SPb HSE, 1 курс, весна 2024/25
Практика по алгоритмам #28

Дерево отрезков

14 мая

Собрано 14 мая 2025 г. в 09:35

Содержание

Дерево отрезков

1. **Пары начинаются**

   Как с помощью шаблонного (`template`) дерева отрезков, которое умеет возвращать только min/max, находить ещё и позицию минимума? Внутренность дерева менять нельзя.

2. **Спуск!**

   Массив из нулей и единичек. Отвечать на запросы $a_i = x$; найти позицию k-й единицы. $\mathcal{O}(\log n)$, дерево отрезков.

3. **Какая операция подходит для дерева отрезков**

   a) Произведение матриц, gcd, композиция перестановок длины k на отрезке.
   b) Покраска отрезка в чёрный и белый, число отрезков белого цвета на отрезке.

4. **BST → дерево отрезков**

   a) Хотим поддерживать множество (не мульти) из чисел от 1 до $n \leqslant 10^6$. Числа добавляются, удаляются. Нужно отвечать на запрос «количество элементов в множестве от L до R).
   b) Хотим поддерживать мультимножество из чисел от 1 до $n \leqslant 10^9$. Запросы те же.

5. **Максимальная по весу возрастающая подпоследовательность**

   a) Найти НВП максимальную по сумме элементов.
   b) Даны пары $\langle x_i, y_i \rangle$, у каждой пары есть вес w_i. Найти последовательность точек (не подпоследовательность; возрастание по i не требуется), строго возрастающую по x_i **и** по y_i, с максимальной суммой весов w_i. $\mathcal{O}(n \log n)$.

6. **k-инверсии**

   k-инверсия в перестановке p – это набор индексов $i_1 < i_2 < \ldots < i_k \colon p[i_1] > p[i_2] > \ldots > p[i_k]$. Найти число k-инверсий за $\mathcal{O}(nk \log n)$.

7. **Отрезки на прямой**

   Даны n отрезков на прямой. За $\mathcal{O}(n \log n)$ посчитать число пар вложенных отрезков.

♥ 8. **Площадь объединения прямоугольников**

   Площадь объединения n прямоугольников с большими координатами за $\mathcal{O}(n \log n)$.

9. **Точки и прямоугольники 3**

   Даны точки и прямоугольники. Найти кол-во точек, покрытых ровно k прямоугольниками.

10. **Точки и прямоугольники 4**

    Даны прямоугольники. Найти кол-во целых точек плоскости, покрытых ровно k прямоугольниками. Могут быть совпадающие прямоугольники.

11. **Все числа $\geqslant X$ на отрезке**

    Вывести все числа $\geqslant X$ на отрезке $[L, R]$ массива. Что хранить в вершине дерева отрезков, чтобы отвечать на запрос за $\mathcal{O}(k + \log n)$, где k – размер ответа.

12. **Отрезки, которые раньше никого не покрывали**

    Даны n отрезков на прямой. Запросы: по данной точке вывести все отрезки, покрывающие её, но еще не выведенные ранее (т.е, каждый отрезок выводится не более одного раза). Суммарное время $\mathcal{O}((m + n) \log n)$.

13. **Ближайший ноль**

    Есть массив из нулей и единиц. В online за $\mathcal{O}(\log n)$ отвечать на запросы:
    поменять элемент; найти ближайший слева/справа ноль к позиции i. `(*)` $\mathcal{O}(\log \log n)$.

14. `(*)` **Разложение отрезка на вершины ДО**

    Есть дерево отрезков над массивом длины 2^k. На какие вершины дерева разбивается $[0, R)$?

15. `(*)` **Для любителей статистики :)**

    Сколько раз встречается число x на отрезке $[L, R]$. Online, static, $\mathcal{O}(\log n)$.

16. `(*)` `+=` **из ниоткуда**

    Пусть у нас уже есть дерево отрезков, умеющее изменение в точке и сумму на отрезке. Выразить через несколько таких структур более мощную, умеющую еще и «`+=` на отрезке».

17. `(*)` **Прибавление линейной функции на отрезок**

    Запросы: сумма на отрезке, прибавить $ai + b$ к i-у элементу отрезка $[L, R]$ для всех i.
    То есть, сделать `array[L + i] += a*i + b`.

18. `(*)` **Количество различных чисел на отрезке**

    Количество различных чисел на отрезке $[L, R]$. Static. *Подсказка про* $\texttt{prev}_i$.
    Offline за $\mathcal{O}(\log^2 n)$. Offline за $\mathcal{O}(\log n)$. А теперь online за $\mathcal{O}(\log n)$.

19. `(*)` **Динамическая k-я статистика на отрезке**

    Решить k-ю статистику на отрезке для случая, когда массив может меняться.

20. `(*)` **Еноты с загребущими лапками**

    На плоскости есть множество енотов $R, |R| = n$, и множество ягод $B, |B| = m$.
    Для каждой ягоды найти ближайшего енота по манхэттенской метрике ($|x_1 - x_2| + |y_1 - y_2|$), среди ближайших min по индексу.
    $\mathcal{O}((n + m) \log^2 n)$ времени, $\mathcal{O}(n + m)$ памяти.

21. `(*)` gcd **в промежутке значений**

    Дан массив натуральных чисел.
    Нужно находить gcd всех чисел, значения которых в промежутке $[X, Y]$.

    a) Массив не меняется. $\mathcal{O}(\log n + \log A)$.
    b) А потом меняется (в точке)! $\mathcal{O}(\log n + \log A)$.
    c) Массив не меняется, нужно брать числа $L \leqslant i \leqslant R, X \leqslant a_i \leqslant Y$. $\mathcal{O}(\log^2 ?)$.

22. `(*)` **k-я статистика различных чисел на отрезке**

    Запросы: k-е по порядку среди различных чисел на отрезке $[L, R]$. Static.

Разбор задач практики

1. **Пары начинаются**

   Хранить пары $\langle a_i, i\rangle$. `RangeTree<pair<int,int>>`.

2. **Спуск**

   Дерево отрезков на сумму. Чтобы ответить на запрос позиции, спускаемся по дереву сверху вниз. Каждый раз идём или налево, или направо, в зависимости от суммы в левом сыне.

3. **Какая операция подходит для дерева отрезков**

   a) Любая ассоциативная и аддитивная операция легко делается.

   b) *Покраска в чёрный и белый, число отрезков белого цвета.*
   `t[v].cnt = t[2v].cnt + t[2v+1].cnt- (t[2v].right == W && t[2v+1].left == W)`
   `t[v].right = t[2 * v + 1].right, t[v].left = t[2 * v].left`
   Покраска – отложенные операции. При `push` легко пересчитать `cnt`, `left`, `right`.

4. **BST → дерево отрезков**

   a) Дерево отрезков на массиве нулей и единиц `isin[x]` – есть ли элемент, равный x.

   b) Дерево отрезков на массиве `count[x]` – число элементов, равных x. Значения большие, поэтому динамическое ДО.

5. **Максимальная по весу возрастающая подпоследовательность**

   a) DP с лекции. $f[a_j] = dp_j$, и $dp_i = a_i +$ max массива f на отрезке $[0, a_i)$.

   b) $f[i]$ – max сумма w_j последовательности, кончающейся на i. $f[i] = w_i + \max\limits_{x_j<x_i \wedge y_j<y_i} f[j]$

   Это можно считать деревом отрезков по x деревьев отрезков по y за $\mathcal{O}(\log^2 n)$.
   Чтобы посчитать $f[i]$ надо, чтобы все нужные $f[j]$ были посчитаны. Будем обходить точки в порядке сортировки по парам (x_i, y_i)

   За $\mathcal{O}(n \log n)$ scanline по возрастанию x_i (при равенстве по убыванию y_i). Когда хотим посчитать $f[i]$, посчитаны все $f[j]\colon x_j < x_i \vee (x_j = x_i \wedge y_j > y_i)$. Среди них нужно $\max f[j]\colon y_j < y_i$.
   Дерево отрезков на координатах y, считающее $\max f$. Запрос на префикс $[0, y_i)$.
   Когда посчитали $f[i]$, обновляем значение в точке y_i.

6. **k-инверсии**

   $f_{k,i}$ – число k-инверсий, заканчивающихся в i. $f_{1,i} = 1$, $f_{k+1,i} = \sum\limits_{j<i, a_j>a_i} f_{k,j}$.

   Scanline по i (гарантирует $j < i$) с деревом отрезков на сумму на координатах a_i.
   $f_{k+1,i}$ – запрос на отрезок $(a_i, n]$. Затем прибавить $f_{k,i}$ в точку a_i.

7. **Отрезки на прямой**

   Отрезок $[l_i, r_i]$ вложен в $[l_j, r_j] \Leftrightarrow l_j \leqslant l_i \wedge r_j \geqslant r_i$.
   Скажем, что (l_i, r_i) – точка на плоскости и получим стандартную задачу на scanline, в решении которой мы умеем даже больше: для каждого i найти число таких j.

8. **Площадь объединения прямоугольников**

   Scanline по x. События стандартные для работы с прямоугольниками.

   - Прямоугольник начался ⇒ `cnt[] += 1` на отрезке y.
   - Прямоугольник кончился ⇒ `cnt[] += -1` на отрезке y.

Когда переходим $x_i \to x_{i+1}$, прибавим к ответу $(x_{i+1} - x_i)$, умноженное на вертикальную длину покрытой прямоугольниками части плоскости. Для этого нужно посчитать число y: `cnt`$[y] > 0$. Это само по себе очень сложно. Но в нашем случае `cnt`$[y] \geqslant 0 \Rightarrow$ можно посчитать число нулей, равное числу минимумов. Если y большие, то в листе нужно записывать $(y_{i+1} - y_i)$, то есть в листе не один минимум, а $(y_{i+1} - y_i)$.

9. **Точки и прямоугольники 3**

   Scanline по x.

   - Прямоугольник начался $\Rightarrow$ `+=1` на отрезке y.
   - Встретили точку $(x_i, y_i) \Rightarrow$ `get(`y_i`) == k`.
   - Прямоугольник кончился $\Rightarrow$ `-=1` на отрезке y.

   При равных x события обрабатываем именно в таком порядке.
   Дерево отрекзов отлично умеет все три запроса.

10. **Точки и прямоугольники** 4

    Scanline по x.

    - Прямоугольник начался $\Rightarrow$ `+=1` на отрезке y.
    - Прямоугольник кончился $\Rightarrow$ `-=1` на отрезке y.

    После каждого запроса проверяем, есть ли число `=k` на отрезке. Нужна структура данных, которая умеет обрабатывать все эти 3 запроса. Такая у нас есть. Корневая.

11. **Все числа $\geqslant X$ на отрезке**

    Дерево отрезков сортированных массивов. Разбиваем $[L, R]$ на $\mathcal{O}(\log n)$ вершин дерева отрезков, в каждой за $\mathcal{O}(k_i)$ выводим все числа $\geqslant X$ – суффикс сортированного массива.

12. **Отрезки, которые раньше никогда никого не покрывали**

    Каждый отрезок разобьём на $\mathcal{O}(\log n)$ вершин дерева отрезков.
    У каждой вершины есть вектор индексов отрезков, добавим туда наш отрезок.
    Когда приходит точка X, берём все $\mathcal{O}(\log n)$ вершин, содержащие её, помечаем их.
    Если среди них были не помеченные, переберём индексы отрезков из векторов этих вершин.
    Если встречаем индекс непомеченного отрезка, помечаем его и выводим.
    В помеченных вершинах не будем заново ходить по вектору $\Rightarrow$ каждый из m отрезков мы просмотрим $\mathcal{O}(\log n)$ раз $\Rightarrow \mathcal{O}((n + m) \log n)$.

13. **Ближайший ноль**

    Короткий разбор:
    Ищем ближайший справа в множестве. По сути `lowerBound`.
    Идея – дерево отрезков с операцией «число нулей».

    Способ #0. `set<int>` подходит! А если его сделать VanEmbdeBoas-ом, то и за $\mathcal{O}(\log \log n)$ начнёт работать. `VanEmbdeBoas.lowerbound(x)`: $i = x/\sqrt{n}$, если i-я ячейка не пуста и max в ней больше x, обращаемся туда, если нет, вызываем lowerbound на «непустых ячейках» и в найденной возвращаем минимальный элемент.

    Способ #1. Считаем k – кол-во нулей на $[0, i)$, ищем $(k+1)$-й ноль.

Способ #2. Запрос снизу.

```
int get(int i):
for (i += n; cnt[i] == 0; (i & 1) ? i++ : i /= 2); // i -> правый брат или отец
while (i < n) // в поддереве точно есть ноль, осталось спуститься
    i = (cnt[2 * i] ? 2 * i : 2 * i + 1);
return i - n;
```

Этот способ можно разогнать до $\mathcal{O}(\log \log n)$. Давайте не ходить в братьев, а всегда идти в отца. Биноиском находим ближайшего предка, в котором есть нули. В вершине храним самый левый и самый правый ноль. Один из них – ближайший к нам с какой-то из сторон. Мы могли попасть в ближайшего не с той стороны, например, слева вместо справа. Храним нули в двухсвязном списке, тогда сосед найденного нуля подходит.

Способ #2b, снизу.
Короткий за $\mathcal{O}(\log \log n)$. Если мы хотим «ближайший справа ноль», можно биноиском сразу переходить в предка, у которого «крайний правый ноль правее i», из него идти в его правого сына и от него за $\mathcal{O}(1)$ брать крайний левый ноль.

Способ #3, сверху.
Храним в вершине «крайний правый ноль», спускаемся, всегда знаем, куда.

Способ #4, сверху.
Дерево отрезков разбивает отрезок $[i, n)$ на $\log n$ вершин. В самой левой из них, содержащей ноль, спустимся в самый левый ноль. Ниже код, который одним запросом сверху и разбивает $[i, n)$ на вершины ДО и в крайней левой, содержащей ноль, спускается в левый ноль.

```
int get(int v, int vl, int vr, int i): // v = [vl, vr)
if (vr <= i || cnt[v] == 0)
    return -1;
if (vl + 1 == vr) return vl;
int vm = (vl + vr) / 2, res = get(2 * v, vl, vm, i);
return res != -1 ? res : get(2 * v + 1, vm, vr, i);
```

14. **(*) Разложение отрезка на вершины ДО**

Запрос $[0, R)$. Рассмотрим все единичные биты R.
Биту 2^i соответствует вершина дерева отрезков на i-м уровне (у корня уровень k).
Номер вершины (считая с 1), выбранной на i уровне, равен $(R \texttt{ >> } i)$.
Его индекс в массиве $(2^{k-i} - 1) + (R \texttt{ >> } i)$ (индекс корня 1).

15. **(*) Для любителей статистики :)**

Задача-шутка. Была в прошлом семестре. Предподсчёт – сортировка пар (a_i, i).
Запрос – `upper_bound` пары (x, r) − `lower_bound` пары (x, l). $\mathcal{O}(\log n)$.

16. **(*) += из ниоткуда**

Операция на отрезке сводится к двум операциям на префиксах.
Пусть нам нужна сумма на $[0, r)$.
Операции прибавления x_1 на $[0, r_1)$: $r_1 \geqslant r$ внесут в эту сумму rx_1.
Операции прибавления x_2 на $[0, r_2)$: $r_2 < r$ внесут в эту сумму $r_2 x_2$.
Отсюда алгоритм. T_1 и T_2 – обычные ДО с изменением в точке и суммой на отрезке.
`add(0, i, x)` – это T_1`.add(i, x);` T_2`.add(i, i * x)`.
`get(0, i) = i *` T_1`.get(i,` $+\infty$`) +` T_2`.get(0, i)`.

17. (*) **Прибавление линейной функции на отрезок**

Аналогично задаче «+= через =».
Прибавление $ai + b$ на $[L, R]$ – это прибавление $ai + b - aL$ на $[0, R]$, $-ai - b + aL$ на $[0, L)$.
Достаточно уметь прибавлять на префикс.
Пусть нам нужна сумма на $[0, r)$.
Операции прибавления $x_1 i$ на $[0, r_1)$: $r_1 \geqslant r$ внесут в эту сумму $\frac{r(r+1)}{2} x_1$.
Операции прибавления $x_2 i$ на $[0, r_2)$: $r_2 < r$ внесут в эту сумму $\frac{r_2(r_2+1)}{2} x_2$.
Храним в одном ДО все добавленные x, еще в одном $x\frac{i(i+1)}{2}$.

18. (*) **Количество различных чисел на отрезке**

$\texttt{prev}_i = \max\{j < i : a_j = a_i\}$.
$\texttt{distinct}[l, r] = |\{i \in [l, r] : \texttt{prev}_i < l\}|$.

a) 2D-дерево отрезков, $\langle \mathcal{O}(n \log n), \mathcal{O}(\log^2 n) \rangle$, online.
b) Offline. Scanline по l с деревом отрезков на сумму.
Как только $\texttt{prev}_i$ стало $< l$, ставим в i единицу.
$\langle \mathcal{O}(n), \mathcal{O}(\log n) \rangle$. Этот подход обощается на произвольные запросы в «стакане».
c) Online. Заменяем scanline на персистентное ДО. $\langle \mathcal{O}(n \log n), \mathcal{O}(\log n) \rangle$.
Другой способ: 2D-дерево отрезков на сумму с fractional cascading.

19. (*) **Динамическая k-я статистика на отрезке**

$\mathcal{O}(\log^3 n)$: дерево отрезков, в вершинах treap.
Бинпоиск по ответу, внутри в $\mathcal{O}(\log n)$ вершинах ДО узнаем, сколько там чисел $\leqslant m$.
Изменение за $\mathcal{O}(\log^2 n)$. Память $\mathcal{O}(n \log n)$.

$\mathcal{O}(\log^2 n)$: дерево отрезков, в вершинах динамические ДО.
Параллельный бинпоиск по ответу в $\mathcal{O}(\log n)$ вершинах ДО.
Изменение за $\mathcal{O}(\log^2 n)$. Память $\mathcal{O}(n \log n)$.
На самом деле во всех оценках $\log M$ от динамического ДО...

20. (*) gcd **в промежутке значений**

a) Static – просто ДО на отсортированном массиве значений.
b) Dynamic – динамическое ДО. Отсутствующие вершины дают вклад 0.
Также храним в листьях счетчики значений.
Изменение в точке – изменить нужные счетчики. Если счетчик занулился, то и значение листа зануляем.
c) gcd на прямоугольнике – ДО по индексам массива, внутри ДО на значениях. $\mathcal{O}(\log^2 n \cdot \log A$.(даже $\mathcal{O}(\log n \cdot (\log n + \log A)))$.

21. (*) **Еноты с загребущими лапками**

Повернём картинку на 45°, увидим квадраты.
Способ #1: для каждой ягоды делаем бинарный поиск по ответу, внутри запрос к дереву отрезков (по x) сортированных массивов (по y). Время $\mathcal{O}(m \log^2 n + n \log n)$, память $\mathcal{O}(n \log n)$.
Способ #2: делаем параллельный бинарный поиск по ответу, внутри scanline для того, чтобы найти минимального енота, или сказать, что его нет, в m квадратах. Время $\mathcal{O}((m+n) \log^2 n)$, память $\mathcal{O}(n + m)$.

22. (*) **k-я статистика различных чисел на отрезке**

a) Offline за $\mathcal{O}(\log^2 n)$. Scanline по L.
На тех i, у которых $\texttt{prev}_i < L$, пишем a_i, на остальных $+\infty$. При каждом L решаем задачу о статистике на отрезке обычным способом за $\mathcal{O}(\log^2 n)$.
За $\mathcal{O}(\log n)$ просто так не получится: нужна структура для *dynamic* варианта задачи о k-й статистике, массив для поиска статистики меняется при движении L.
Память такая же, как у структуры для поиска k-й статистики: $\mathcal{O}(n \log n)$.

b) Online за $\mathcal{O}(\log^3 n)$. ДО на значениях, в вершинах массив из соответствущих индексов, на котором построена структура для online подсчета числа различных на отрезке.
Бинпоиск по ответу. Внутри $\mathcal{O}(\log n)$ вершин ДО задают числа $\leqslant m$. Считаем, сколько в них на отрезке $[L, R]$ различных чисел.
Память и предподсчет $\mathcal{O}(n \log^2 n)$.

c) Online за $\mathcal{O}(\log^2 n)$. Просто замена бинпоиска на спуск по внешнему дереву.
Еще можно сделать персистентной первую версию.

d) Dynamic. Для пересчёта `prev` храним `map<int,set<int>> positions`.
Бинпоиск по ответу, внутри ответ на запрос $|\{i \in [L, R] \colon a_i \leqslant X, \texttt{prev}_i < L\}|$.
Отвечает на внутренний запрос ДО по i деревьев отрезков по `prev` treap-ов по a.
Время этого запроса $\mathcal{O}(\log^3 n)$, память $\mathcal{O}(n \log^2 n)$. В сумме с бинпоиском $\mathcal{O}(\log^4 n)$ на запрос.

Заменим внутренний treap на динамическое ДО. А бинпоиск на параллельный спуск по $\mathcal{O}(\log^2 n)$ деревьям, на которые бьется прямоугольник $\{i \in [L, R], \texttt{prev} < L\}$.
Получаем время $\mathcal{O}(\log^2 n \log M)$ на запрос и $\mathcal{O}(n \log^2 n \log M)$ памяти.

Домашнее задание

3.1. Обязательная часть

1. **(3) Произведение и присваивание на отрезке**

   Дан массив чисел. Все операции по модулю фиксированного M. Online запросы:

   - посчитать произведение всех чисел на отрезке $[L, R]$;
   - присвоить значение x всем числам на отрезке $[L, R]$.

   Убедитесь, что ваше решение работает именно за $\mathcal{O}(\log n)$.

2. **(2) Ближайший большой элемент**

   Дан массив целых чисел. В online обрабатывать запросы:

   - за $\mathcal{O}(\log n)$ по данным pos и x искать ближайший к pos элемент $\geqslant x$;
   - изменить значение i-го числа.

   (1) можно получить за решение за $\mathcal{O}(\log^2 n)$.

3. **(3) Возрастающая плавно подпоследовательность**

   Даны массив длины n и число k. Найти самую длинную возрастающую подпоследовательность, в которой все числа отличаются хотя бы на k. $\mathcal{O}(n \log n)$.

4. **(3) Правильный скобочный подотрезок**

   Дана скобочная последовательность из круглых скобок. Запросы: является ли отрезок $[L, R]$ правильной скобочной последовательностью; изменить i-ю скобку. $\mathcal{O}(\log n)$, online.

5. **(1.5+1.5) Отрезки без котиков**

   Дан набор отрезков на клетчатой полоске. Сначала в каждой клетке сидит котик.
   Запрос: из клетки i ушёл котик, сколько отрезков сейчас не содержат ни одного котика?

   - **(1.5)** Offline.
   - **(1.5)** Online.

3.2. Дополнительная часть

1. **(2) Сумма кубов**

   Дан массив чисел. За $\mathcal{O}(\log n)$ в online обрабатывать запросы:

   - посчитать сумму кубов чисел на отрезке $[L, R]$;
   - прибавить x ко всем числам на отрезке $[L, R]$;
   - получить значение i-го числа.

2. **(3) Вес точек, покрытых прямоугольником**

   Дан набор точек на плоскости с весами $w_i > 0$.
   Покрыть прямоугольником $a \times b$ со сторонами, параллельными осям координат, набор точек max веса.

3. **(4) Максимальный пустой квадрат**

   Дан набор точек на плоскости, лежащих в квадрате $[0, S] \times [0, S]$. Найти квадрат max площади, лежащий целиком в $[0, S] \times [0, S]$ и не содержащий ни одной точки.
   Можно решить на **(2)** балла за $n\text{poly}(\log n)$. Можно на **(4)** за $\mathcal{O}(n \log n)$.