Сбалансированные деревья поиска.

Проблемы известных структур данных и пути из решения:

1) нельзя использовать операции сравнения из-за неэффективности поиска -> использовать порядок на элементах

2) во всех струкутурах данных в чистом виде, кроме двоичного дерева поиска, есть хоть одна операция, выполняемая за O(n)

3) в простом двоичном дереве поиска все операции выполняются за O(h), значит надо уменьшить максимальную высоту дерева, для этого сбалансировать его

Структуры данных, которые эффективно выполняют:

1) поиск произвольного элемента
2) вставку произвольного элемента
3) удаление найденного элемента
4) удаление произвольного элемента (1)+(3)

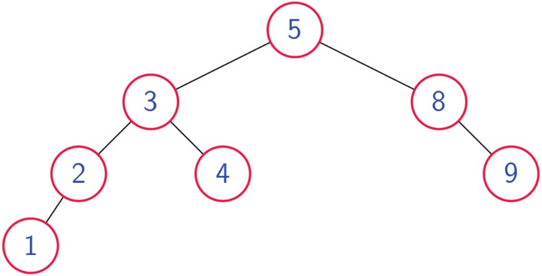
Сбалансированность достигается несколькими способами:

Способ 1. Поддержание инвариантов сбалансированности

- Пример: Высота левого поддерева отличается от высоты правого поддерева не больше, чем на единицу

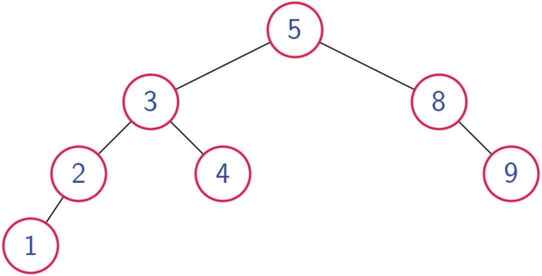
Способ 1. Поддержание инвариантов сбалансированности

- Пример: Высота левого поддерева отличается от высоты правого поддерева не больше, чем на единицу
- Реализация: АВЛ-дерево
  - АВЛ: Адельсон-Вельский и Ландис



Способ 1. Поддержание инвариантов сбалансированности

- Пример: Высота левого поддерева отличается от высоты правого поддерева не больше, чем на единицу
- Реализация: АВЛ-дерево
  - АВЛ: Адельсон-Вельский и Ландис



- Высота дерева: $h = O(\log n)$

Способ 1. Поддержание инвариантов сбалансированности

- Пример: Разделим узлы на два типа: «черные» и «красные»

Способ 1. Поддержание инвариантов сбалансированности

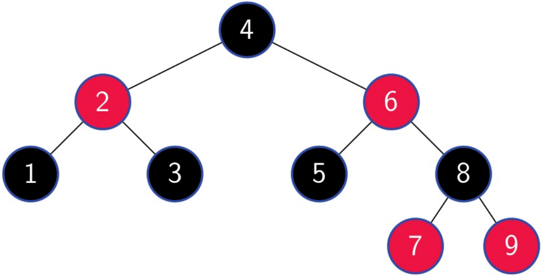
- Пример: Разделим узлы на два типа: «черные» и «красные»
  - Дети красных узлов обязательно черные

Способ 1. Поддержание инвариантов сбалансированности

- Пример: Разделим узлы на два типа: «черные» и «красные»
  - Дети красных узлов обязательно черные
  - Число черных узлов на пути от корня к любому узлу, не имеющему хотя бы одного ребенка, одинаково

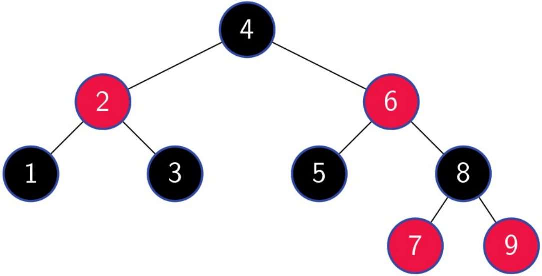
Способ 1. Поддержание инвариантов сбалансированности

- Пример: Разделим узлы на два типа: «черные» и «красные»
  - Дети красных узлов обязательно черные
  - Число черных узлов на пути от корня к любому узлу, не имеющему хотя бы одного ребенка, одинаково
- Реализация: красно-черное дерево



Способ 1. Поддержание инвариантов сбалансированности

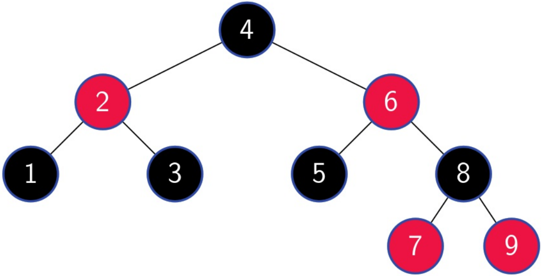
- Пример: Разделим узлы на два типа: «черные» и «красные»
  - Дети красных узлов обязательно черные
  - Число черных узлов на пути от корня к любому узлу, не имеющему хотя бы одного ребенка, одинаково
- Реализация: красно-черное дерево



- Высота дерева: $h = O(\log n)$

Способ 1. Поддержание инвариантов сбалансированности

- Пример: Разделим узлы на два типа: «черные» и «красные»
  - Дети красных узлов обязательно черные
  - Число черных узлов на пути от корня к любому узлу, не имеющему хотя бы одного ребенка, одинаково
- Реализация: красно-черное дерево



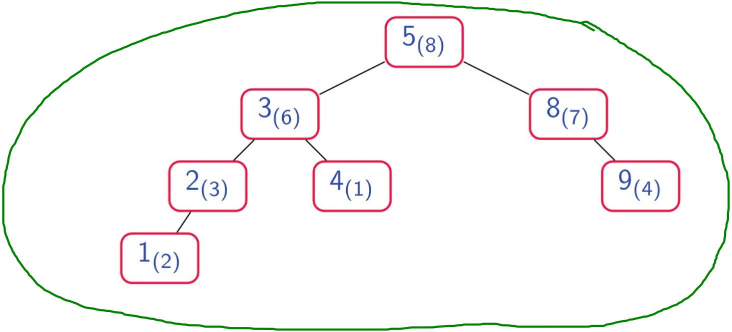
- Высота дерева: $h = O(\log n)$
- Используется в стандартных библиотеках языков программирования (`std::set`, `java.util.TreeSet`, . . .)

Способ 2. Строить дерево частично случайно

- Пример: Использовать дополнительный ключ, генерируемый случайно, и поддерживать свойство кучи

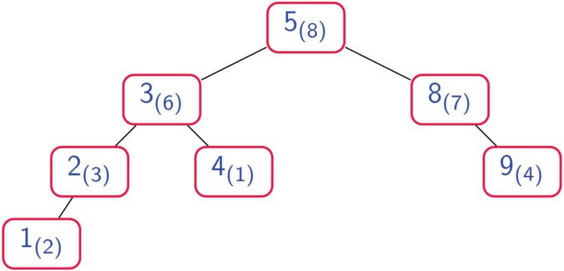
Способ 2. Строить дерево частично случайно

- Пример: Использовать дополнительный ключ, генерируемый случайно, и поддерживать свойство кучи
- Реализация: декартово дерево
  - по основному ключу — дерево поиска
  - по дополнительному ключу (в скобках) — куча



Способ 2. Строить дерево частично случайно

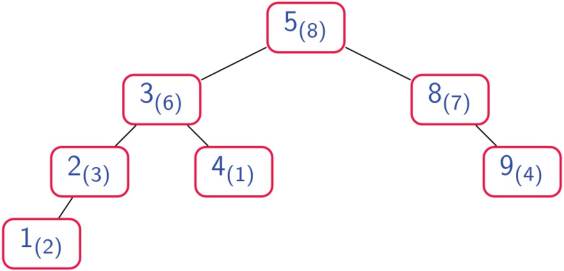
- Пример: Использовать дополнительный ключ, генерируемый случайно, и поддерживать свойство кучи
- Реализация: декартово дерево
  - по основному ключу — дерево поиска
  - по дополнительному ключу (в скобках) — куча



- Математическое ожидание высоты дерева: $h = O(\log n)$

Способ 2. Строить дерево частично случайно

- Пример: Использовать дополнительный ключ, генерируемый случайно, и поддерживать свойство кучи
- Реализация: декартово дерево
  - по основному ключу — дерево поиска
  - по дополнительному ключу (в скобках) — куча



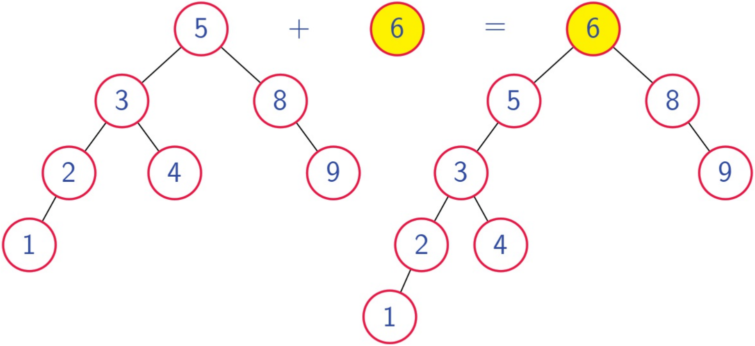
- Математическое ожидание высоты дерева: $h = O(\log n)$

Способ 3. Использовать эвристики при перестроении

- Пример: Перемещать вершину в корень при вставке, поиске и удалении

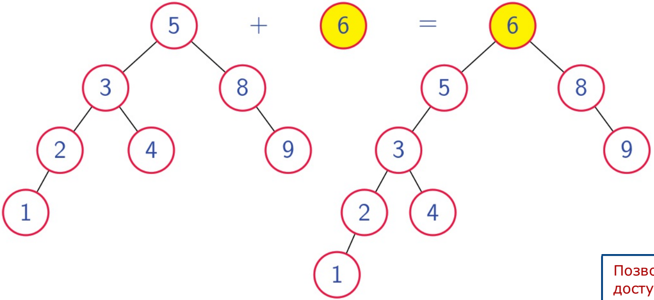
Способ 3. Использовать эвристики при перестроении

- Пример: Перемещать вершину в корень при вставке, поиске и удалении
- Реализация: splay-дерево



Способ 3. Использовать эвристики при перестроении

- Пример: Перемещать вершину в корень при вставке, поиске и удалении
- Реализация: splay-дерево



Позволяет быстрее получать доступ к объектам, обрабатывавшимся недавно

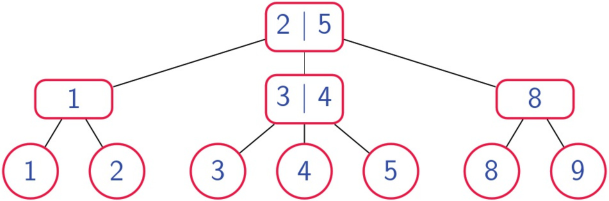
- Сложность операции вставки/поиска/удаления: $O(\log n)$ в среднем за $O(n)$ операций

Способ 4. Использовать недвоичные деревья

- Пример: позволить каждому внутреннему узлу иметь два или три потомка

Способ 4. Использовать недвоичные деревья

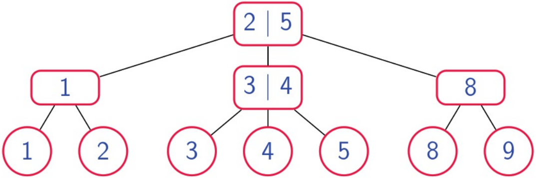
- Пример: позволить каждому внутреннему узлу иметь два или три потомка
- Реализация: 2-3-дерево (вариант B-дерева)
  - Элементы хранятся только в листьях
  - Все листья имеют одинаковую высоту
  - Внутренние узлы содержат максимальные ключи левого и (если есть) среднего поддерева



Способ 4. Использовать недвоичные деревья

- Пример: позволить каждому внутреннему узлу иметь два или три потомка
- Реализация: 2-3-дерево (вариант B-дерева)
  - Элементы хранятся только в листьях
  - Все листья имеют одинаковую высоту
  - Внутренние узлы содержат максимальные ключи левого и (если есть) среднего поддерева

Применяется в базах данных и дисковых хранилищах



- Высота дерева: $O(\log n)$