Бинарная куча min/max: теория, операции и псевдокод

31 октября 2025 г.

Содержание

1 Определение

Определение

Бинарная куча (heap) — это полное бинарное дерево, удовлетворяющее кучному инварианту. Различают два варианта:

- Min-куча (min-heap): ключ в любой вершине не превосходит ключей её потомков.
- Max-куча (max-heap): ключ в любой вершине не меньше ключей её потомков.

Полнота означает, что дерево заполняется уровнями слева направо без "дырок". Благодаря этому кучу удобно хранить в массиве.

2 Представление в массиве

Пусть массив A хранит вершины по уровням (корень — в $A[1]$). Тогда индексация такова:

$$\text{left}(i) = 2i, \qquad \text{right}(i) = 2i+1, \qquad \text{parent}(i) = \left\lfloor \frac{i}{2} \right\rfloor.$$

Размер актуальной кучи обозначим n (первые n элементов массива). Такая раскладка обеспечивает $O(1)$ доступ к родителю/детям и занимает $O(n)$ памяти.

Важно: Во многих языках программирования (C++, Python, Java) принята индексация с нуля. В этом случае формулы меняются:

$$\text{left}(i) = 2i+1, \quad \text{right}(i) = 2i+2, \quad \text{parent}(i) = \left\lfloor \frac{i-1}{2} \right\rfloor.$$

3 Базовые операции и сложности

Операция	Min-куча	Max-куча
Extremum() (минимум/максимум, чтение корня)	$O(1)$	$O(1)$
Insert(x)	$O(\log n)$	$O(\log n)$
ExtractExtremum() (удаление корня)	$O(\log n)$	$O(\log n)$
DecreaseKey(i,Δ)/		

Идея подъёма/спуска. Для восстановления инварианта применяются две примитивные процедуры:

- SiftUp(i) (Просеивание вверх) — сравнение с родителем и подъём узла вверх.
- SiftDown(i) (Просеивание вниз) — сравнение с лучшим потомком и спуск узла вниз.

В полном бинарном дереве высота $\Theta(\log n)$, отсюда асимптотики.

4 Псевдокод для min-кучи

Ниже приведён типичный вариант на массиве $A[1..n]$. Замены $<$ на $>$ дают max-кучу.

Algorithm 1 SiftUp(i) для min-кучи

```
while i > 1 and A[i] < A[⌊i/2⌋] do
    Swap A[i] и A[⌊i/2⌋]
    i ← ⌊i/2⌋
end while
```

Algorithm 2 SiftDown(i) для min-кучи

```
while true do
    l ← 2i, r ← 2i + 1, m ← i
    if l ≤ n and A[l] < A[m] then
        m ← l
    end if
    if r ≤ n and A[r] < A[m] then
        m ← r
    end if
    if m = i then
        break
    end if
    Swap A[i] и A[m]
    i ← m
end while
```

Algorithm 3 Insert(x)

```
n ← n + 1
A[n] ← x
SiftUp(n)
```

Algorithm 4 GetMin()

```
return A[1]
```

Algorithm 5 ExtractMin()

```
ans ← A[1]
A[1] ← A[n]
n ← n − 1
SiftDown(1)
return ans
```

Algorithm 6 DecreaseKey(i, Δ)

```
A[i] ← A[i] − Δ                                  ▷ Δ > 0
SiftUp(i)
```

Algorithm 7 BuildMinHeap(A)

```
n ← |A|
for i ← ⌊n/2⌋ down to 1 do
    SiftDown(i)
end for
```

Max-куча. Для max-кучи сравнения меняются на противоположные ($< \leftrightarrow >$), имена процедур — GetMax, ExtractMax, IncreaseKey и т.п.

5 Сортировка кучей (HeapSort)

Вариант 1 (через max-кучу).

1. Построить max-кучу за $O(n)$.
2. Повторять n раз: обменять $A[1]$ и $A[n]$, уменьшить n, выполнить SiftDown(1).

Получаем неубывающую сортировку за $O(n \log n)$, in-place и без дополнительной памяти (кроме $O(1)$). Алгоритм неустойчив.

Algorithm 8 HeapSort(A)

```
n ← |A|
BuildMaxHeap(A)                ▷ Использует SiftDown с обратными сравнениями
for i ← n down to 2 do
    Swap A[1] и A[i]
    n ← n − 1
    SiftDown(1)                ▷ На куче размера n − 1
end for
```

6 Почему BuildHeap работает за $O(n)$

Элементы на глубине h могут опуститься вниз не более чем на h уровней. Число вершин на глубине h около $n/2^{h+1}$. Суммарная сложность

$$\sum_{h \geq 0} \frac{n}{2^{h+1}} \cdot h = O(n).$$

Интуитивно: большая часть вершин — листья или почти листья; их спуск короткий, что и даёт линейное время.

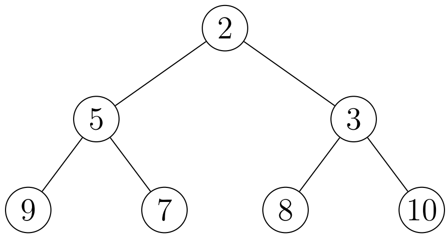
7 Пример работы с min-кучей

Рассмотрим последовательность операций для min-кучи. Исходно куча пуста: $A = [\,]$, $n = 0$.

1. Insert(5): $A = [5]$
2. Insert(3): $A = [3, 5]$ (после SiftUp)
3. Insert(8): $A = [3, 5, 8]$
4. Insert(1): $A = [1, 3, 8, 5]$ (после SiftUp)
5. GetMin(): возвращает 1
6. ExtractMin():
    - Возвращает 1

- $A[1] = A[4] = 5$, $n = 3$, $A = [5, 3, 8]$
- SiftDown(1): $A = [3, 5, 8]$

8 Наглядный пример (min-куча)



Этот рисунок соответствует массиву $A = [\,2,\ 5,\ 3,\ 9,\ 7,\ 8,\ 10\,]$ (индексация с 1). Видно, что каждый родитель не больше своих детей.

9 Практические заметки по реализации

- Индексация с нуля удобна в языках C/C++/Python. Тогда формулы: left(i)=2i+1, right(i)=2i+2, parent(i)=(i-1)/2.
- При равных ключах стоит определиться с политикой стабильности и направлением сравнения, чтобы поведение было детерминированным.
- Для структур с часто изменяемым приоритетом используйте DecreaseKey/IncreaseKey и храните обратные индексы (handles) к позициям в куче.
- Вставки пачкой быстрее делать через BuildHeap, а не последовательными Insert.
- Сложность по памяти: $O(n)$ для хранения массива элементов.

10 Краткая сводка различий min- и max-кучи

- Инвариант: min: $parent \le child$; max: $parent \ge child$.
- Экстремум: min: корень — минимум; max: корень — максимум.
- Остальные операции: идентичны по структуре, меняется лишь знак сравнения.