Лекция 4

Очередь с приоритетами. Часть 1.

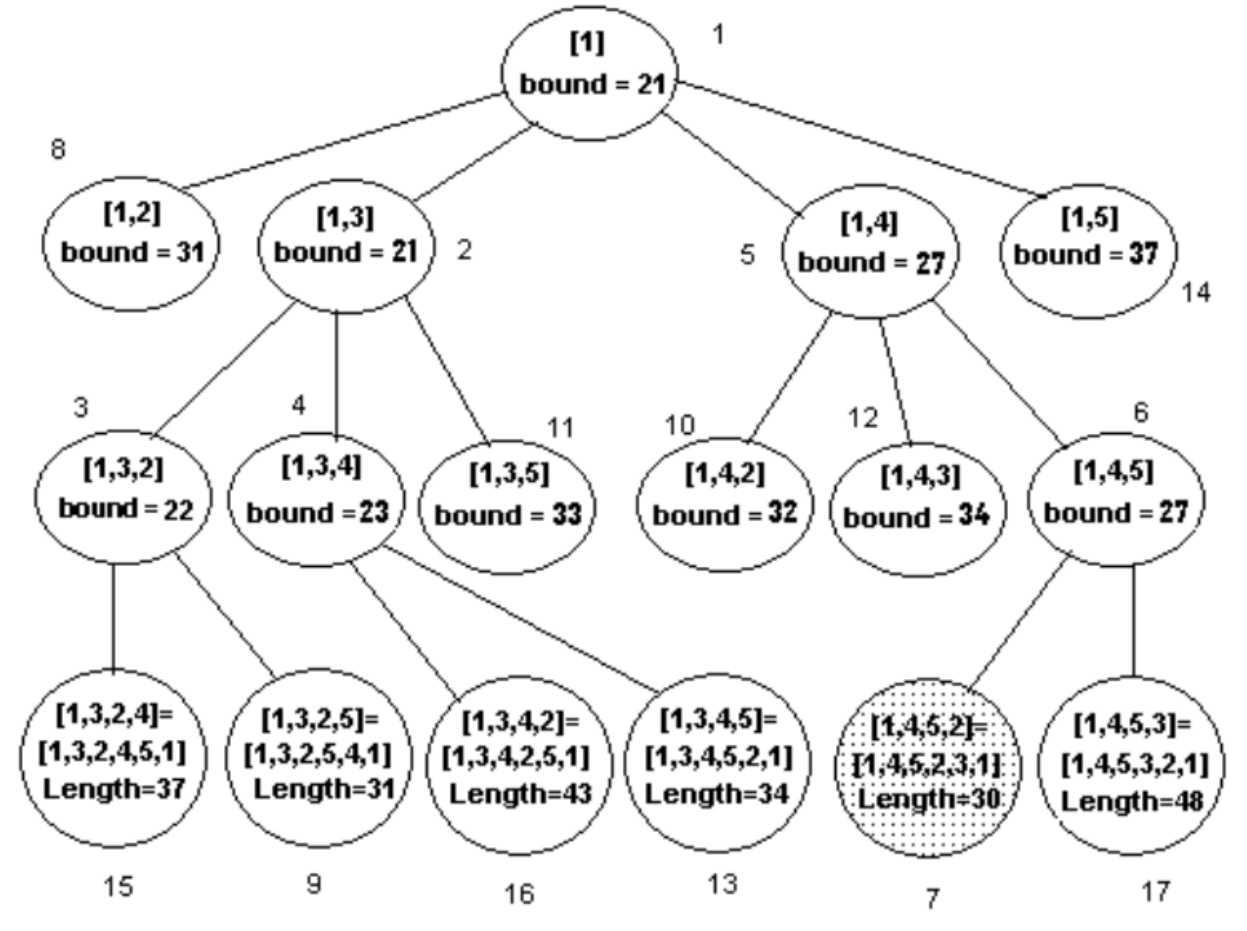
План лекции

Очередь с приоритетами

- Формальное определение
- Применение к МВГ для ЗК
- Варианты реализации
  - Массив
  - Двоичное дерево поиска

МВГ + «сначала наилучшие»

- «Классический» МВГ = обход дерева решений в глубину

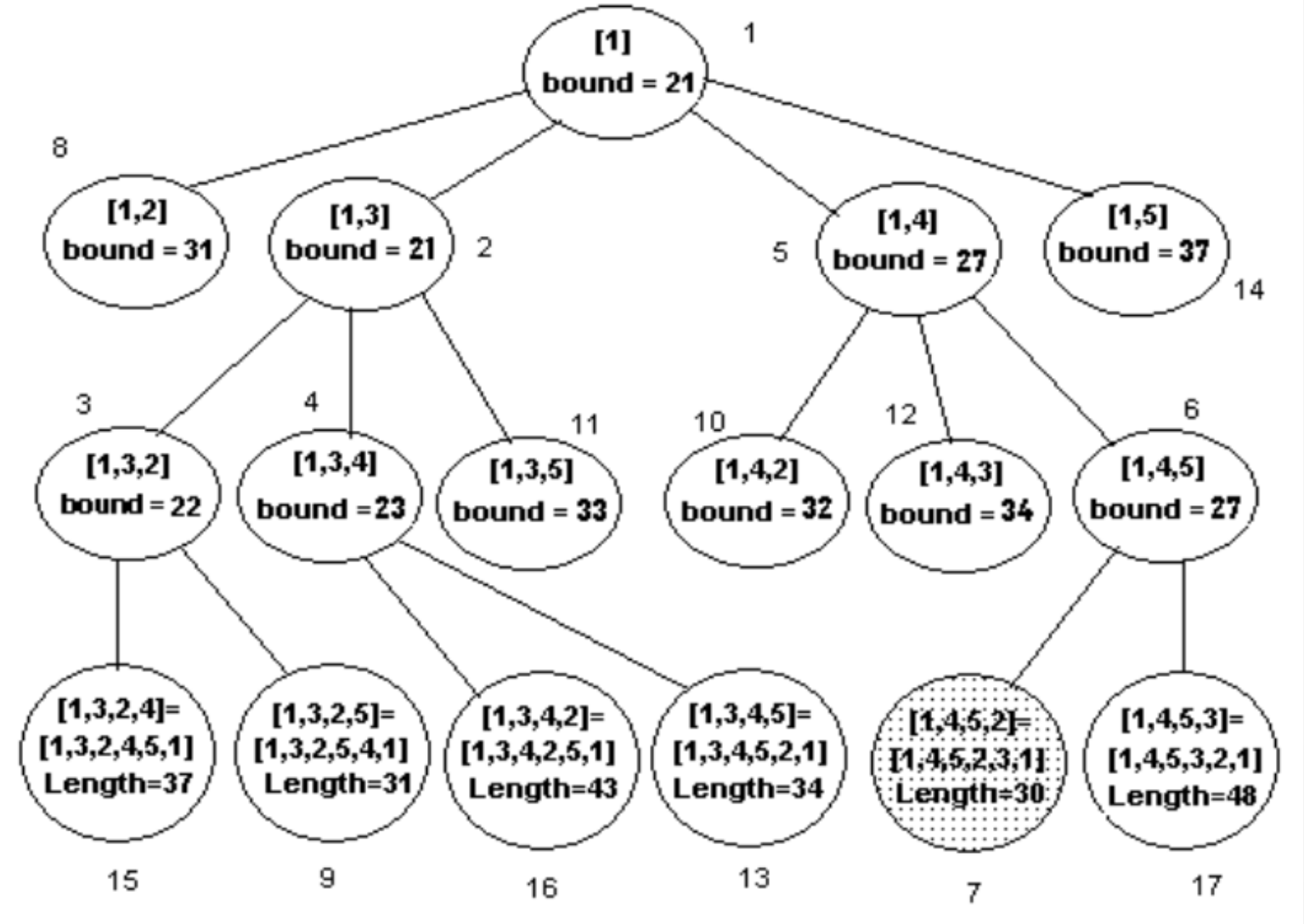


МВГ + «сначала наилучшие»

- Применение эвристики «сначала наилучшие» («best-first»):
  - Каждый «вариант» (подмножество решений) после расчёта границы сохраняем в очередь с приоритетами.
  - Для ветвления берём не текущий (только что оцененный) вариант, а извлекаем из очереди наилучший (с минимальной/максимальной оценкой).

Метод содержит черты обхода дерева решений в ширину.

МВГ + «сначала наилучшие»



Очередь с приоритетами

- Абстрактная структура данных
- Хранит элементы, которым приписаны *ключи* из упорядоченного множества. Ключ называют *приоритетом* элемента.
- Реализует набор операций:
  - Добавить элемент: Insert(x, key)
  - Получить элемент с максимальным (минимальным) ключом: GetMax() / GetMin()
  - Удалить элемент с макс/мин ключом: DelMax() / DelMin()
  - Инициализация: Init()

Очередь с приоритетами

Варианты реализации:

- Массив
- Дерево поиска
  - Двоичное дерево
  - 2-3 дерево
- Куча (пирамида)

Очередь с приоритетами

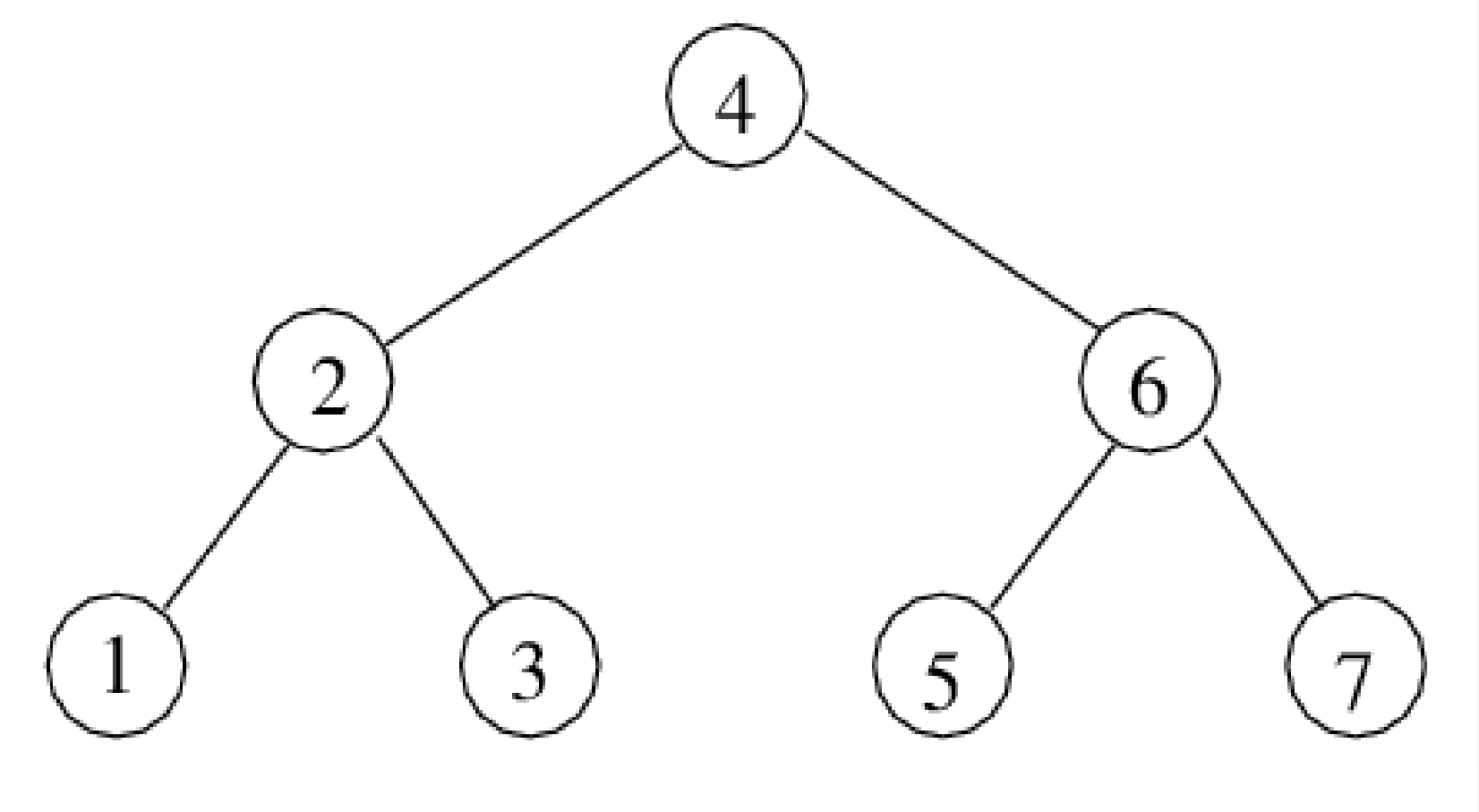
- Обычный массив (динамический список)
  - Неотсортированный
    - Insert: O(1)
    - GetMin: O(*n*)
    - DelMin: O(1)
  - Отсортированный
    - Insert: O(*n*)
    - GetMin: O(1)
    - DelMin: O(1)

Если операций Insert и GetMin примерно поровну, то оба варианта равносильны и средняя стоимость операции O(*n*).

Двоичное дерево поиска

- Двоичное дерево поиска
  - Двоичное дерево
  - Во всех вершинах хранятся элементы
  - Упорядоченность ключей в каждом поддереве:
    - все ключи в левом поддереве — меньше ключа в корне
    - все ключи в правом поддереве — больше ключа в корне

Двоичное дерево



Двоичное дерево

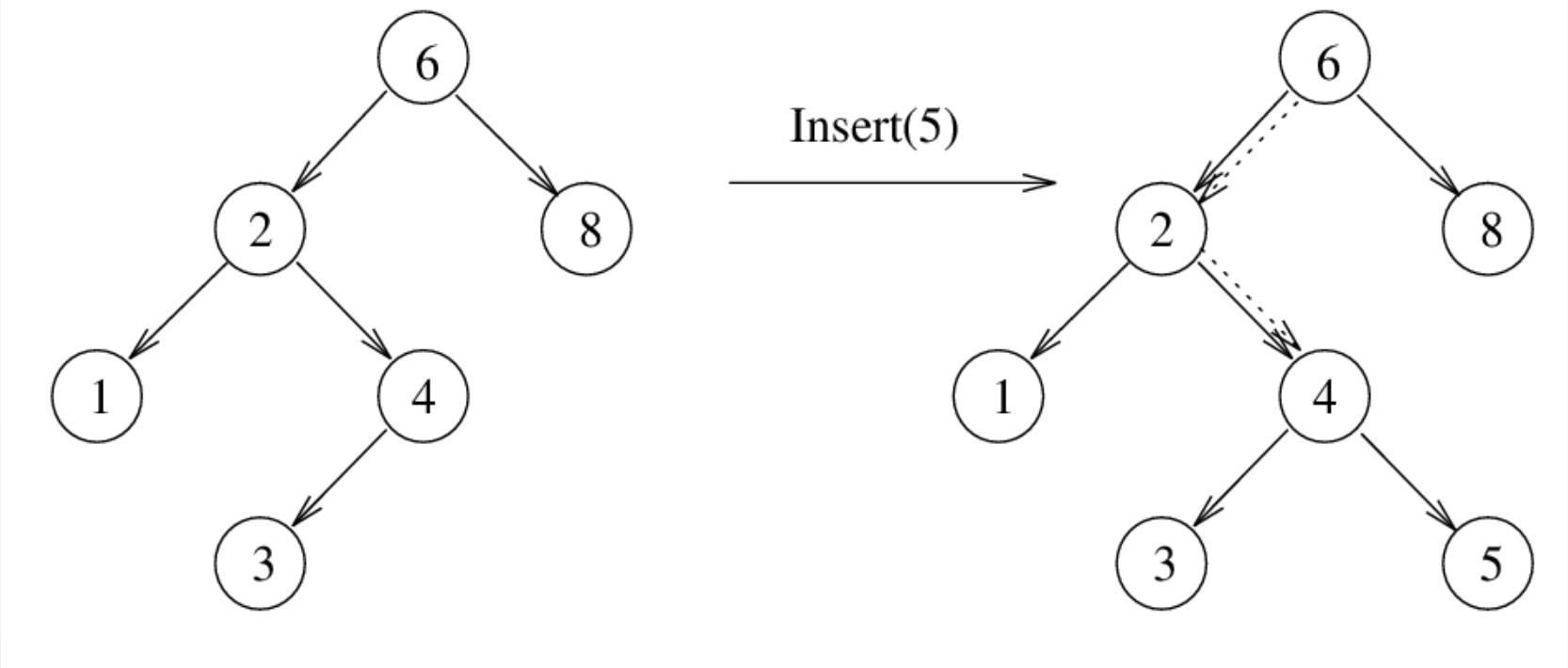
- Find: Поиск элемента с ключом *x*:
  - Сравниваем x с ключом в корне
  - Если совпадает, то нашли
  - Иначе ищем в левом или правом поддереве, в зависимости от результата сравнения
  - Если дошли до листа и не нашло — значит, нет
  - Сложность: O(*h*), где *h* — высота дерева.
- GetMin: проходим в самый левый узел.
  - Сложность: O(*h*), где *h* — высота дерева.
- GetMax: проходим в самый правый узел.
  - Сложность: O(*h*), где *h* — высота дерева.

Двоичное дерево

- Вставка: Insert(*x*, key):
  - Выполнить поиск
  - Создать новый лист в том месте, где остановились, установить в нём значение (x, key)
  - Сложность: O(*h*), где *h* — высота дерева.

Двоичное дерево

- Вставка: Insert(*x*, 5):

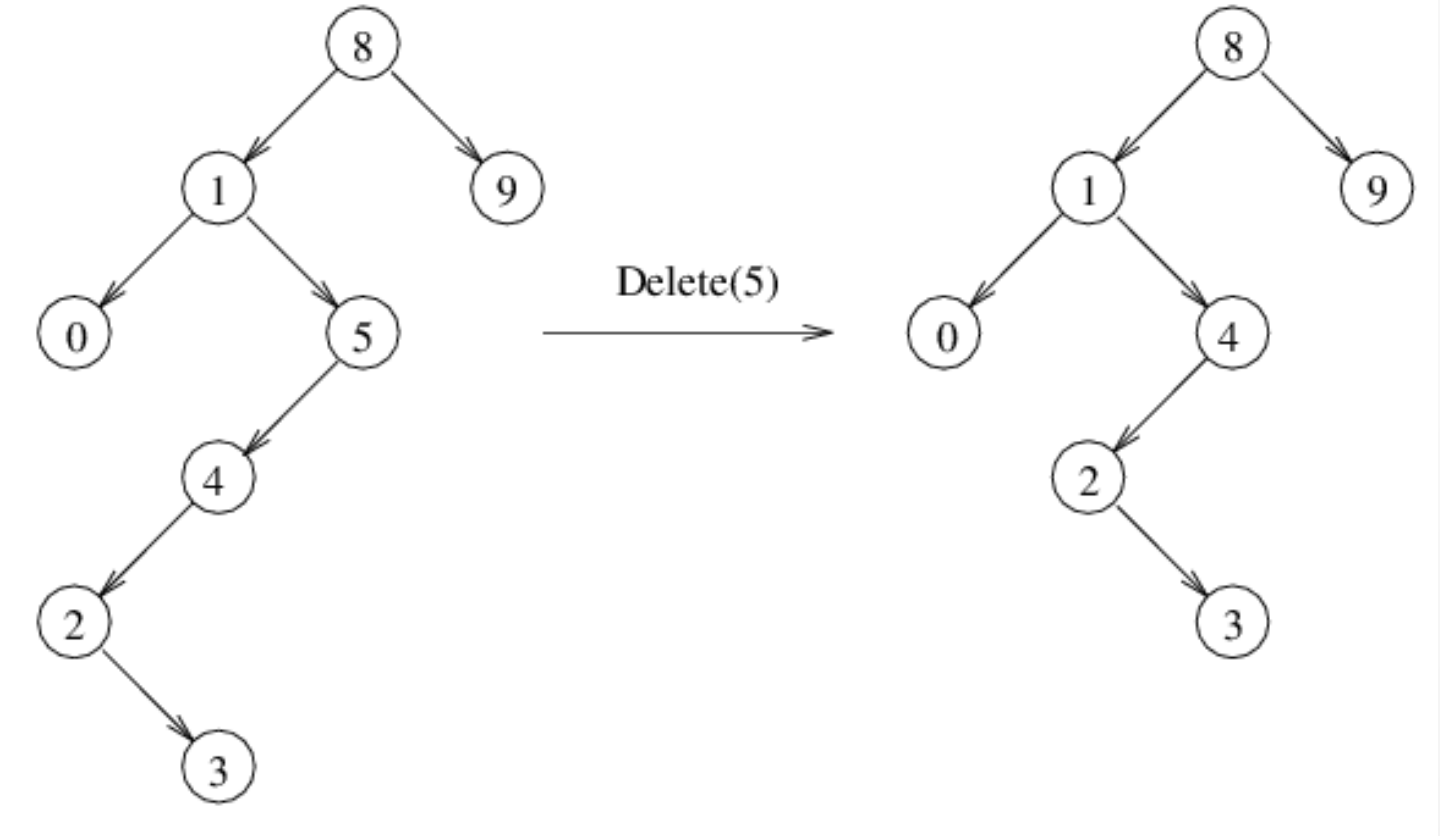


Двоичное дерево

- Удаление:
  - DelMin, DelMax : удалить самый левый/правый лист
  - Удаление произвольного элемента x
    - Произвольный лист — просто удаляем
    - Внутренняя вершина с 1 потомком: заменяем вершину имеющимся потомком
    - Внутренняя вершина с 2 потомками:
      - находим в правом поддереве минимальный элемент y
      - удаляем вершину с y
      - y переносим на место удаляемой вершины x

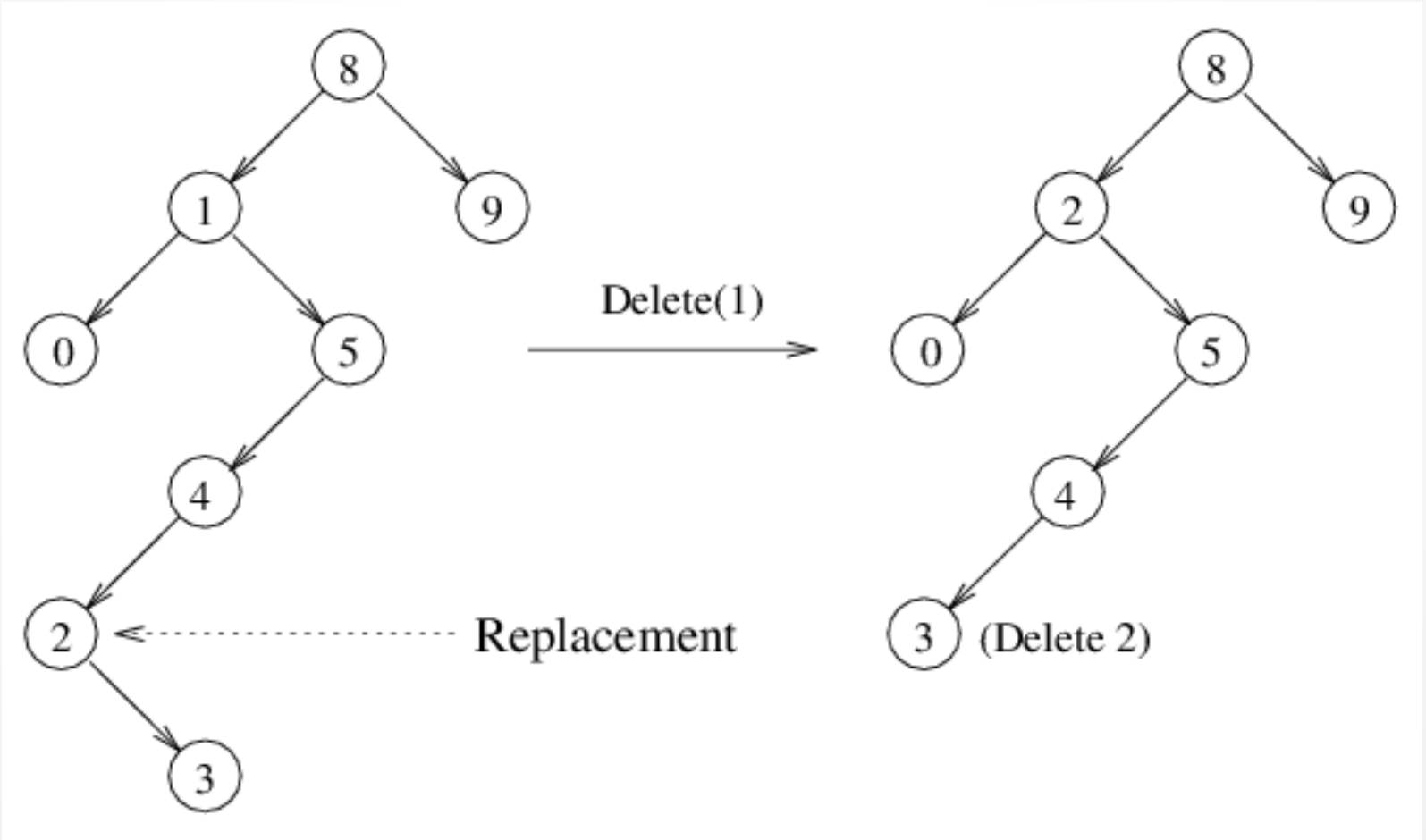
Двоичное дерево

Удаление вершины с 1 потомком



Двоичное дерево

Удаление вершины с 2 потомками

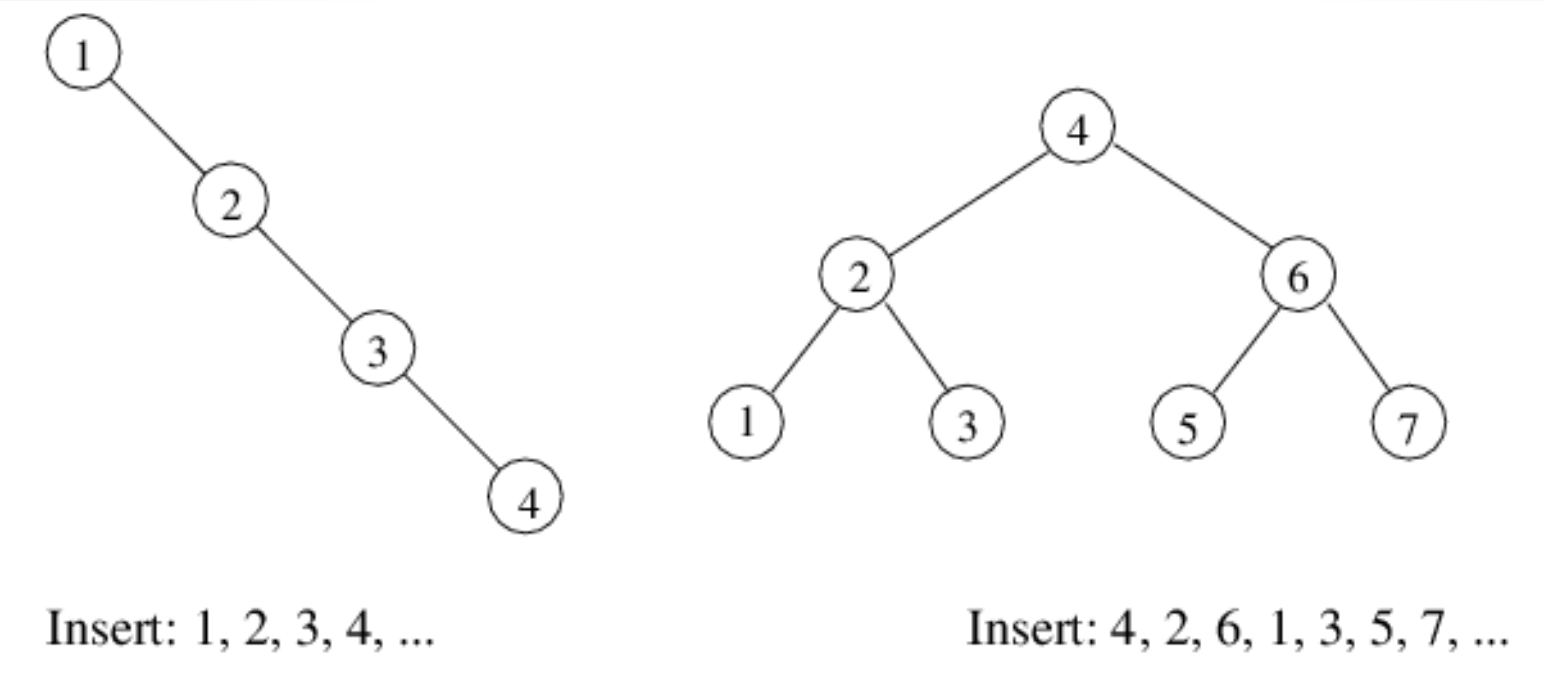


Двоичное дерево

Анализ эффективности

- GetMin, GetMax, Find, Insert, Delete: O(*h*)

Высота *h*: от O(lg *n*) до O(*n*). Среднее: O(lg *n*)



Двоичное дерево

Вывод: нужна структура, для которой будет гарантированная оценка $h \sim O(\log n)$.

- Сбалансированные деревья:
  - хранить в каждой вершине информацию о мощностях поддеревьев
  - при необходимости — перестраивать дерево для сохранения сбалансированности
  - AVL, красно-чёрные деревья
  - 2-3 деревья