечатать число
  - поместить детей в очередь, если они не NULL

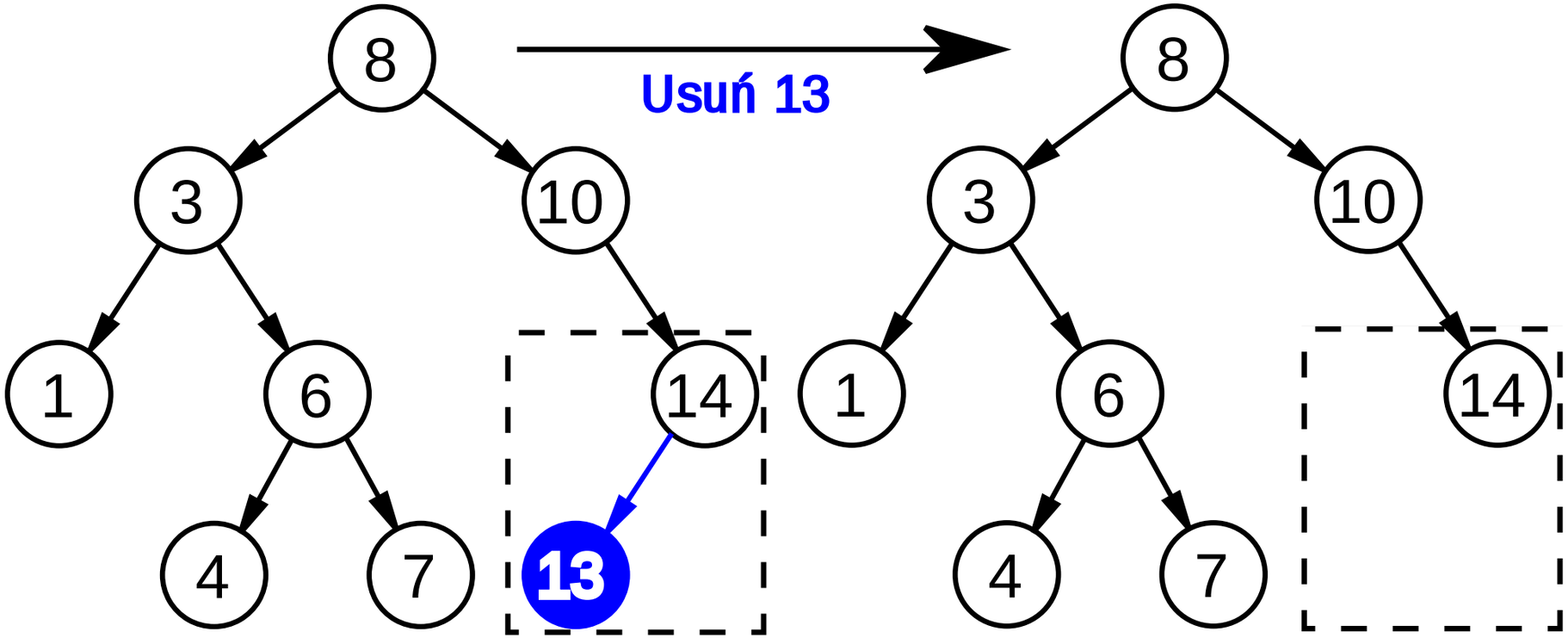


Удалить узел из дерева

- 8
- 9
- 3

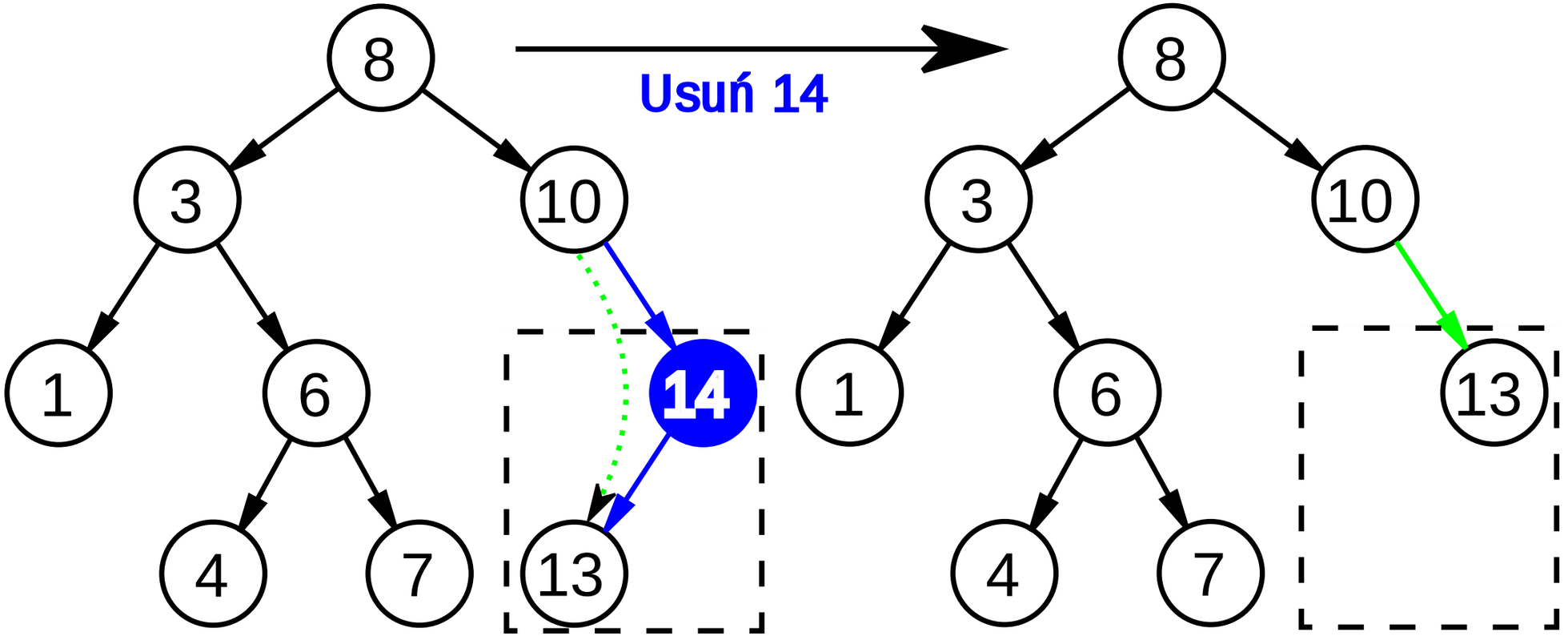


Удалить лист



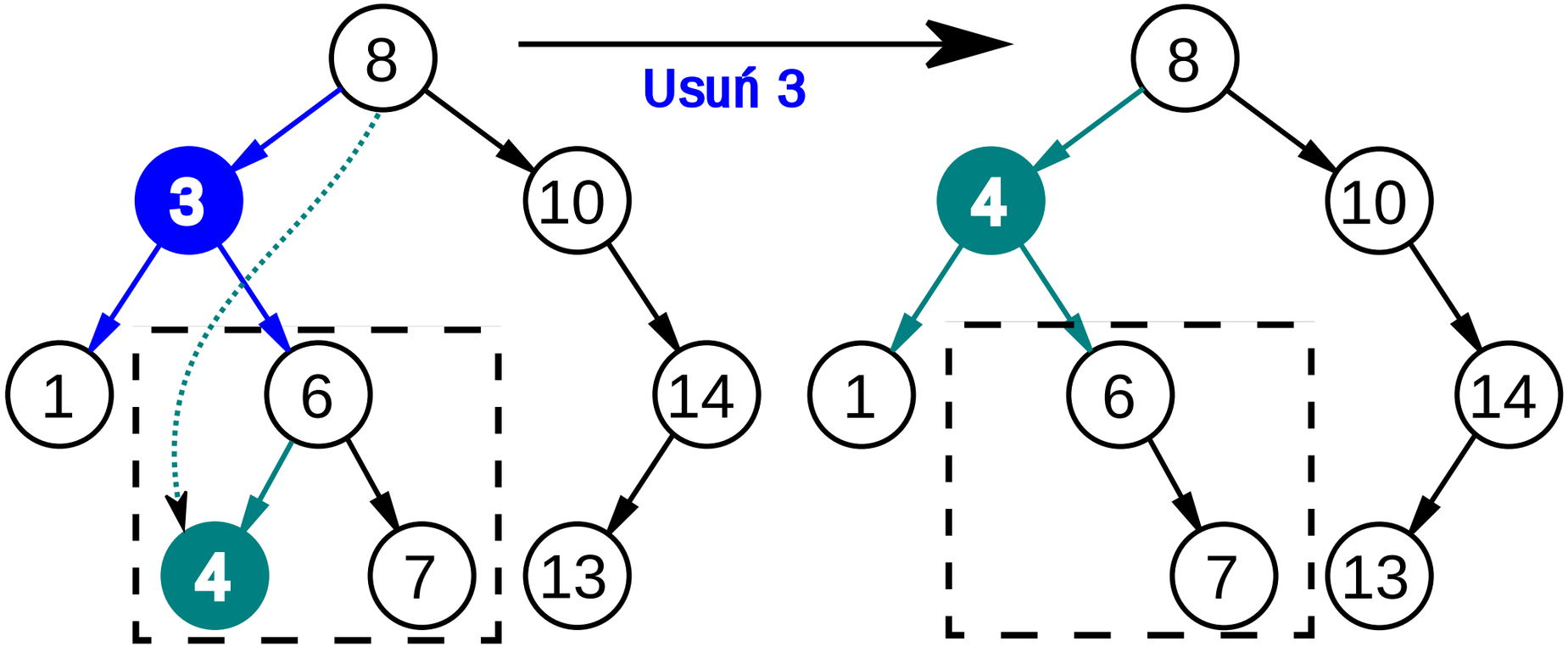
- Booyabazookaderivative work: movax - Binary_search_tree.svg, Public Domain, https://commons.wikimedia.org/w/index.php?curid=11079558

Удалить узел с 1 ребенком



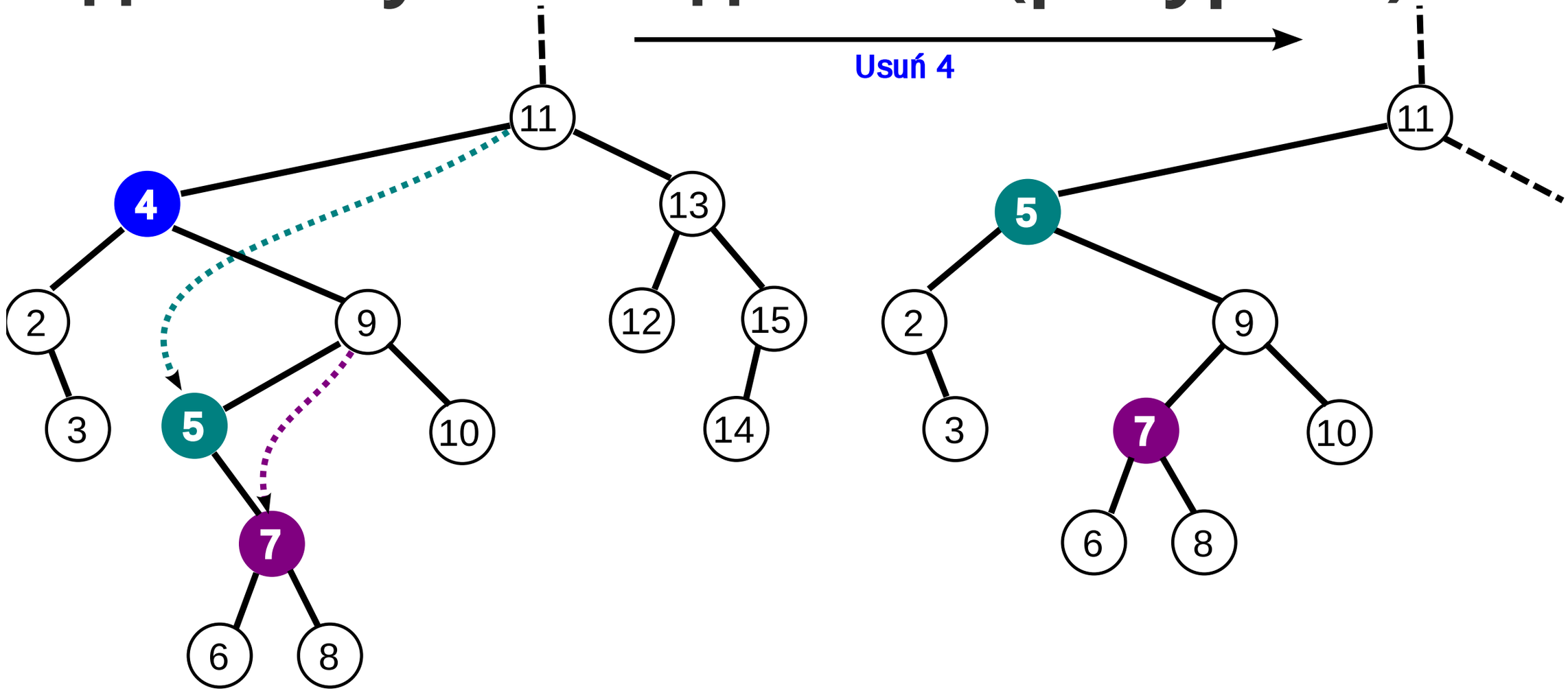
- Booyabazookaderivative work: movax - Binary_search_tree.svg, Public Domain, https://commons.wikimedia.org/w/index.php?curid=11079558

Удалить узел с 2 детьми



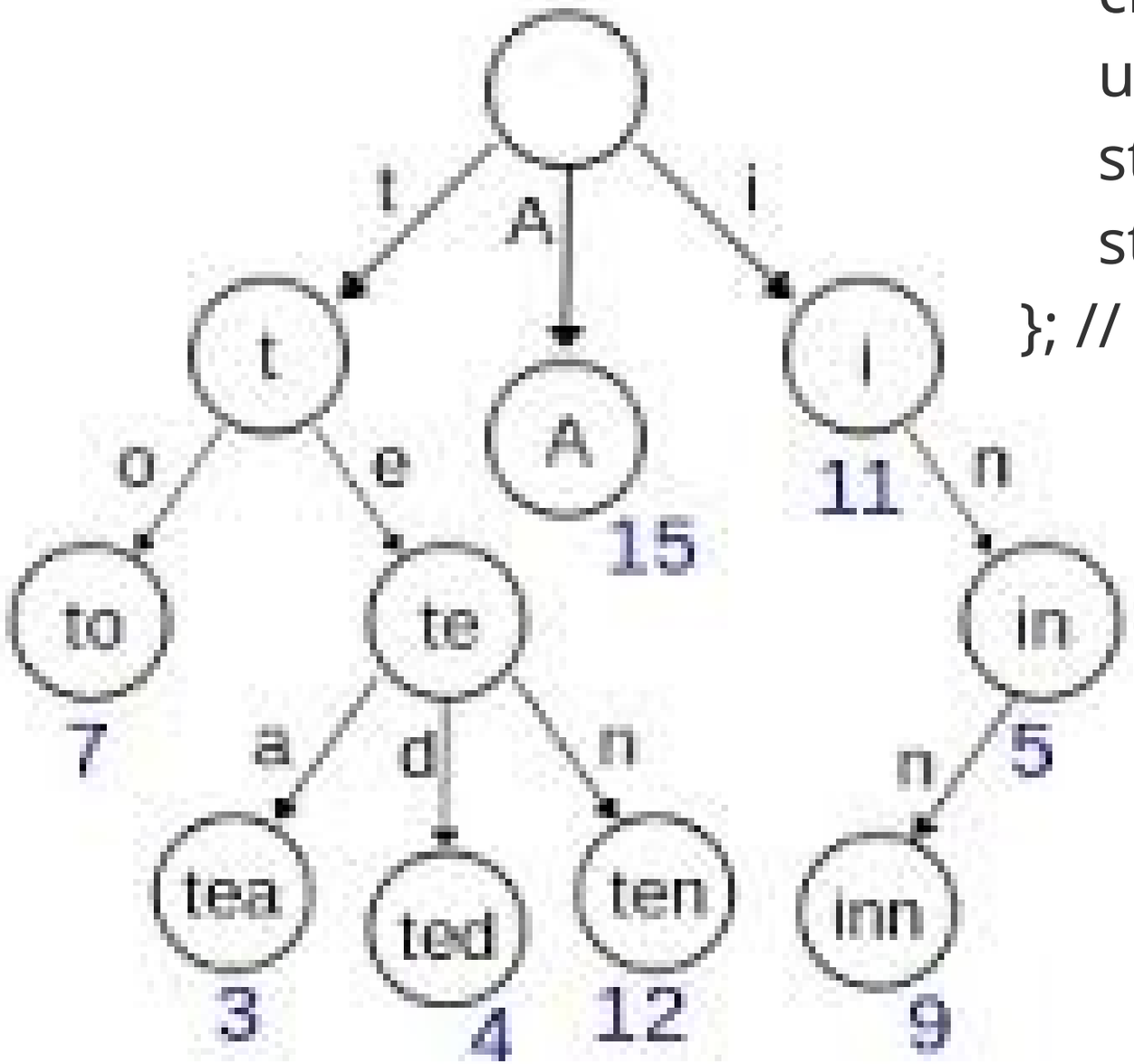
- By Binary_search_tree_delete_14.svg: movaxBinary_search_tree.svg: BooyabazookaBinary_search_tree_delete_13.svg: movaxderivative work: movax - Binary_search_tree.svgBinary_search_tree_delete_14.svg, Public Domain, https://commons.wikimedia.org/w/index.php?curid=11079606

Удалить узел с 2 детьми (рекурсия)



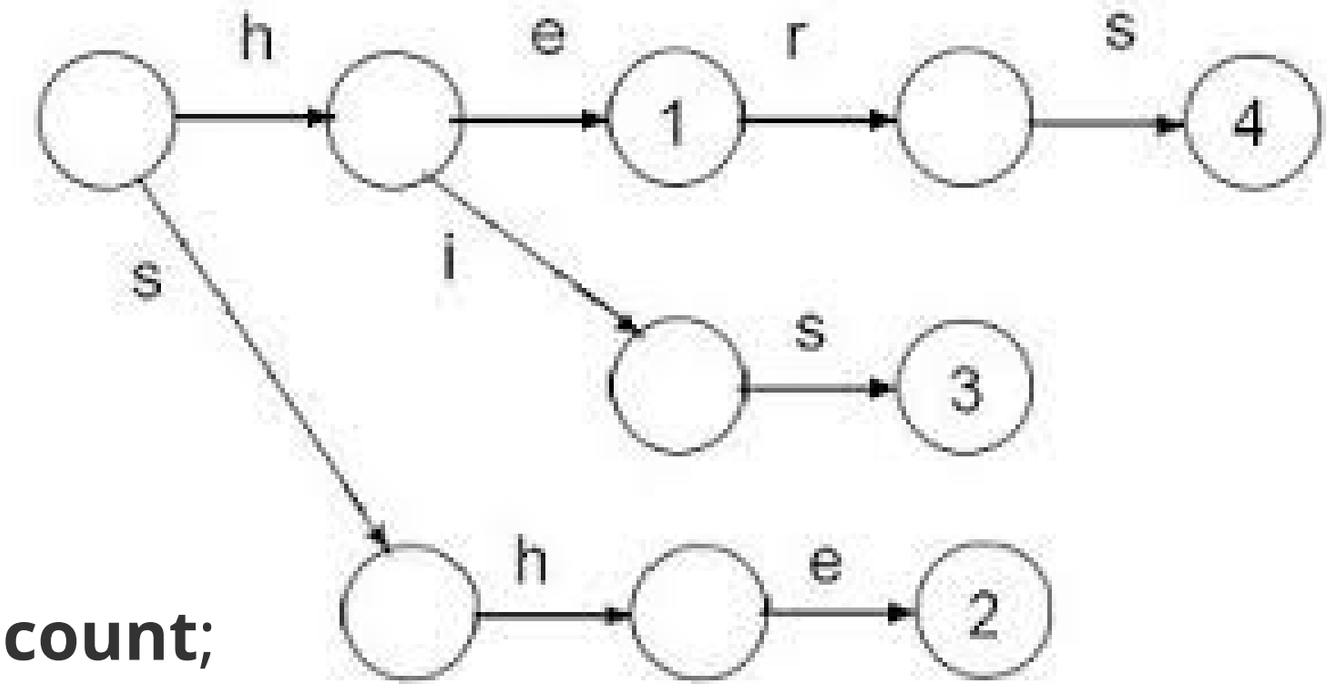
- By Binary_search_tree.svg: Booyabazookaderivative work: movax - Binary_search_tree.svg, Public Domain, https://commons.wikimedia.org/w/index.php?curid=11091911

Частотный словарь



- struct Node {
  char * word;
  unsigned int count;
  struct Node * left;
  struct Node * right;
  }; // много памяти

Лес (частотный словарь)



- struct Letter {
  char **c**;
  unsigned int **count**;
  struct Node * **child**;
  };
  struct Node {
  struct Letter a[26];
  };