Кучи

Борисов Иван Максимович

2025.11.09

1 Алгоритм построения кучи за $O(n \log n)$

1.1 Описание алгоритма

Первый алгоритм основан на функции `shift_up` и имеет сложность $O(n \log n)$. Докажем это.

1.2 Доказательство сложности $O(n \log n)$ в случае использования `shift_up`

Пусть n - количество элементов в массиве, h - высота кучи.

1. $\forall i = 1, \ldots, n-1$ вызывается функция `shift_up`, где i - индекс элемента в массиве.
2. В худшем случае операция `shift_up` для элемента на уровне h требует h перестановок элементов.
3. Высота кучи $h = \lfloor \log_2 n \rfloor$.

Объединим эти 3 факта:

$$T(n) = \sum_{i=1}^{n} O(\log i) = O\left(\sum_{i=1}^{n} \log i\right)$$

Преобразуем:

$$\sum_{i=1}^{n} \log i = \log(n!) = \log(1 \cdot 2 \cdot \ldots \cdot (n-1) \cdot n) \leq \log(n \cdot n \cdot \ldots \cdot n \cdot n) =$$
$$= \log(n^n) = n \log n$$

Таким образом:

$$\boxed{T(n) = O(n \log n)}$$

2 Алгоритм построения кучи за $O(n)$

2.1 Описание алгоритма

Второй алгоритм основан на функции `shift_down` и имеет сложность $O(n)$. Докажем это.

2.2 Доказательство сложности $O(n)$

Пусть n - количество элементов, h - высота кучи $\Leftrightarrow$ высота корня (определим высоту вершины как длину самого длинного пути от этой вершины до листа).

1. Высота кучи: $h = \lfloor \log_2 n \rfloor$
2. Пусть d_k - количество вершин высоты $k \Rightarrow d_k \leq \lceil \frac{n}{2^{k+1}} \rceil$
3. Количество перестановок в функции `shift_down` для вершины на уровне k не более k $\Rightarrow O(k)$

Объединим эти 3 факта:

$$T(n) = \sum_{k=0}^{h} d_k \cdot O(k) \leq \sum_{k=0}^{h} \left\lceil \frac{n}{2^{k+1}} \right\rceil \cdot c \cdot k = \frac{cn}{2} \cdot \sum_{k=0}^{h} \frac{k}{2^k}$$

Известно, что:

$$S(x) = \sum_{j=0}^{\infty} j x^j = \frac{x}{(1-x)^2}, \text{при } |x| < 1$$

$$S\left(\frac{1}{2}\right) = \frac{\frac{1}{2}}{(1-\frac{1}{2})^2} = 2 \Rightarrow \sum_{k=0}^{h} \frac{k}{2^k} \leq 2 \text{ (сходится)}$$

Таким образом:

$$\boxed{T(n) = \hat{c} \cdot n = O(n)}$$

3 Сравненение времени работы

Size	**O(n log n)**	**O(n)**	**Ratio**
10^2	0.000031	0.000025	1.23
10^3	0.000299	0.000285	1.05
10^4	0.003069	0.002790	1.10
10^5	0.030624	0.026554	1.15
10^6	0.309803	0.275316	1.13

Таблица 1: Сравнение производительности алгоритмов в секундах при заданных размерах массива (элементы массива генерируются случайно)