

SPb HSE, 1 курс, осень 2021/22

Практика по алгоритмам #7

Разделяй и властвуй, кучи

15 октября

Собрано 9 ноября 2021 г. в 12:44

Содержание

1. Разделяй и властвуй и друзья	1
2. Кучи	1
3. Дополнительные задачи	2
4. Разбор задач практики	3
4.1. Разделяй и властвуй и друзья	3
4.2. Кучи	4
4.3. Дополнительные задачи	5
5. Домашнее задание	7
5.1. Обязательная часть	7
5.2. Дополнительная часть	8

Разделяй и властвуй и друзья

Мы уже знакомы с методом «Разделяй и властвуй» на примерах алгоритмов: MergeSort, Карацубы, QuickSort, nth_element...

1. Число инверсий

Для каждого i найти $|\{j < i \mid a_j > a_i\}|$ и $|\{j > i : a_j < a_i\}|$. То есть, число инверсий, образованных элементом i слева и справа. Для упрощения все a_i различны.

2. Минимум в offline

Есть массив длины n и m отрезков $[l_i, r_i]$. За $\mathcal{O}((n + m) \log n)$ найти минимум на каждом из этих отрезков.

3. Ближайшие точки

Дано n точек (x_i, y_i) , найти пару ближайших.

4. Inplace алгоритмы

- Unique: оставить в массиве одну копию каждого элемента за $\mathcal{O}(n \log n)$.
- Reverse: перевернуть массив за $\mathcal{O}(n)$.
- Rotate: сделать циклический сдвиг массива за $\mathcal{O}(n)$.
- Придумайте stable inplace merge за $\mathcal{O}(n \log n)$.

Кучи

1. d -куча

d -куча – куча, где у каждой вершины d детей. Как ее хранить?

Нам нужна d -куча на n элементах, из которой мы n раз сделаем ExtractMin и m раз DecreaseKey. Каков оптимальный выбор d ? В будущем это пригодится Дейкстре.

2. Повторение Van Emde Boas tree

- Научитесь делать lowerBound(x) за $\mathcal{O}(\log \log C)$.
- (*) В Van Emde Boas tree постоянно удваиваются все мелкие хеш-таблицы. Пусть мы знаем, что в нашей куче никогда не больше n элементов, как обезопасить себя от потребности с удвоением?
- (*) Нужно ли как-то модифицировать Van Emde Boas tree, чтобы там можно было хранить повторяющиеся элементы?

3. Левацкий бамбук

Предъявить последовательность действий, при которой LeftistHeap из пустой станет бамбуком (цепочкой из вершин, у каждой один ребенок).

4. Операции над Skew Heap

- Придумайте разумный способ удаления из Skew Heap по ссылке на узел.
- Придумайте DecreaseKey для Skew Heap.
- Покажите, что в любой куче можно реализовать одну из операций DecreaseKey, Delete, если уже реализована другая. DecreaseKey \Leftrightarrow Delete.

5. Куча с медианой

- Придумайте структуру данных на основе `heap`, которая умеет делать `Insert(x)`, `GetMedian()`, `DeleteMedian()`, все операции за $\mathcal{O}(\log n)$.
- `Insert(x)`, `Get[Min,Max,Median]`, `Delete[Min,Max,Median]`, все операции за $\mathcal{O}(\log n)$.
- В предположении, что в куче никогда не будет больше n элементов, постройте предыдущую структуры с использованием $n \times \text{sizeof}(\text{Item}) + \mathcal{O}(1)$ памяти.

Дополнительные задачи

1. Подматрица с максимальной медианой

В матрице Q размера $N \times N$ найти подматрицу размера $H \times W$ с максимальной медианой. H, W – нечетные. $\mathcal{O}(N^2 \log N)$.

2. Сумма векторов

Дано множество из n векторов на плоскости. Разрешается любой вектор умножить на -1 . Найти пару векторов с минимальной суммой.

3. Сумма векторов #2

Дано множество из n векторов на плоскости. Разрешается любой вектор умножить на -1 . Минимизировать сумму всех векторов.

4. Медиана без памяти

Вы можете одним проходом прочесть массив $2n - 1$ чисел. Разрешается хранить $n + 1$ элемент массива и небольшую константу памяти. Найти медиану за $\mathcal{O}(n \log n)$.

5. Статистика на суммах подмножеств

Дано множество из n элементов. Найдите k порядковую статистику среди сумм всех 2^n подмножеств. $n \leq 10^5, k \leq 10^6$.

6. Трудная куча

Вспомним шаги `HeapSort`.

- Поменять первый и последний элемент кучи местами.
- Уменьшить размер кучи на единицу.
- Запустить `SiftDown` на первом элементе. `SiftDown` сравнивает детей элемента, затем элемент с большим ребенком, и при необходимости меняет их и продолжает работу с новой позиции.

Построить перестановку из n элементов, являющуюся `max` кучей, на которой произойдет максимальное число сравнений элементов кучи во время `HeapSort`. $\mathcal{O}(n \log n)$.

Разбор задач практики

4.1. Разделяй и властвуй и друзья

1. Число инверсий

Рекурсивно считаем для левой и правой половины. Заодно сортируем, сохраняя индексы. Теперь для каждого элемента правой половины узнаем, сколько в левой половине больших его. А для элемента левой, сколько меньших его в правой.

Рассмотрим процесс `merge`. Пусть сейчас левый указатель i , правый j , слева n_l элементов.

Если сейчас выписываем $r[j]$, то слева ровно $(n_l - i)$ элементов $> r[j]$.

Если сейчас выписываем $l[i]$, то справа ровно j элементов $< l[i]$.

По сути, надо просто добавить две строки в код `MergeSort`.

2. Минимум в offline

Отрезки, не пересекающие середину, обрабатываем в рекурсивных запусках от $[0, n/2)$, $(n/2, n)$.

Для обработки отрезков, пересекающих середину, за $\mathcal{O}(n + m)$, считаем минимумы на всех отрезках вида $[i, n/2]$, $[n/2, j]$. Тогда ответ $\min(\min([l_i, n/2]), \min([n/2, r_i]))$.

Время $\Theta((n + m) \log n)$.

Можно ускорить до $\mathcal{O}(m + n \log n)$: заранее про каждый отрезок понять, на каком уровне он нужен. Уровень определяется длиной максимального общего префикса l_i, r_i как двоичных чисел, то есть старшей единичкой $l_i \hat{=} r_i$. Ее можно предсчитать для всех чисел из $[0, 2n]$.

3. Ближайшие точки

Разобьем множество точек вертикальной прямой $x = m$ на равные множества L и R . Находим ответ в обеих половинах. Итоговый ответ – минимум из двух половин, а также может быть пара близких точек разных половин.

Пусть минимум из рекурсии d . Переберем все точки (x_i, y_i) из L , и для каждой поищем точки из R в прямоугольнике $[x_i, y_i - d] \times [x_i + d, y_i + d]$. Выберем минимум из таких пар.

Как перебрать это за $\mathcal{O}(n)$? Оставим в R только точки из полосы $[m, m + d]$. Утверждается, что теперь в любом прямоугольнике $[x_i, y_i - d] \times [x_i + d, y_i + d]$ осталось не более 8 точек, иначе среди них была бы пара ближе d .

Тогда остается пройти двумя указателями по L и R и смотреть в R на 8 ближайших по y к каждой точке из L .

Для этого еще надо, чтобы точки были отсортированы по y , а для разбиения по вертикали нужна медиана по x . Просто отсортируем точки в начале один раз одну копию по x , другую по y , при разбиении будем делить эти копии и передавать нужные точки в рекурсию.

Время $T(n) = \mathcal{O}(n) + 2T(\frac{n}{2}) \Rightarrow T(n) = \Theta(n \log n)$.

4. Inplace алгоритмы

a) **Unique**. Сначала сортируем inplace, например, `HeapSort`. Затем:

```
1 int size = 1;
2 for (int i = 1; i < n; ++i)
3     if (a[size - 1] != a[i])
4         a[size++] = a[i];
```

- b) **Reverse.** `for i=0..n/2 do swap(a[i], a[n-i-1]);`
- c) **Rotate.** `rotate(a, a+k, a+n) = reverse(a, a+k) + reverse(a+k, a+n) + reverse(a, a+n)`. Заметим, что теперь умеем перетащить кусок массива из одного места в другое. За $\mathcal{O}(n)$ inplace.

Способ #2. Для всех $i \in [0, \gcd(n, k))$ делаем сдвиг на один индекс $i, (i+k) \bmod n, (i+2k) \bmod n, \dots$, пока они не заикляются.

Способ #3. Быстрый из C++.

- d) **Stable inplace merge.**

Левая часть – a , правая – b . Пусть $|a| \geq |b|$, x – средний элемент a .

a разбивается на части « $< x$ », « $= x$ », « $> x$ ». Назовем их a_1, a_2, a_3 . Разобьем и b на b_1, b_2, b_3 по тому же x . Сделаем парочку `rotate`, получим $a_1 b_1 a_2 b_2 a_3 b_3$.

Рекурсивно делаем `merge(a1, b1)`, `merge(a3, b3)`.

4.2. Кучи

1. d -куча

Можно хранить в массиве аналогично бинарной: корень = 0,

дети вершины $i - d \cdot i + 1, d \cdot i + 2, \dots, d \cdot i + d$, отец вершины $i - \lfloor \frac{i-1}{d} \rfloor$.

`extractMin` – это `siftDown`, на каждом шаге берём минимального из d детей, $\Theta(d \cdot \log_d n)$.

`decreaseKey` – это `siftUp`, $\Theta(\log_d n)$.

Итого, если нужно сделать n `extractMin` и m `decreaseKey`, время $\Theta(nd \log_d n + m \log_d n) = \Theta(\max(nd \log_d n, m \log_d n))$. С ростом d , первое слагаемое растёт, второе убывает, значит, максимум минимален при $nd \log_d n = m \log_d n \Rightarrow d = \frac{m}{n}$, время $m \log_{m/n} n$.

2. Повторения в van Emde Boas tree

- a) `lowerBound(x)`

Мы знаем i , номер куска, где лежит x : $i = \lfloor \frac{i}{\text{sqrt}C} \rfloor$. Смотрим на максимальный элемент в i -м куске: `heap[i] -> max`. Если он больше x , ищем `lowerBound` в `heap[i]`, иначе возвращаем минимальный элемент в следующем непустом куске: `notEmpty -> lowerBound(i+1) -> min`.

- b) (*) **Мелкие хеш-таблицы**

Универсальное решение для случая, когда нужно хранить массив хеш-таблиц `hashTable<int> tables[n]`: храним одну большую `hashTable<pair<int, int>> T`.

Вместо `tables[i][x]` теперь будем обращаться `T[pair(i, x)]`.

- c) (*) **Повторяющиеся элементы**

Пока длина чисел > 1 , повторения ни на что не влияют.

Когда длина чисел 1, можно хранить счётчики нулей и единиц (если в двоичной системе).

3. Левацкий бамбук

Добавляем числа в порядке убывания. Происходит подвешивание бамбука к новой вершине справа, затем `Swap` детей, куча остается растущим влево бамбуком.

4. Операции над Skew Heap

- a) Для удаления делаем `Merge` детей и подвешиваем их к предку удаленной вершины.

Время работы. С потенциалом в новом поддереве все хорошо.

Мог измениться потенциал у каких-то вершин выше. Как сверху какой-то правый сын мог

из лёгкого стать тяжёлым? Значит, мы поднимаемся по легкому ребру из левого сына. Лёгких ребер на пути до корня не более $\log n$.

b) Мы умеем делать `DecreaseKey` в любой куче через удаление и вставку.

Почему нельзя проще? Заметим, что сделать `SiftUp` не получится, даже если при проходе вверх всегда делать `Swap`: мы могли много раз пройти по тяжелому ребру из левого ребенка, тогда потенциал сильно увеличится, что ухудшит амортизационную оценку. В `Merge` это работало потому, что мы всегда шли только вправо.

c) `DecreaseKey` через `Delete` уже умеем.

`Delete(v) = DecreaseKey(v, $-\infty$) + ExtractMin()`.

5. Куча с медианой

a) Храним две кучи – `max`-кучу на минимальных $n/2$ элементах, `min`-кучу на остальных.

При добавлении понимаем, куда класть, глядя на корень первой кучи. Если нарушились размеры, перекидываем корень одной кучи в другую.

b) Можно хранить три структуры с обратными ссылками, для медианы, для минимума и для максимума.

c) А можно хранить две `MinMax`-кучи размера $(n + 1)/2$.

4.3. Дополнительные задачи

1. Подматрица с максимальной медианой

Бинпоиск по ответу. Внутри нужно проверить, есть ли подматрица с медианой $\geq x$.

Все элементы исходной матрицы заменим на 1, если $\leq x$, и 0, если нет.

В каждой подматрице посчитаем за $\mathcal{O}(1)$ число единиц (частичные суммы).

Время $\mathcal{O}(N^2 \log M)$. Чтобы получить $\log N$, будем искать бинпоиском по ответу не среди чисел от 1 до M , а среди N^2 отсортированных чисел исходной матрицы.

2. Сумма векторов

$a + b = a - (-b)$. Т.е. нужна минимальная разность векторов, это расстояние между точками.

3. Сумма векторов #2

Пусть все вектора имеют нулевой y . Пусть можно получить в сумме 0. Тогда наш алгоритм должен уметь расставлять числам знаки «плюс» и «минус», чтобы получился 0. Тогда он умеет решать задачу о рюкзаке.

4. Медиана без памяти

Если найти n минимальных элементов массива, то медиана – максимальный из них. Будем хранить n элементов в `max`-куче. Каждый раз, если $x_i < h.top()$, то `h.pop()`, `h.push(x_i)`, иначе ничего делать не надо.

5. Статистика на суммах подмножеств

Сперва избавимся от отрицательных элементов. Они все входят в минимальное множество, дальше выкидывание $a_i < 0$ равносильно добавлению $-a_i > 0$. Т.е. a_i можно заменить $-a_i$, а в конце прибавить к ответу сумму исходных отрицательных. Теперь решим для неотрицательных элементов.

Бинпоиск по ответу. Внутри нам нужно проверить, есть ли хотя бы k множеств суммы не более x . Проверим это за $\mathcal{O}(k)$ рекурсивной функцией

```
1 int count = 0;
2 void go(int i, int64_t x)
3     count++; // взяли текущее множество
4     if (count >= k) return;
5     go(i + 1, x);
6     if (a[i] <= x) go(i + 1, x - a[i]);
```

6. Трудная куча

Экспериментальные данные:

Обозначим перестановку π , число сравнений при построении кучи $f(\pi)$.

$$\max f(\pi) - \min f(\pi) \approx n$$

$$\max f(\pi) - f(n \ n-1 \ n-2 \ \dots \ 2 \ 1) \approx \log n - \text{близкий к максимуму ответ.}$$

Структуру максимума мы не понимаем.

Домашнее задание

5.1. Обязательная часть

1. (2) Хорошее множество точек

Назовём множество точек на плоскости хорошим iff \forall пары точек $(x_i, y_i), (x_j, y_j): x_i \neq x_j, y_i \neq y_j$, верно, что в прямоугольнике $[x_i, x_j] \times [y_i, y_j]$ (включая границу) лежит еще хотя бы одна точка из множества.

Примеры хороших множеств: $\{(0, 0), (0, 3)\}$, $\{(0, 0), (3, 3), (0, 3)\}$, $\{(0, 0), (3, 3), (-1, 2), (0, 2), (0, 3)\}$.

Примеры плохих: $\{(0, 0), (3, 3)\}$, $\{(0, 0), (3, 3), (1, 3)\}$, $\{(0, 0), (3, 3), (-1, 2), (0, 1)\}$.

Дано множество из n точек на плоскости. За $\mathcal{O}(n \log n)$ добавить туда $\mathcal{O}(n \log n)$ точек, чтобы оно стало хорошим. Все исходные и новые точки в совокупности должны быть различны.

2. (4) Недофибоначчи

Вы уже знаете список-кучу, которая умеет делать всё кроме `ExtractMin` за $\mathcal{O}(1)$, а `ExtractMin` за $\Theta(n)$. Оптимизируйте её, чтобы `ExtractMin` работал $\mathcal{O}(\sqrt{n})$.

Более строго: изначально есть бесконечно много пустых куч, вы знаете число n , что всего будет вызвано n операций из множества $\{\text{Add}, \text{Merge}, \text{DecreaseKey}, \text{ExtractMin}\}$, все кроме последней должны работать в худшем за $\mathcal{O}(1)$ каждая, а все вызовы `ExtractMin` должны в сумме работать за $\mathcal{O}(n^{3/2})$.

Подсказка: список элементов нужно заменить на «список групп элементов».

(2) Корректный алгоритм: `Add`, `Merge`, `ExtractMin`

(3) Корректный алгоритм: `Add`, `Merge`, `DecreaseKey`, `ExtractMin`

(4) Корректный алгоритм + доказать корректность и время работы.

(+1) если операция `ExtractMin` из кучи размера k работает амортизированно за $\mathcal{O}(\sqrt{k})$.

(+1) если получится добавить ещё и `Min` за $\mathcal{O}(1)$.

3. (2) Куча с порядковыми статистиками

Даны m чисел $0 \leq p_1, p_2, \dots, p_m < 1$.

Придумайте детерминированную структуру данных на основе `heap`, которая умеет `Insert(x)` за $\mathcal{O}(m \log n)$, `GetStatistic(p_i)` за $\mathcal{O}(1)$ и `DelStatistic(p_i)` за $\mathcal{O}(m \log n)$.

n – число элементов в структуре.

p_i – вещественное число из $(0, 1)$. `Statistic(p_i) = SortedArray[$[n \cdot p_i]$]`.

4. (3) Потоп

Есть рельеф: n столбцов высоты a_i . Уровень поверхности земли в точке i равен a_i . В результате дождя сверху вылилось T единиц осадков. Под землёй столбцы связаны узкими трубками \Rightarrow после дождя уровень поверхности воды будет одинаковым у всех столбцов.

Найти итоговый уровень воды за (2) $\mathcal{O}(n \log n)$ и (3) $\mathcal{O}(n)$.

5. (2) Длинный факториал

Посчитайте $n!$ быстрее чем за $\mathcal{O}(n^2)$.

При решении этой задачи нельзя ссылаться на ещё неизвестные нам алгоритмы.

5.2. Дополнительная часть

1. (10) Weak Heap

Рассмотрим структуру данных Weak Heap (слабая куча). Она хранится как обычная бинарная за тем исключением, что у корня есть только правый сын. Таким образом, ее можно хранить в массиве так, что у детей вершины i дети $2i$, $2i + 1$, корень в индексе 0.

Чтобы удобно менять поддеревья местами, для каждого i хранится дополнительный бит $r[i]$. Левый сын i равен $2i + r[i]$, правый $2i + 1 - r[i]$. Операция $r[i] \wedge= 1$ меняет местами детей i .

Инвариант Weak Heap: все вершины в **правом** поддереве вершины v имеют не меньший ключ, чем v . Ключи в левом поддереве могут быть произвольными, (и меньше, и больше).

Операции Add и ExtractMin, как и в бинарной куче, выражаются через SiftUp и SiftDown:
 ExtractMin это $\{-n, \text{swap}(h[0], h[n]), \text{SiftDown}(0)\}$;
 Add(x) это $\{+n, h[n-1] = x, \text{SiftUp}(n-1)\}$.

- (1) Придумайте, как найти минимум в Weak Heap за $\mathcal{O}(1)$.
- (2) Сделайте SiftUp для Add(x) за $\mathcal{O}(\log n)$.
- (1) Найти второй минимум за $\mathcal{O}(\log n)$.
- (1/2) Сделайте SiftDown для ExtractMin за $\mathcal{O}(\log^2 n)$.
- (2/2) Сделайте SiftDown для ExtractMin за $\mathcal{O}(\log n)$. Вашими SiftUp, SiftDown никто кроме Add, ExtractMin пользоваться не будет.
- (2) Придумайте, как построить по массиву Weak Heap за $\mathcal{O}(n)$ и $n - 1$ сравнение.
- (2) BulkInsert для WeakHeap. Придумать, как сделать k операций Add за $\mathcal{O}(k + \log n)$.

2. (3) Потоп #2

Есть стакан с основанием $n \times 1$ бесконечной высоты. На его дне стоят n столбцов с основанием 1×1 высоты a_i метров, плотно прилегающих друг к другу. В стакан льется ливень со скоростью один кубометр в секунду. Вода стекает с более высоких столбцов на более низкие. Вода стекает мгновенно. Если есть плато длины l , соседи которого ниже, то по $\frac{l}{2}$ польётся влево и вправо. Какой будет уровень воды в каждом из столбцов через T секунд?

Пример: $a = \{3, 1, 10, 4\}$, $T = 4$, тогда итоговый $a' = \{7, 7, 10, 10\}$.

Пример: $a = \{3, 1, 10, 4\}$, $T = 5$, тогда итоговый $a' = \{9, 9, 10, 10\}$.

3. (2) Ещё одна задача про поиск точки

Есть n точек на прямой. Точки разных типов.

Найти $x \in \mathbb{R}$ на прямой с минимальной суммой квадратов расстояний до всех типов (по каждому типу берётся ближайшая точка такого типа).