Лекция 5

Очередь с приоритетами. Часть 2.

План лекции

Очередь с приоритетами

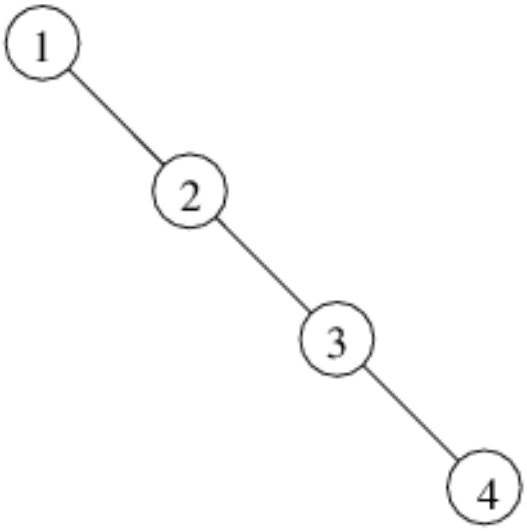
- Варианты реализации
  - 2-3 дерево поиска
  - Куча (пирамида)
- Задание 5

Двоичное дерево

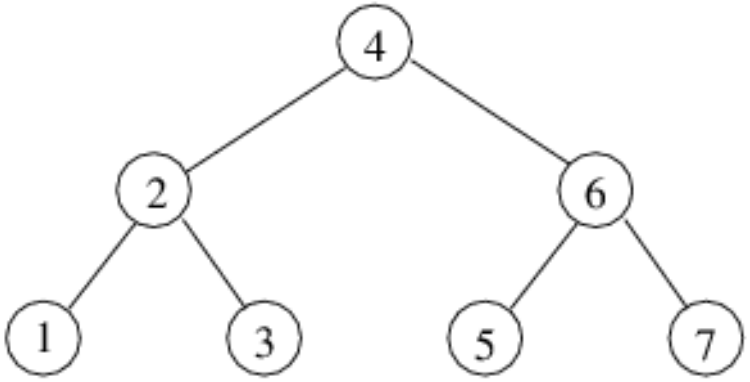
Анализ эффективности

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

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



Insert: 1, 2, 3, 4, ...



Insert: 4, 2, 6, 1, 3, 5, 7, ...

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

2-3 дерево

Свойства

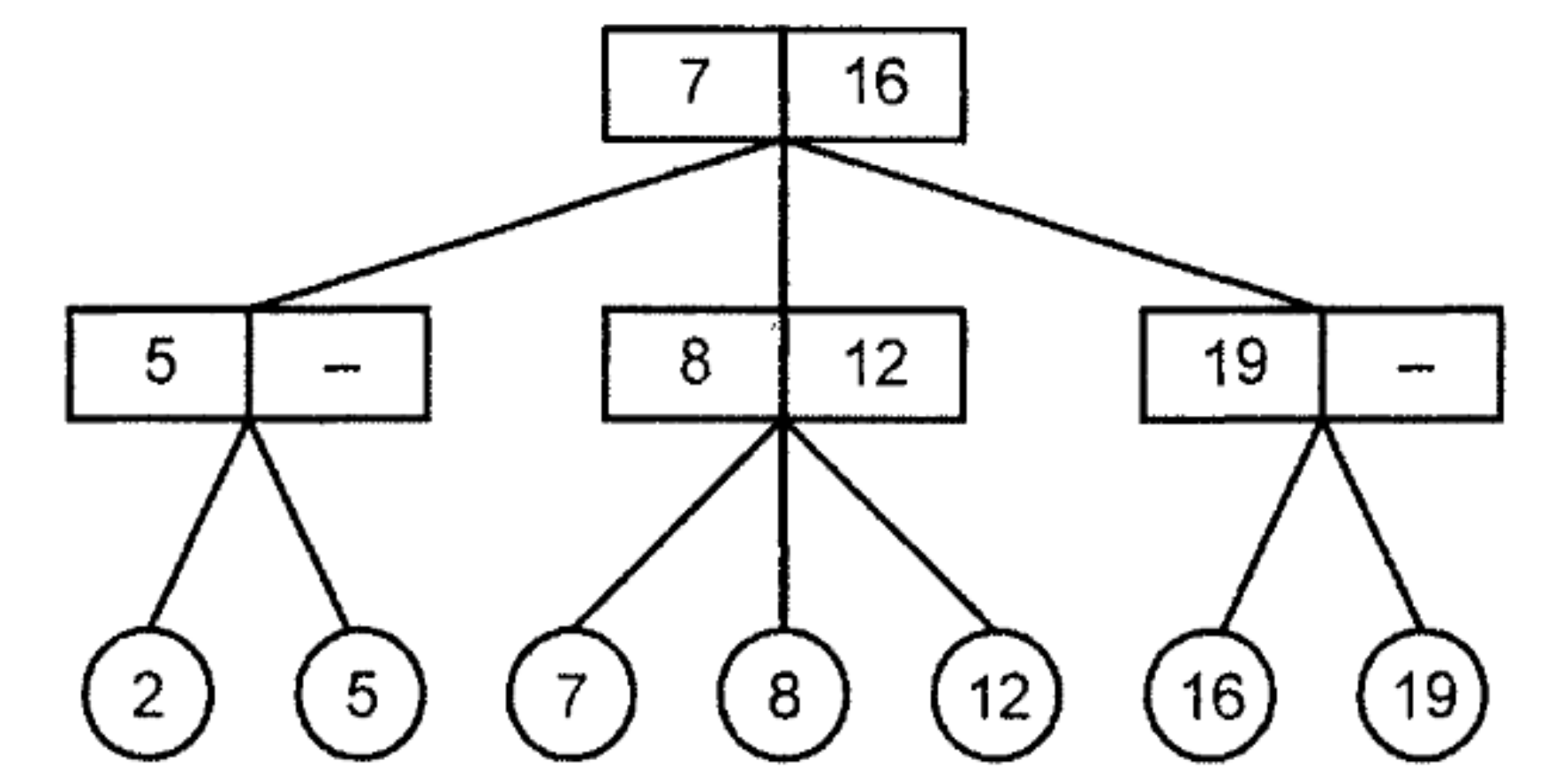
- Каждая внутренняя вершина имеет 2 или 3 потомков
- Пути от корня к каждому листу имеют одинаковую длину

Структура

- Хранимые элементы размещаются в листьях
- Каждая внутренняя вершина хранит 2 метки:
  - наименьший ключ в среднем поддереве
  - наименьший ключ правого поддерева (если всего 2 поддерева, то NULL).

Глубина дерева: $O(\log n)$

2-3 дерево



2-3 дерево

Операции

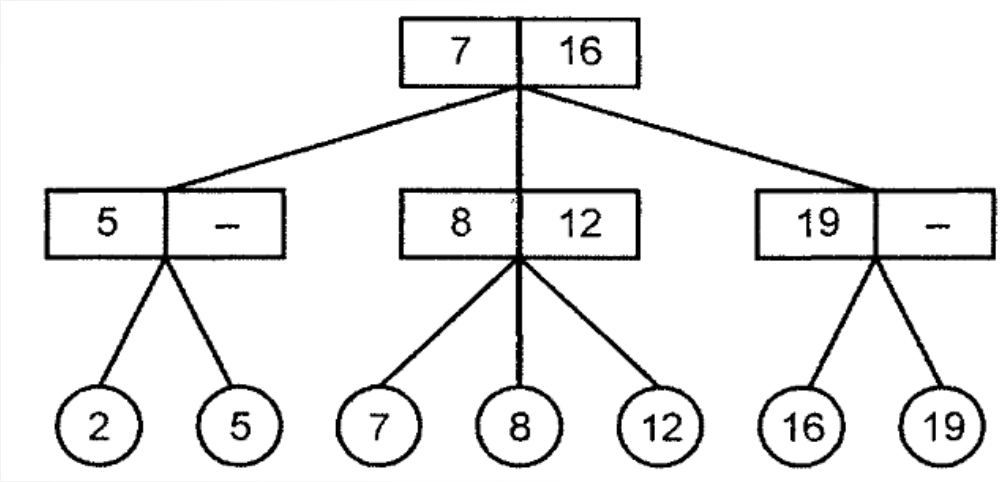
- Поиск ключа *x*: начинаем с корня. При анализе внутренней вершины с метками (y,z)
  - если x < y, то ищем в левом поддереве
  - если z = NULL, то ищем во втором поддереве
  - иначе:
    - если x < z, то ищем в среднем поддереве,
    - иначе ищем в правом поддереве
- GetMin / GetMax: выбираем значение из самого левого / правого листа

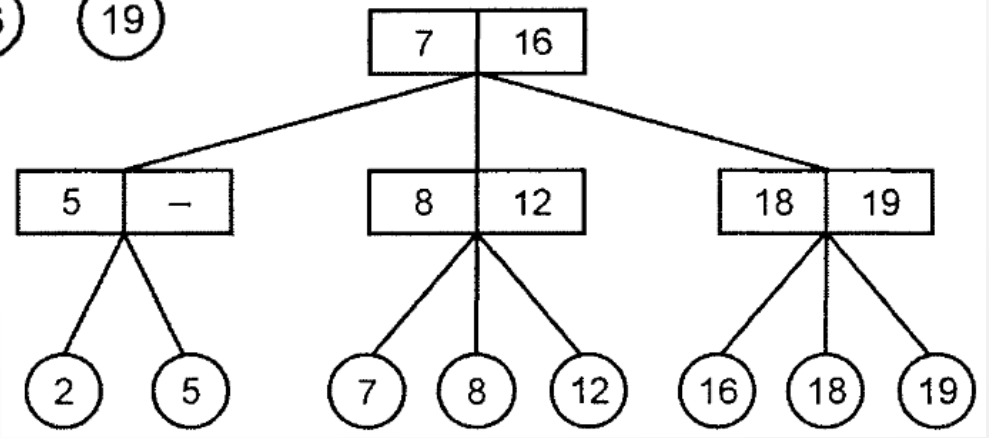
2-3 дерево

- Вставка элемента с ключом *x* :
  - Выполняем поиск, пока не дойдём до предпоследнего уровня
  - Пусть дошли до вершины *v*
  - Если у вершины *v* две дочерние вершины, то добавляем новую в нужном порядке и обновляем пометки в вершине *v*.

2-3 дерево

Вставка элемента с ключом 18



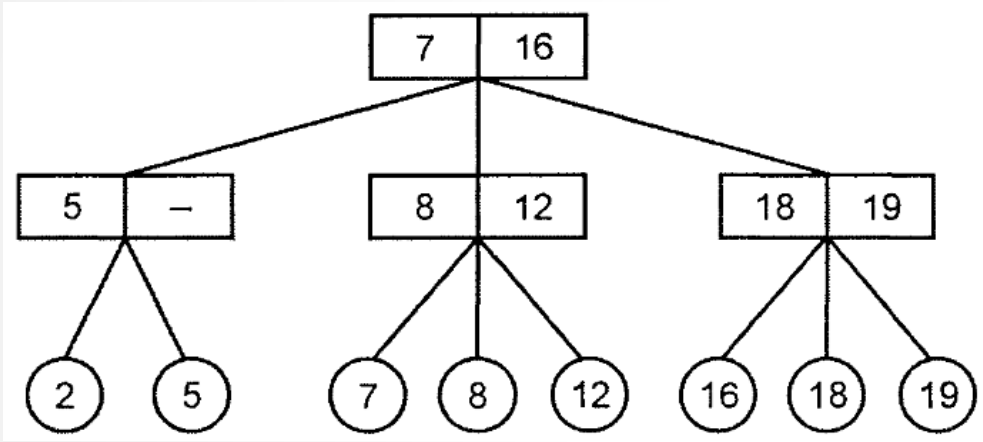


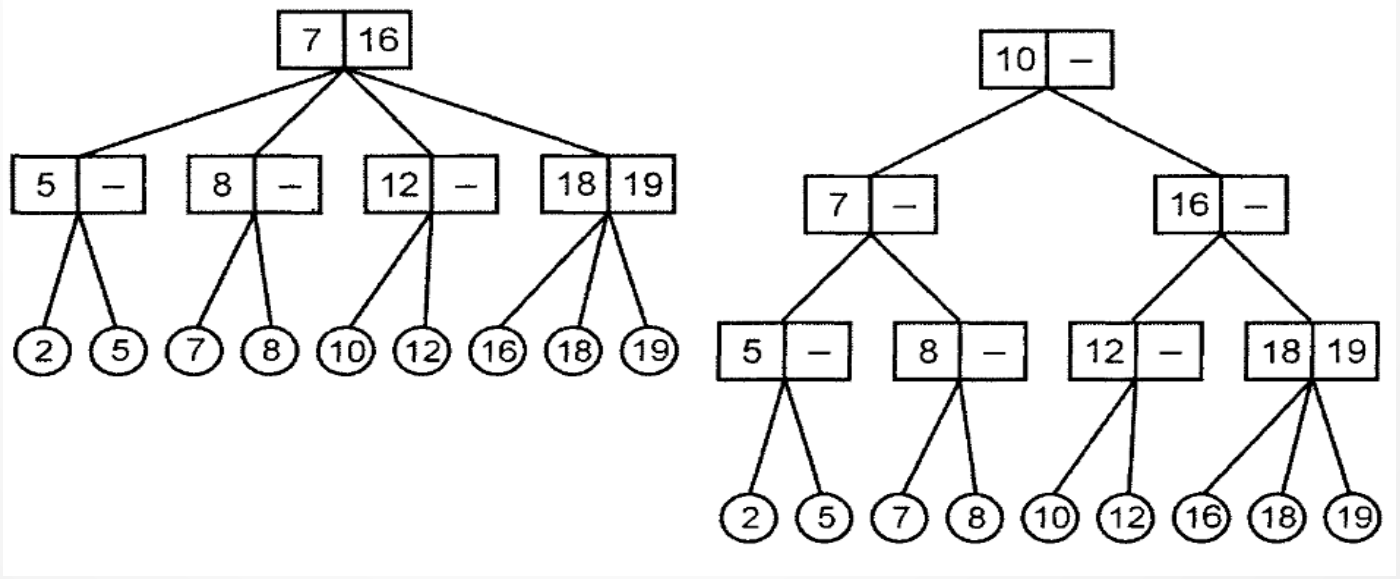
2-3 дерево

- Если у вершины *v* уже 3 дочерние вершины, то расщепляем v на две вершины v' и v'':
  - к v' относим дочерние вершины с 2 меньшими значениями
  - к v'' относим дочерние вершины с 2 бОльшими значениями
  - если у родителя v (теперь — v' и v'') оказалось 4 дочерние вершины, то рекурсивно выполняем аналогичные действия к вышестоящим вершинам
  - при применении такого преобразования к корню увеличивается общая глубина дерева

2-3 дерево

Вставка элемента с ключом 10





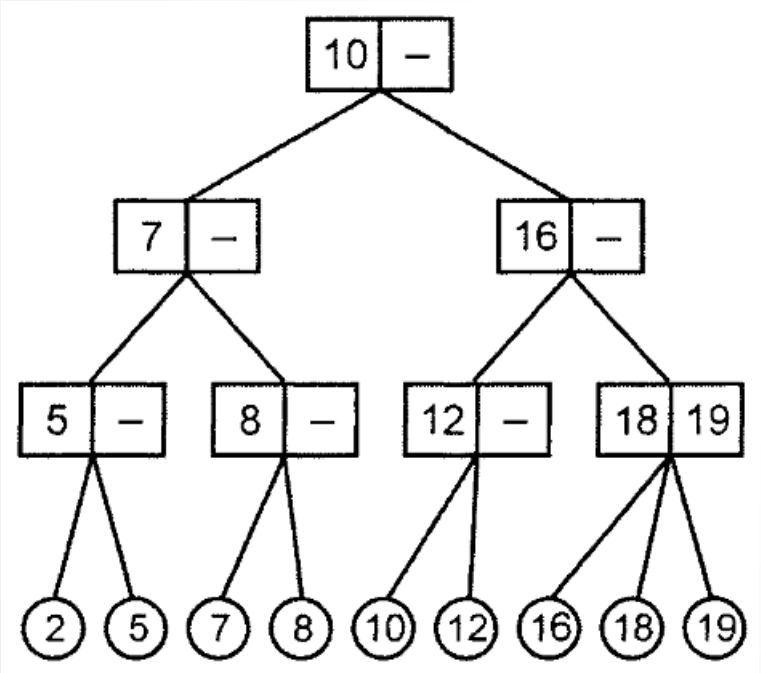
2-3 дерево

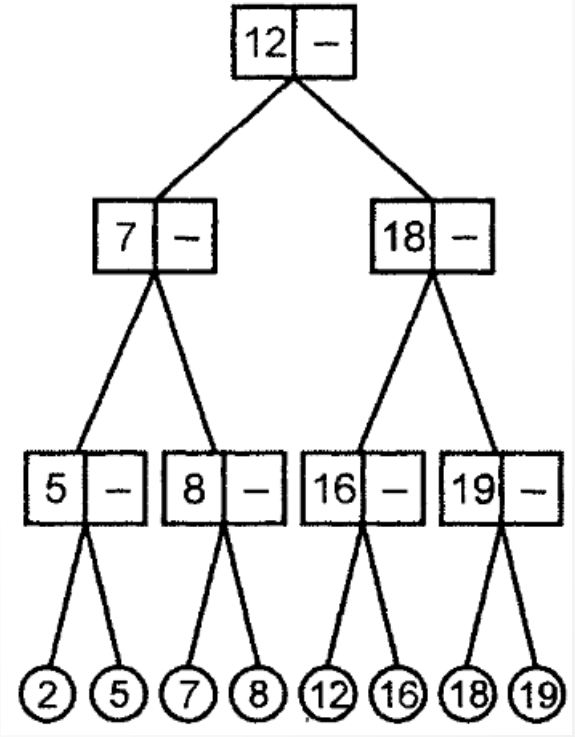
- Удаление элемента с ключом *x* :
  - Найти лист, содержащий значение x
  - Удалить этот лист
  - Пусть *v* — родитель удалённого листа
  - Если у вершины *v* осталось 2 дочерних вершины, то всё в порядке
  - Иначе: пусть p — родитель v.
    - Если у вершины p кроме v есть дочерняя вершина q, с 3 дочерними вершинами, то одна из дочерних вершин q становится дочерней вершиной v.

2-3 дерево

- Если у вершины p кроме v есть дочерняя вершина q, с 3 дочерними вершинами, то одна из дочерних вершин q становится дочерней вершиной v.

Пример: удаляем элемент с ключом 10:

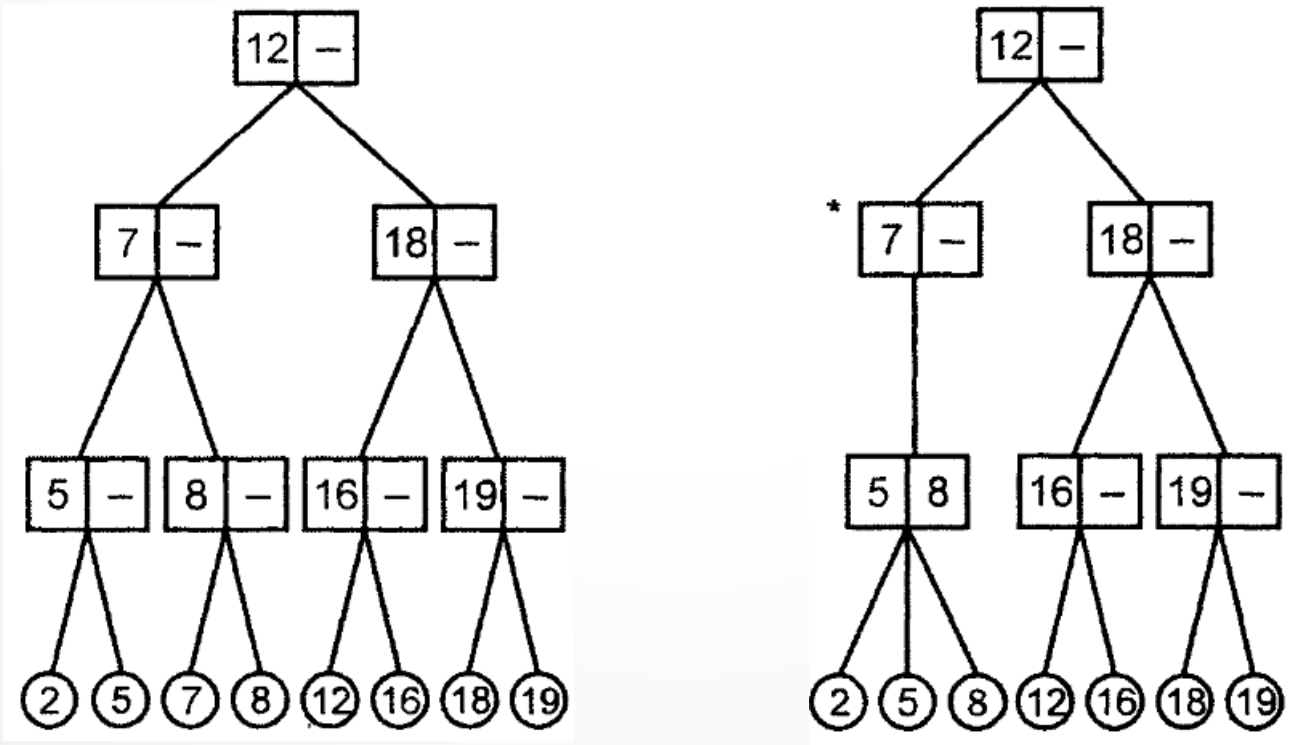




2-3 дерево

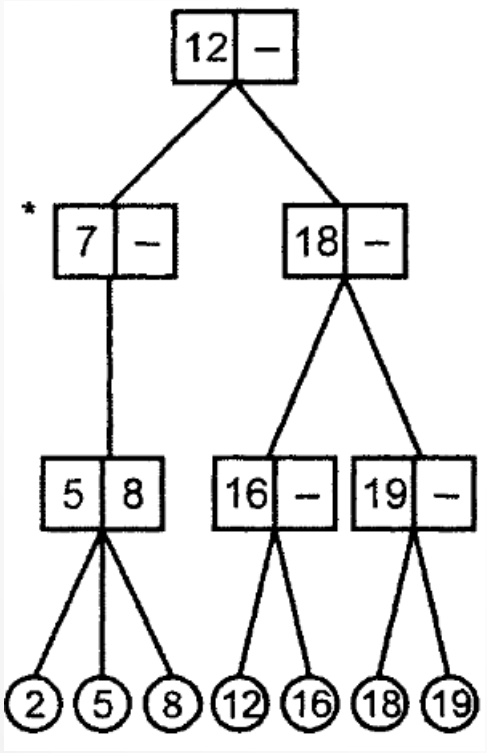
- Если у вершины p кроме v есть дочерняя вершина q, с 2 дочерними вершинами, то дочерняя вершина v становится дочерней вершиной q.

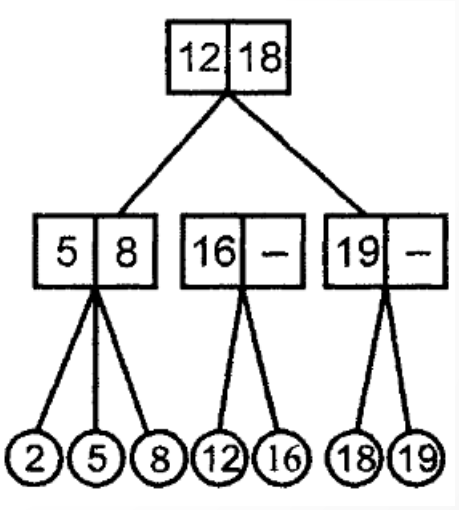
Пример: удаляем элемент с ключом 7:



2-3 дерево

- Эта процедура повторяется рекурсивно, при необходимости — до корня.
- Если у корня остаётся единственная дочерняя вершина, то она становится новым корнем.





Кучи

Куча (пирамида, сортирующее дерево): реализация очереди с приоритетами со сложностью операций:

- GetMin / GetMax : O(1)
- DelMin / DelMax : O(log *n*)

Представляет собой дерево, в вершинах которого (и внутренних, и листьях) хранятся элементы с ключами. Выполняются *условия кучи*:

(1) Ключ корня является максимальным / минимальным из всех ключей

(2) Для каждого из поддеревьев выполняются условия (1) и (2)

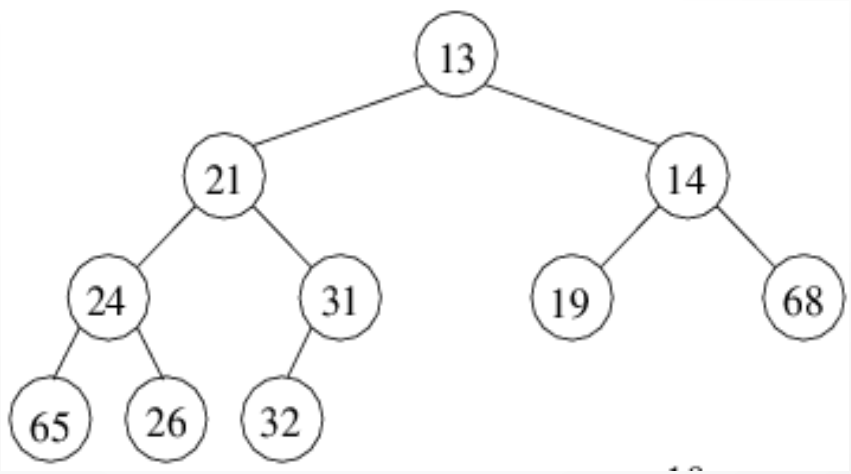
Для двоичных куч — двоичное дерево.

Двоичная куча

Двоичная куча — почти полное двоичное дерево, удовлетворяющее условию кучи.

«Почти полное»:

- все уровни, кроме, может быть, самого нижнего, заполнены полностью (на глубине k содержится 2^k вершин);
- на нижнем уровне заполнение идёт «слева направо»



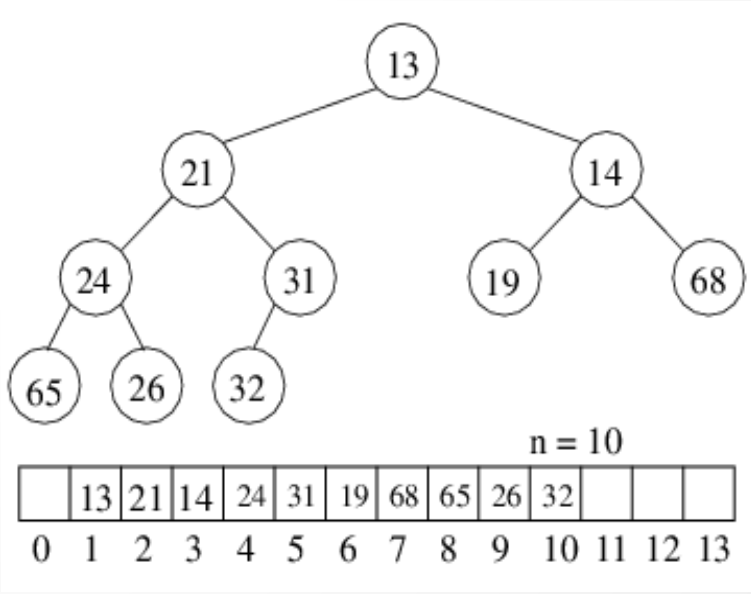
Двоичная куча

Двоичные кучи удобно хранить «упакованными» в обычный массив.

Если *x* хранится в ячейке с индексом *i*, то:

- Левый потомок — в ячейке $2i$ (если $2i > n$, то левого потомка нет).
- Правый потомок — в ячейке 2i+1 (если 2i+1 > n, то правого потомка нет).

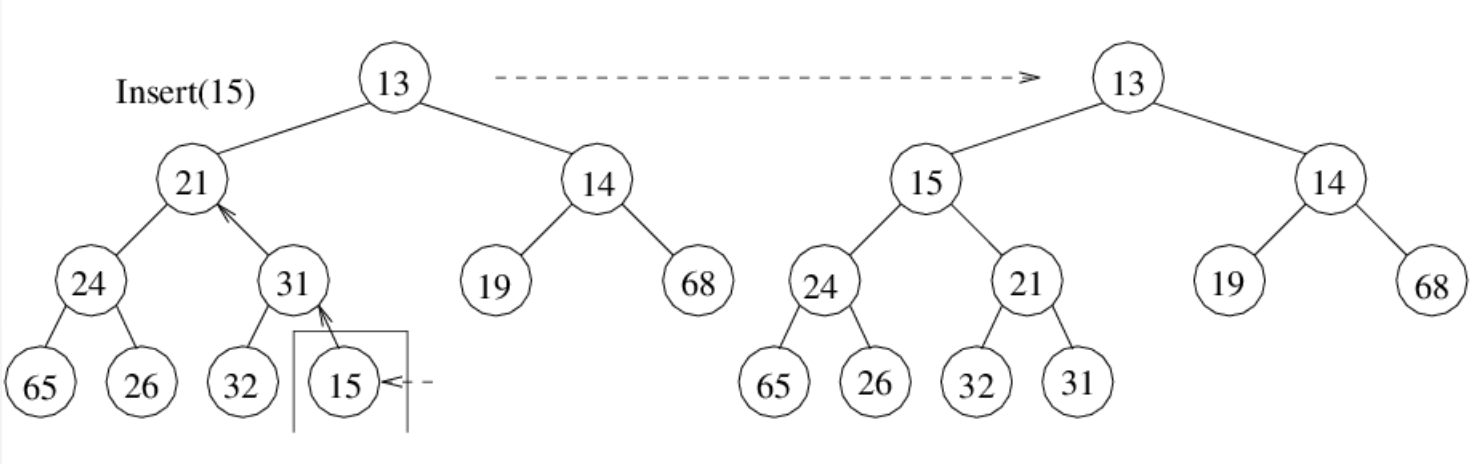
Родитель — в ячейке $\lfloor i/2 \rfloor$ (для $i > 1$).
Корень — в ячейке с индексом 1.



Двоичная куча

Операции с двоичной кучей (для минимизирующей кучи):

- Insert:
  - добавляем новый элемент в конец
  - если его ключ меньше, чем у родителя, то меняем его местами с родителем
  - и т. д. (процедура *всплывание* , sift-up).

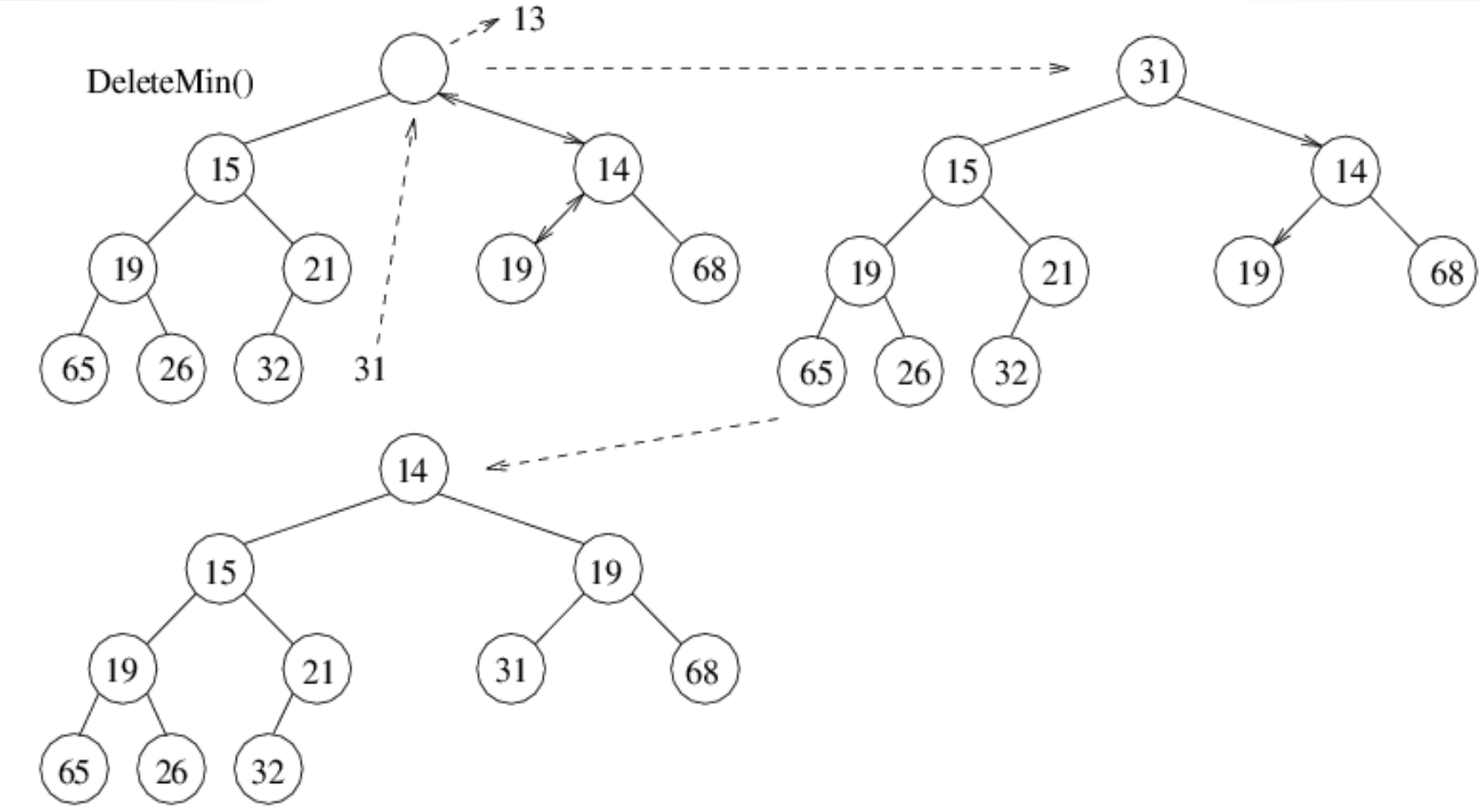


Двоичная куча

- GetMin: вернуть значение ячейки с индексом 1
- DelMin:
  - В ячейку с индексом 1 переместить элемент из последней ячейки.
  - Рекурсивно выполнить процедуру «*пересыпка*» (sift-down):
    - меняем текущий элемент с элементом из дочерней вершины, имеющей наименьший ключ;
    - если у всех дочерних вершин ключ больше, чем в текущей вершине — останавливаемся.

Двоичная куча

- DelMin:



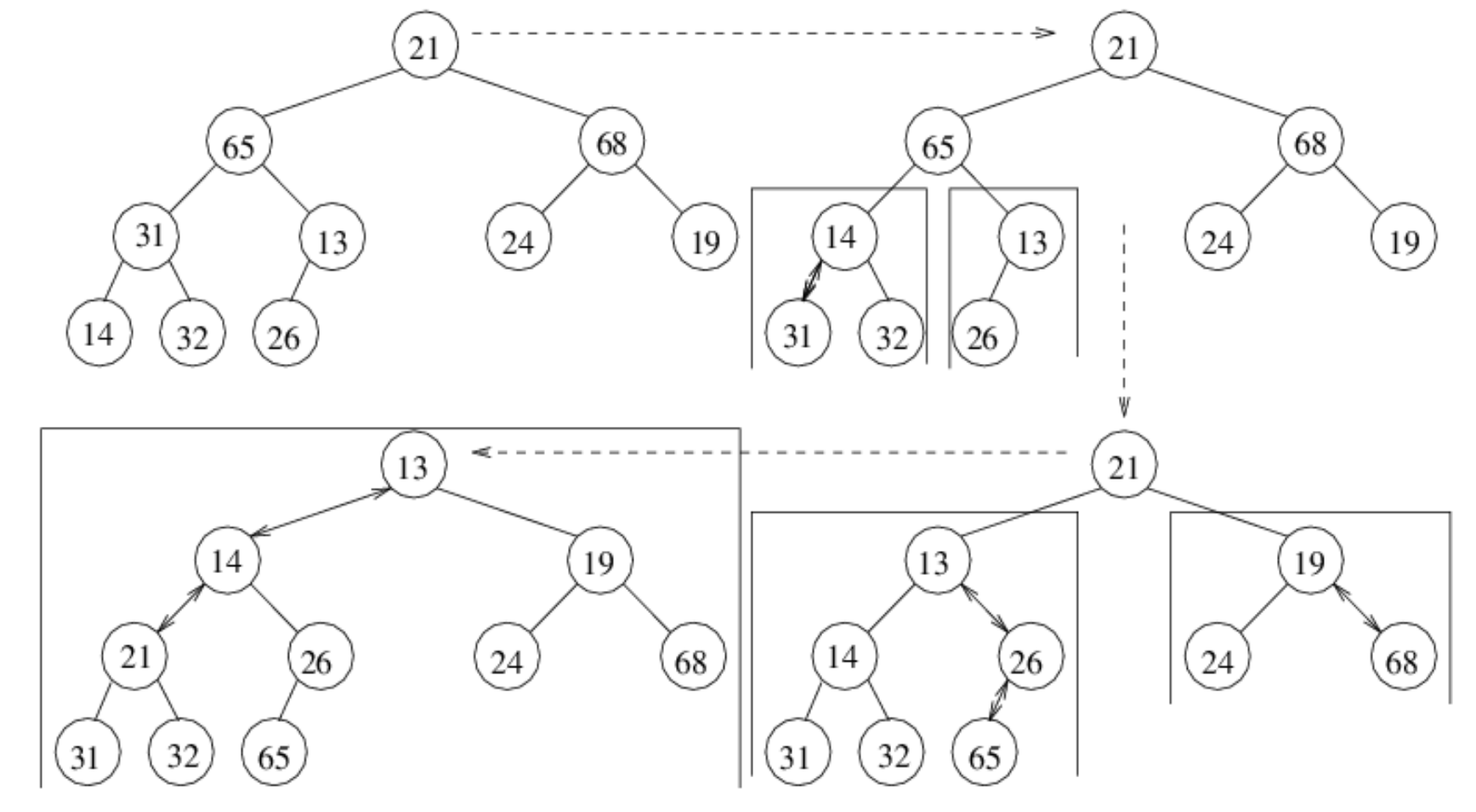
Двоичная куча

Построение двоичной кучи:

- Данные хранятся в массиве A[1..*n*]
- Представим массив в виде структуры двоичной кучи (но с нарушением *условия кучи*).
- Начиная с самой глубокой внутренней вершины, и проходя по каждому уровню справа налево и потом вверх к предыдущему уровню, применим «пересыпку» (sift-down). Пока не поднимемся к корню.

Двоичная куча

Построение кучи:



Двоичная куча

В терминах массива:

*BuildBinaryHeap(*A[1..n]*)*:

for i:= n div 2 downto 1 do

SiftDown(A, *i*, *n*)

Сложность процедуры построения:

- Для поддерева высоты *r* процедура требует времени O(*r*).
- Количество вершин высоты *r* равно 2^{h-r}, где $h \sim \log_2 n$
- => T(n) = сумма $r2^{h-r}$ по *r*=0..*h*.
- => T(n) = O(n)

Другие кучи

Операция	Двоичная	Биномиальная	Фибоначчиева
GetMin	O(1)	O(1)	O(1)
DelMin	O(log n)	O(log n)	O(log n)
Insert	O(log n)	O(1)	O(1)
DecreaseKey	O(log n)	O(log n)	O(1)
Merge	O(m log (m+n))	O(log n)	O(1)

Задание 5

Доработать программу решения задачи коммивояжёра (задание 4), добавив к 3 реализованным методам решения метод «МВГ + эвристика 'сначала наилучшие'».

Для реализации очереди с приоритетами использовать двоичную кучу.