Какое-то сегодняшнее дополнение про бинарную кучу есть в прошлом конспекте

K-чная куча Куча на полном K-чном дереве. Эту кучу можно так же хранить в массиве, с 0-индексацией.

- $sift\_up$ такой же, $O(\log_k n)$
- $sift\_down$ ищет минимум среди k элементов на каждом шаге, поэтому работает за $O(k \cdot \log_k n)$.

	$2 - heap$	$k - heap$
$insert$	$O(\log n)$	$O(k \log_k n)$
$extract\_min$	$O(\log n)$	$O(k \log_k n)$
$decreasekey$	$O(\log n)$	$O(\log_k n)$
$increasekey$	$O(\log n)$	$O(k \log_k n)$

Дейкстра на K-ной куче Алгоритм Дейкстры достает минимум n раз, и улучшает ключ m раз

$$a \cdot \log_k n = b \cdot k \cdot \log_k n, a = m, b = n$$

Отсюда $k = \frac{a}{b} = \frac{m}{n}$ в случае Дейкстры. Еще отметим, что $k \geq 2$.

В случае когда $a = b^q,\ q > 1$, то k-куча структура работает за $O(1)$ (вроде бы этот факт мы докажем в домашке), причем с хорошей константой, поэтому применимо на практике (привет, фибоначчиева куча!).

Амортизационный анализ Идея в том, что мы хотим оценить суммарное число операций, а не на каждом шаге работы. То есть вполне может быть итерация алгоритма за $O(n)$, но нам важно, что суммарное число $O(n \log n)$

Так что есть $t_{real} = t,\ t_{amortized} = \tilde{t}$.

Метод кредитов Элементам структуры сопоставляем сколько-то монет. Этими монетами элемент «расплачивается» за операции. Также мы накидываем сколько-то монет на операцию. Запрещаем отрицательное число монет. Начинаем с нулем везде.

Обозначим состояния структуры за $S_0, S_1, \ldots, S_n$. Каждый переход стоил t_i, $t_i \geq |operations|$, Где t_i — это сколько мы потратили. Также на i-м шаге мы вбрасываем в систему $\tilde{t}_i$ монет. Тогда

$$\sum t_i \leq \sum \tilde{t}_i \leq A \to O(A)$$

Стек с минимумом

- min_stack
- push
- pop
- get_min

$m_i = \min(m_{i-1}, a_i)$ — поддерживаем минимумы. Операции тривиальны

Очередь с минимумом на двух стеках Храним два стека с минимумом, один из которых мысленно наращиваем в одну сторону, а другой в другую, при этом очередь выглядит как бы как склеенные стеки. То есть мы добавляем элемент в первый стек, а извлекать хотим из второго.

$$X \to a_n, a_{n-1}, \ldots, a_1, \ | \ , b_1, b_2, \ldots, b_n \to Y$$

Тогда единственная сложная операция — если мы хотим извлечь минимум, а второй стек пустой. Тогда мы все элементы из первого перекинем во второй по очереди с помощью «извлеки-добавь»

Почему это работает за $O(1)$ на операцию амортизированно? Представим каждому элементу при рождении 2 монеты, одну из которых мы потратим на добавление в первый стек, а вторую на удаление через второй.

set для бедных Хотим не делать $erase$, только $insert$, $find$, get_min. Храним $\log n$ массивов, $|a_i| = 2^i$, каждый из которых по инварианту будет отсортирован. Тогда get_min рабтает за $O(\log n)$ — просто берем минимум по всем массивам. Аналогично $find$ делается бинпоисками за $O(\log^2 n)$

А как добавлять за $\tilde{O}(\log n)$? Каждый элемент при добавлении в структуру получает $\log n$ монет. Когда мы добавляем элемент, мы создаем новый массив ранга 0. Если было два массива ранга 0, сольем их в новый массив ранга 1 за суммарный размер (и заберем монетку у всех элементов во время слияния), и так далее, пока не создадим уникальный массив для текущего ранга.

Метод потенциалов $\Phi(S_i)$ — потенциал, который зависит только от состояния структуры (**не от** последовательности действий, которая к такому состоянию привела).

Опять вводим t_i, $\sum t_i = O(f(n))$. Определим амортизированное время работы:

$$\tilde{t_i} = t_i + (\Phi(S_{i+1}) - \Phi(S_i))$$

$$t_i = \tilde{t_i} + \Phi(S_i) - \Phi(S_{i+1})$$

Пусть мы показали $\tilde{t_i} \le f(n)$. Тогда

$$\sum t_i \le n \cdot f(n) + \Phi(0) - \Phi(n)$$

Нормальный потенциал — такой, что из неравенства выше все еще можно показать O-оценку на $\sum t_i = O(n \cdot f(n))$.

deque для богатых Хотим deque с поддержкой минимума.

Храним два стека как для обычной очереди. Все операции хорошо работают как на очереди, кроме перестройки структуры. В случае с очередью надо было переливать стеки только в одну сторону, а теперь иногда нужно туда-сюда.

Теперь мы будем перекидывать только половину элементов. Тогда нам понадобитсяя 3 стека, один из которых будет вспомогательным для перестройки (там иначе стеки развернуты).

$\Phi(S_i) = |Size_1 - Size_2|$