Двоичная куча (heap, пирамида)

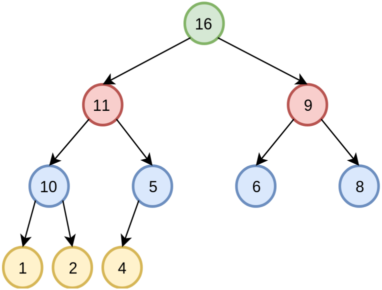
Булгаков Илья, Гусев Илья, Валерий Сенотов, Виталий Ерошин

Московский физико-технический институт

Москва, 2023

Содержание

Двоичная куча (heap, пирамида)



1. Двоичное дерево (связный ациклический граф, у которого у любой вершины не больше 2 потомков)
2. Если узел B является потомком узла A, то $A.key \geq B.key$ (max-куча). Для min-кучи наоборот.
3. Глубина всех листьев (расстояние до корня) отличается не более чем на 1 слой.
4. Последний слой заполняется слева направо без «дырок».

Двоичная куча

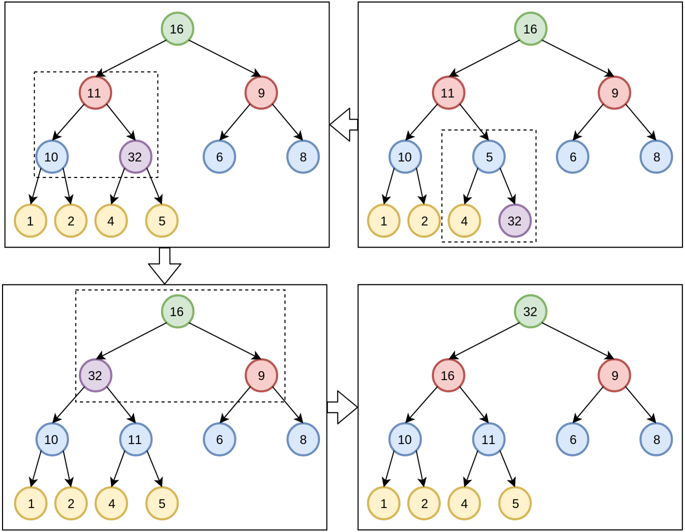
Реализация

0	1	2	3	4	5	6	7	8	9
16	11	9	10	5	6	8	1	2	4

- $A[0]$ - корень
- $\forall i \; A[2i+1]$ - левый потомок $A[i]$
- $\forall i \; A[2i+2]$ - правый потомок $A[i]$

Двоичная куча

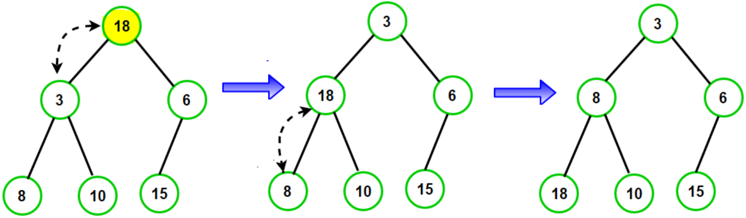
Подъем элемента в куче (SiftUp)



Двоичная куча

Просеивание (heapify, SiftDown)

- Применяется, если корень не удовлетворяет свойству кучи
- Правое и левое поддерево удовлетворяют
- Итеративно меняем местами с меньшим (для min-кучи) потомком, пока свойство кучи не будет восстановлено

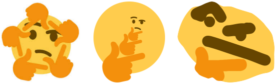


Двоичная куча

Действия и сложность

1. Добавить элемент в кучу: добавить в конец и осуществить подъем SiftUp. Сложность $\mathcal{O}(\log n)$
2. Исключить максимальный элемент из кучи: поставить последний элемент в корень, уменьшить количество элементов, выполнить heapify. Время работы $\mathcal{O}(\log n)$
3. Изменить значение любого элемента. Время работы $\mathcal{O}(\log n)$

- Превратить неупорядоченный массив элементов в кучу. Сложность $\mathcal{O}(n)$



Двоичная куча

Построение

Есть разные способы построить кучу

- Строить через обычные операции добавления в конец и SiftUp. Тут будет сложность $\mathcal{O}(n * \log n)$
- Превратить неупорядоченный массив элементов в кучу более эффективно. Сложность $\mathcal{O}(n)$

Двоичная куча

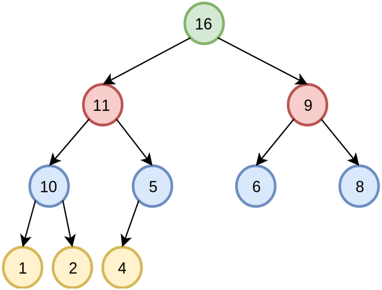
Построение за $\mathcal{O}(n)$

Идея: Сделаем siftDown для вершин, имеющих хотя бы одного потомка: от n/d до 0, так как поддеревья, состоящие из одной вершины без потомков, уже упорядочены.

Двоичная куча

Построение за $\mathcal{O}(n)$

$\lceil \frac{n}{2^{h+1}} \rceil$ - максимум количества элементов на уровне h
$\mathcal{O}(h)$ - сложность вставки элемента на уровень h
$\lfloor \lg(n) \rfloor$ - высота n-элементной пирамиды



Двоичная куча

Построение за $\mathcal{O}(n)$

$$\sum_{h=0}^{\lfloor \lg(n) \rfloor} \lceil \frac{n}{2^{h+1}} \rceil \mathcal{O}(h) = \mathcal{O}(n \sum_{h=0}^{\lfloor \lg(n) \rfloor} \frac{h}{d^h})$$

$$\sum_{n=1}^{\infty} \frac{n}{d^n} = \frac{d}{(d-1)^2}$$

Обозначим за s сумму ряда. Заметим, что $\frac{n}{d^n} = \frac{1}{d} \cdot \frac{n-1}{d^{n-1}} + \frac{1}{d^n}$.

$\sum\limits_{n=1}^{\infty} \frac{1}{d^n}$ — это сумма бесконечной убывающей геометрической прогрессии, и она равна $\frac{\frac{1}{d}}{1-\frac{1}{d}} = \frac{1}{d-1}$.

Получаем $s = \frac{1}{d} \cdot s + \frac{1}{d-1}$. Откуда $s = \frac{d}{(d-1)^2}$.

$$\sum_{h=0}^{\infty} \frac{h}{d^h} = \frac{d}{(d-1)^2}$$

$$\sum_{h=0}^{\infty} \frac{h}{2^h} = \frac{2}{(2-1)^2} = 2$$

$$\mathcal{O}(n \sum_{h=0}^{\lfloor \lg(n) \rfloor} \frac{h}{2^h}) = \mathcal{O}(n \sum_{h=0}^{\infty} \frac{h}{2^h}) = \mathcal{O}(2n) = \mathcal{O}(n)$$

Двоичная куча

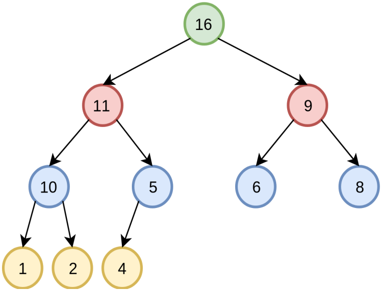
Сортировка за $\mathcal{O}(nlogn)$

$\lceil n \rceil = 2^k$

2^{k-h+1} - максимум количества элементов, поднятых с уровня h

$\mathcal{O}(k-h)$ - сложность просеивания элемента с уровня h

k - высота n-элементной пирамиды



Двоичная куча

Сортировка за $\mathcal{O}(nlogn)$

$$\sum_{h=0}^{k} 2^{k-h+1}\mathcal{O}(k-h) = 2\mathcal{O}(\sum_{h=0}^{k}(k-h)2^{k-h})$$

$$\sum_{h=0}^{k}(k-h)2^{k-h} = \sum_{i=0}^{k} i2^{i} = S(k)$$

$$S(k) = 2 * S(k) - S(k) = \sum_{i=0}^{k} i2^{i+1} - \sum_{i=0}^{k} i2^{i} = \sum_{i=1}^{k+1}(i-1)2^{i} - \sum_{i=0}^{k} i2^{i} = k * 2^{k+1} - \sum_{i=1}^{k} 2^{i} = k * 2^{k+1} - (2^{k} - 1)$$

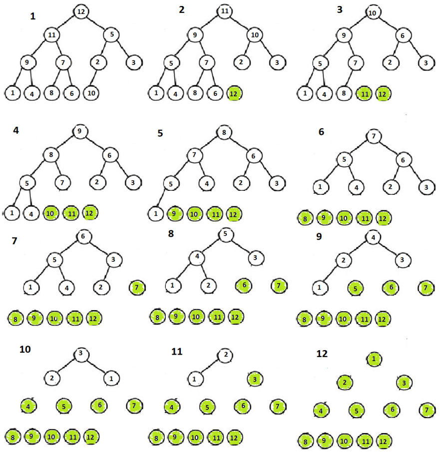
$$\mathcal{O}(\sum_{h=0}^{k}(k-h)2^{k-h}) = \mathcal{O}(k * 2^{k+1} - 2^{k}) = \mathcal{O}(k * 2^{k+1}) = \mathcal{O}(nlogn)$$

Библиотечные функции для работы с кучей

1. std::make_heap - метод построения кучи
   Объявлен в заголовочном файле <algorithm>

```
#include <algorithm>
int main() {
    std::vector<int> v { 3, 2, 4, 1, 5, 9 };
    std::make_heap(v.begin(), v.end());
    std::pop_heap(v.begin(), v.end());
}
```

Пирамидальная сортировка (HeapSort)



1. Строим над коллекцией кучу
2. Делаем ExtractMin n раз (минимум перемещаем в конец)
3. ...
4. PROFIT!

Пирамидальная сортировка (HeapSort)

1. Сложность?
2. Устойчивость?
3. Доп. память?
4. Сложность на уже сортированных массивах?

Пирамидальная сортировка (HeapSort)

1. (-) неустойчивая;
2. (-) на почти отсортированных данных работает столь же долго, как и на хаотических данных;
3. (+) худшее время работы гарантированный $n * log(n)$;
4. (+) требует $O(1)$ дополнительной памяти.

Кучи. Сортировки

Задача 1

Отсортированные массивы:

$A_1 \quad [a_1^1 \ldots a_{n_1}^1]$

$\vdots$

$A_k \quad [a_1^k \ldots a_{n_k}^k]$

Как получить отсортированное объединение массивов за $O((|A_1| + \cdots + |A_k|) logk)$?

Кучи. Сортировки

Задача 1

Отсортированный массив из n элементов перемешали так, что каждый элемент сдвинут не более, чем на k позиций.
Как заново отсортировать массив за $O(nlogk)$?

Полезные ссылки I

- Т.Кормен, Ч.Лейзерсон, Р.Ривест, К.Штайн - Алгоритмы. Построение и анализ. Глава 6
  https://bit.ly/2wFzphU
- Lecture Slides for Algorithm Design
  https://algs4.cs.princeton.edu/lectures/