Первый курс, осенний семестр 2017/18
Практика по алгоритмам #4

Бинпоиск, два указателя

27 сентября

Собрано 29 сентября 2018 г. в 15:10

Содержание

Бинпоиск, два указателя

1. **C++.STL**
   ```
   priority_queue<int> q; q.push(1); q.top(); q.pop(); // max element
   make_heap, pop_heap, push_heap, sort_heap; // делает ровно то, что обсудили на лекции
   ```
2. **Strange bound**
   Дан x и сортированный массив. Найти $\max i\colon a[i] \leqslant x$. Используйте STL.
3. **Два указателя**
   a) Найти в данном массиве число отрезков, содержащих ровно k единиц. $\mathcal{O}(n)$.
   b) Найти отрезок максимальной длины без повторяющихся элементов. $\mathcal{O}(n)$.
   c) Найти в массиве `a` позиции `i`, `j`, `k`: `a[i] + a[j] + a[k] = S` за $\mathcal{O}(n^2)$.
4. **Поиск статистик**
   Даны два отсортированных массива длины n. Без дополнительного предподсчета найти k-ю порядковую статистику в объединении массивов.
   a) За $\mathcal{O}(\log^2 n)$.
   b) За $\mathcal{O}(\log n)$.
5. **Для любителей статистики**
   Дан массив размера n. За $\mathcal{O}(\log n)$ отвечать на запрос «сколько раз встречается число x на отрезке $[L, R]$ массива?» Можно сделать предобработку массива.
6. **d-куча**
   d-куча – куча, где у каждой вершины d детей. Ее можно хранить в массиве аналогично полной бинарной: корень – 0, дети вершины i – это $di+1, di+2, \ldots, di+d$, отец вершины i – это $\lfloor \frac{i-1}{d} \rfloor$. 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.
7. **Статистика в бинарной куче**
   Дана бинарная min-куча. Найти k-ую статистику за:
   a) $\mathcal{O}(k \log n)$
   b) $\mathcal{O}(k^2)$
   c) $\mathcal{O}(k \log k)$
8. **Генерация сортировки пар**
   Даны два массива a и b длины n, сгенерировать все попарные суммы a_i+b_j в отсортированном порядке.
   a) За $\mathcal{O}(n^2 \log n)$.
   b) За $\mathcal{O}(n^3)$ с использованием $\mathcal{O}(n)$ дополнительной памяти.
   c) За $\mathcal{O}(n^2 \log n)$ с использованием $\mathcal{O}(n)$ дополнительной памяти.
   d) За $\mathcal{O}(n^3)$ с использованием $\mathcal{O}(1)$ дополнительной памяти.
9. **Поиск периода**
   Дана последовательность чисел: $x_0 = a$, $x_{i+1} = f(x_i)$.
   За время $\mathcal{O}(L+T)$ с $\mathcal{O}(1)$ дополнительной памяти найти:
   a) длину периода T,
   b) длину предпериода L.

10. **Поиск повтора**
    Дан массив из $n+1$ целого числа от 1 до n. Массив доступен только на чтение, есть $\mathcal{O}(1)$ дополнительной памяти. Найти за $\mathcal{O}(n)$ любое число, которое встречается хотя бы два раза.

11. **K-best**
    Даны два массива из положительных чисел a и b. $|a| = |b| = n \leqslant 10^5$. Выбрать массив p: k различных чисел от 1 до n так, чтобы $\frac{\sum_{i=1}^{k} a_{p_i}}{\sum_{i=1}^{k} b_{p_i}} \to \max$.

12. **(*) Треугольники**
    Даны N различных чисел. Рассмотрим треугольники с попарно различными сторонами. Сколько существует треугольников с такими длинами сторон? $\mathcal{O}(N^2)$.

13. **(*) Среднее арифметическое**
    Найти отрезок с максимальным средним арифметическим, при этом длины от L до R.

14. **(*) Meet-in-the-middle**
    Дана последовательность $a_1, a_2, \cdots, a_n \in \mathbb{N}$ и $S \in \mathbb{N}$. Нужно найти *подмножество* этой последовательности с суммой S. Существует решение за $\mathcal{O}(n2^{n/2})$. Сложим в `set` суммы всех подмножеств первых $n/2$ чисел. Переберем все подмножества вторых $n/2$ чисел, рассматривая сумму x, ищем в `set`-е число $S-x$. Придумайте, как решить эту задачу за $\mathcal{O}(\texttt{poly}(n)2^{n/2})$ времени и $\mathcal{O}(\texttt{poly}(n)2^{n/4})$ памяти.

15. **(*) Треугольники-2**
    Даны N различных чисел. Рассмотрим треугольники с попарно различными сторонами.
    a) Найти треугольник минимальной площади с такими длинами сторон. $\mathcal{O}(N^2)$.
    b) Найти треугольник максимальной площади с такими длинами сторон. $\mathcal{O}(N^2)$.

Разбор задач практики

1. **C++.STL**
   Пример с `prioryiy_queue`, пример с `make_heap`.

2. **Strange bound**
   `upper_bound(x) - 1` – последний $\leqslant x$.

3. **Два указателя**
   a) **Число отрезков, содержащих ровно k единиц.**

```
int l1 = -1, l2 = -1, r = -1;
for (int cnt = 0; cnt < k && r < n; cnt += a[++r])
  ;
for (; r < n; ++r) {
  if (a[r] == 1) {
    l1 = l2;
    for (l2++; a[l2] == 0; l2++)
      ;
  }
  ans += l2 - l1;
}
```

   Инварианты между итерациями: `a[l1] == a[l2] == a[r] == 1`, на отрезке `[l2, r]` ровно k единиц, на отрезке `[l1, r]` ровно $k+1$.

   Способ #2.

```
int c1 = 0, c2 = 0, l1 = 0, l2 = 0;
for (int r = 0; r < n; ++r) {
  if (a[r] == 1) ++c1, ++c2;
  while (c1 > k) if (a[l1++] == 1) --c1;
  while (c2 >= k) if (a[l2++] == 1) --c2;
  ans += l2 - l1;
}
```

   Способ #3.

```
int pos = -1, ans = 0;
vector<int> d;
for (int i = 0; i <= n; i++)
  if (i == n || a[i] == 1)
    d.push_back(i - pos), pos = i;
for (size_t j = k; j <= d.size(); j++)
  ans += d[j - k] * d[j];
```

   b) **Отрезок максимальной длины без повторяющихся элементов.**

```
int l = 0;
for (int r = 0; r < n; ++r) {
  while (used[a[r]])
    used[a[l++]] = false;
  used[a[r]] = true;
  ans = max(ans, r - l + 1);
}
```

При больших числах в качестве `used` используется хеш-таблица.

Способ #2.
```
int l = 0;
for (int r = 0; r < n; ++r) {
  if (last[a[r]] >= l) l = last[a[r]] + 1;
  last[a[r]] = r;
  ans = max(ans, r - l + 1);
}
```

c) **3-SUM.** Сортируем `a`. Перебираем `k`, ищем за $\mathcal{O}(n)$ `a[i] + a[j] = S1 = S - a[k]`.
```
int j = n - 1;
for (int i = 0; i < n; ++i) {
  while (a[i] + a[j] > s1) --j;
  if (a[i] + a[j] == s1) ok;
}
```

4. **Поиск статистик**

   a) $\mathcal{O}(\log^2 n)$: бинпоиск по ответу, внутри бинпоиск по массиву.
   Пусть ответ в массиве a. Бинпоиск по a, внутри бинпоиска надо узнать, каким по порядку в объединении является элемент $a[m]$. В a есть m чисел $\leqslant a[m]$, во втором массиве ищем число $\leqslant a[m]$ вложенным бинпоиском (нужен `upper_bound`).
   Если не нашли ответ, ищем его во втором массиве так же.

   b) $\mathcal{O}(\log n)$. $a[i] \leqslant b[k-i] \Rightarrow a[i]$ не больше k статистики, иначе больше. Ищем бинпоиском $\max i \colon a[i] \leqslant b[k-i]$.

5. **Для любителей статистики**
   Отсортируем пары `(a[i], i)`. Ответ: `upper_bound(x, R) - lower_bound(x, L)`.

6. **d-куча**
   Операций $\Theta(nd\log_d n + m\log_d n) = \Theta(\max(nd\log_d n, m\log_d n))$. Первое слагаемое растет с ростом d, второе убывает, значит, минимальный максимум при $nd\log_d n = m\log_d n \Rightarrow d = \frac{m}{n}$.

7. **Статистика в бинарной куче**

   a) $\mathcal{O}(k\log n)$. k раз вынимаем минимум.

   c) $\mathcal{O}(k\log k)$. Минимум в корне. Следующий в одном из его детей. Где третий? Либо в детях меньшего сына, либо в большем сыне. Итого у нас на шаге i есть i кандидатов, выбираем минимального и делаем кандидатами его детей. Храним кандидатов в (другой) куче.

8. **Генерация сортировки пар**

   a) За $\mathcal{O}(n^2\log n)$: просто сгенерируем все пары и отсортируем.

   c) За $\mathcal{O}(n^2\log n)$ с $\mathcal{O}(n)$ дополнительной памяти.
   Отсортируем B. Для каждого a_i мы выведем суммы в порядке $a_i + b_1, a_1 + b_2 \ldots$
   Создаем кучу из всех $a_i + b_1$. n^2 раз делаем `extractMin`. Когда вывели в ответ $a_i + b_j$, кладем в кучу $a_i + b_{j+1}$.

   d) За $\mathcal{O}(n^3)$ с $\mathcal{O}(1)$ дополнительной памяти: отсортируем оба массива. После того, как вывели в ответ x, двумя указателями за $\mathcal{O}(n)$ найдем минимальную сумму $> x$, и сколько раз она встречается.

9. **Поиск периода**
   Tortoise-and-hare алгоритм Флойда.

- `x = x0, y =` f`(x0); while (x != y) x =` f`(x), y =` f`(`f`(y));`
  Точка, в которой мы остановились, лежит на цикле. Обозначим ее `a`. Каждый шаг расстояние между x и y увеличивается на один, за $\leqslant L+T$ шагов найдем.
- `x =` f`(a), T = 1; while (x != a) x =` f`(x), ++T;`

Способ #2: алгоритм Брента.
Пытаемся угадать $L+T$, ищем такое k, что $2^{k-1} \leqslant L+T \leqslant 2^k$.
`x =` $f^{(2^k)}(x_0)$, перебираем `y` от f`(x)` не более 2^k шагов вперёд.
Время работы $1+2+4+\cdots+2^k = \mathcal{O}(L+T)$.

Поиск предпериода: `x = x0, y =` $f^{(T)}$`(x0); while (x != y) x =` f`(x), y =` f`(y);`

10. **Поиск повтора**
    Последовательность $x_1 = n+1$, $x_{i+1} = a[x_i]$ периодична и имеет ненулевой предпериод (ни один элемент не равен $n+1$). Нам нужно начало периода, для этого достаточно узнать длину предпериода, воспользуемся предыдущей задачей.

11. **K-best**
    Бинпоиск по ответу. Проверяем, что можно выбрать $\frac{\sum_{i=1}^k a_{p_i}}{\sum_{i=1}^k b_{p_i}} \geqslant m$. Преобразуем к виду $\sum_{i=1}^k (a_{p_i} - m \cdot b_{p_i}) \geqslant 0$. Жадно выберем k таких пар, которые дают максимальные слагаемые, проверим что сумма хотя бы 0.

12. **(*) Треугольники**
    Сортируем стороны. Перебираем сторону `x[i]`. Другие две двумя указателями: первый `j` от `i` по возрастанию, второй находит максимальное `k: x[k] <= x[i] + x[j]`. `ans += (k - j)`.

13. **(*) Среднее арифметическое**
    Бинпоиск по ответу. Проверяем, что $\exists r, l \in [r-R, r-L]$: $\frac{\mathtt{pref}[r+1]-\mathtt{pref}[l]}{r-l} \geqslant M \Leftrightarrow \mathtt{pref}[r+1] - Mr \geqslant \mathtt{pref}[l] - Ml$. Перебираем r, храним очередь с минимумом для $\mathtt{pref}[l] - Ml$.

14. **(*) Meet-in-the-middle**
    Рассмотрим другое решение за $\mathcal{O}(n2^{n/2})$ времени и памяти: сгенерируем два массива a и b, в одном суммы всех подмножеств первых $n/2$ чисел, в другом – вторых. Сортируем их и двумя указателями ищем элементы двух массивов, дающие сумму S.
    Теперь вместо сортированного массива a длины $2^{n/2}$ рассматриваем массивы a_1, a_2 длины $2^{n/4}$: суммы подмножеств первых и вторых $n/4$ чисел.
    Используем задачу про генерацию сортированных по сумме пар за $\mathcal{O}(n^2 \log n)$ с $\mathcal{O}(n)$ памяти.
    Передвижение указателя по a заменяется на генерацию очередной пары из массивов a_1, a_2.
    Время $(2^{n/4})^2 \log(2^{n/4}) = \Theta(n2^{n/2})$, память $\Theta(2^{n/4})$.

15. **(*) Треугольники-2**
    Считаем, что стороны отсортированы и $a < b < c$.
    a) **Минимальная площадь.** При фиксированных a и c высота, опущенная на сторону c, тем меньше, чем ближе $a+b$ к c. Тогда при фиксированных a, b нам нужно $\max c \leqslant a+b$. Перебираем a, внутри b и c двумя указателями.
    b) **Максимальная площадь.** По рассуждению выше, можно перебрать a и b, а в качестве c брать следующую за b сторону. Можно показать, что на самом деле ответ – три последовательных элемента массива.

Домашнее задание

3.1. Обязательная часть

1. **(2) Параллельный минимум и максимум**
   Дан массив из $2n$ чисел. Найти минимальное **И** максимальное за $3n - 2$ сравнения.

2. **Поиск статистики**
   Даны m сортированных массивов длины n. Нужно без дополнительного предподсчета найти k-ю порядковую статистику. Числа от 0 до `MAX`.
   a) **(2)** За $\mathcal{O}(m + k \log m)$.
   b) **(2)** За $\mathcal{O}(m \log \mathtt{MAX} \log n)$.
   c) **(+1)** допбалл: вероятностное $\mathcal{O}(m \log^2 poly(n, m))$.

3. **(1) Ближайший по значению**
   Даны отсортированные массивы a и b длины n.
   Для каждого элемента a найти ближайший по значению элемент b. $\mathcal{O}(n)$.

4. **В этом задании требуется прислать код на языке `C/C++`!** [1]
   Множество и мультимножество можно хранить в виде отсортированного массива. Даны два множества A и B в отсортированном виде, за $\mathcal{O}(|A| + |B|)$ построить в таком же виде их
   a) **(1)** множество-разность. Пример: $\{1, 2, 3\} \setminus \{2, 4\} = \{1, 3\}$.
   b) **(1)** множество-объединение. Пример: $\{1, 2, 3\} \cup \{2, 4\} = \{1, 2, 3, 4\}$.

5. **(2) Ближайший по координате**
   Даны $n \leqslant 10^6$ точек на плоскости, координаты целые до 10^9 по модулю. Приходят $q \leqslant 10^6$ запросов: дана точка p, найти для нее ближайшие по X слева и справа точки с таким же Y и ближайшие по Y точки с таким же X сверху и снизу. Запросы online, нужно на каждый запрос отвечать сразу, а не на все вместе в конце.

6. **(2) Отложенные операции**
   Рассмотрим структуру данных, хранящую мультимножество целых чисел S и умеющую обрабатывать два запроса:

   - `count`$(S \cap [L..R])$ (количество элементов от L до R)
   - `add(x)`

   Структура устроена так: храним отсортированный массив a и массив b в произвольном порядке, при этом поддерживаем $|b| \leqslant \sqrt{|a|}$. На запросы `count` отвечаем, делая бинпоиск в a и линейный проход в b. При добавлении элемента добавляем его в конец b, если после этого $|b| > \sqrt{|a|}$, то $\{a = \mathtt{merge}(a, \mathtt{sort}(b)),\ b = \mathtt{[]}\}$.
   Найти и доказать амортизированное время обработки запросов.

[1] Можно прикрепить код, можно вставить его в tex. Код должен компилироваться и работать.

3.2. Дополнительная часть

1. **(3) Коробки с шарами**
   Дано $2 \cdot n - 1$ коробок с черными и белыми шарами. В i-ой коробке находится w_i белых и b_i черных шаров. Всего в коробках находится W белых и B черных шаров. Требуется выбрать n коробок, чтобы суммарное число белых шаров в них было не менее $\frac{W}{2}$, а черных не менее $\frac{B}{2}$. Решить за $\mathcal{O}(n \log n)$.

2. **(3) Коробки с предметами**
   Даны N предметов с весами w_i и бесконечный набор коробок размера W. Разложить предметы в минимальное число коробок при условии, что в одну коробку можно класть не более двух предметов.

3. **(3) О трудности коммуникации**
   У Алисы есть массив $a_1, a_2, \ldots, a_{k_1}$, у Боба есть $b_1, b_2, \ldots, b_{k_2}$.
   В каждом массиве числа от 1 до n.
   В каждом массиве числа различны, но может быть одно и то же число в обоих массивах.
   Алиса и Боб знают n, но не знают ничего про чужой массив.
   Они хотят найти медиану объединения своих массивов, то есть $\lfloor (k_1 + k_2)/2 \rfloor$ элемент отсортированного объединения. Алиса и Боб могут общаться! Придумайте стратегию, по которой они найдут медиану, если они могут переслать другу другу суммарно $\mathcal{O}(\log n)$ **бит**.
   За $\mathcal{O}(\log^2 n)$ **бит** можно получить **(2)** балла.