Разработка и анализ алгоритмов

Лекция 13
Дерево отрезков. Разреженные таблицы

Сергей Леонидович Бабичев

Дерево отрезков

Дерево отрезков: обработка запросов сверху вниз

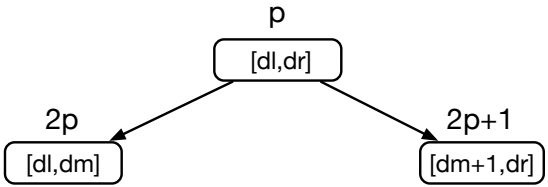
Задача 1. Имеется массив из N элементов. К нему поступают запросы двух типов:

1. Изменить значение элемента.
2. Вычислить заданную операцию на подотрезке $[l, r]$.

При изменении элемента мы поднимались по дереву, корректируя значения в родителе.
При исполнении функции мы рекурсивно поднимались вверх, исполняя операции на подотрезках меньшего размера.
Другой вариант — исполнение запросов сверху вниз.

Дерево отрезков: обработка запросов сверху вниз

- Каждый узел доминирует над группой терминальных узлов, над интервалом $[dl, dr]$.



- Для обработки отрезка надо проверить, доминирует ли текущий узел над какой-то частью отрезка.
- Если он полностью доминирует — подотрезок добавляется к результату.
- Если не полностью — разбиваем запрос на два и проверяем каждый.
- Отрезок разбивается пополам на подотрезки $[dl, dm]$ и $[dm + 1, dr]$.

Дерево отрезков: обработка запросов сверху вниз

```
int down(size_t p, size_t l, size_t r, size_t dl, size_t dr) {
    int ret = E;
    if (l <= dl && r >= dr) return OP(ret, body[p]);
    size_t bm = (dl + dr) / 2;
    if (l <= bm) ret = OP(ret, down(pos * 2, l, bm, dl, bm));
    bm++;
    if (r >= bm)  ret = OP(ret, down(pos*2+1, bm, r, bm, dr));
    return ret;
}
```

Дерево отрезков: обработка запросов сверху вниз

- Для вычисления функции достаточно вызвать `down` из самого доминирующего отрезка.

Задача о двоичном спуске

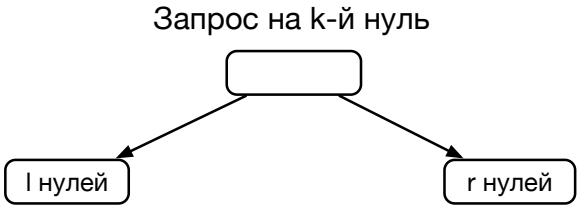
Задача 2. Массив содержит элементы 0 и 1. Можно исполнять следующие запросы:

1. Инвертировать бит в позиции p.
2. Найти позицию $k-$го бита на подотрезке $[l, r]$.

Задача о двоичном спуске

- Сначала сведём задачу нахождения k-нуля на отрезке $[l, r]$ на две задачи: найти k-й нуль с начала отрезка.
- Если он вылезет за границу r — ответ: не существует.
- Тогда достаточно хранить количество нулей на подотрезках.
- В узле храним количество нулей в поддереве.
- Запрос 1: традиционно поднимаемся вверх.
- Запрос 2: найти позицию k-го нуля во всём массиве.

Задача о двоичном спуске



- Если $k \geqslant l$, то уходим в левое поддерево.
- Иначе будем искать нуль номер $k - r$ в правом поддереве.
- Очень напоминает поиск k-й порядковой статистики в массиве через $selection(k)$.

Отложенные операции в дереве отрезков.

Присвоение на подмассиве

Задача 3. Имеется массив $a_1, a_2, \ldots, a_n$. Требуется исполнять операции двух типов:

1. Прибавить одно и то же число d на отрезке массива $[l, r]$.
2. Вычислить сумму элементов на отрезке массива $[l, r]$.

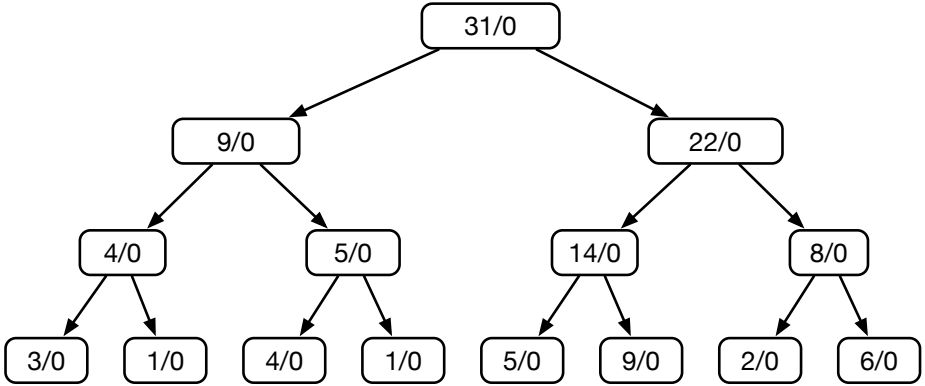
Присвоение на подмассиве

- До сих пор мы умели производить изменения в точке и исполнять операции на отрезке массива за $O(\log N)$.
- Старая техника даст время исполнения запроса изменения на отрезке $(r - l + 1) \log N$.
- Можно уменьшить это время амортизированно до $O(\log N)$.
- Для этого применяем *отложенные изменения*.

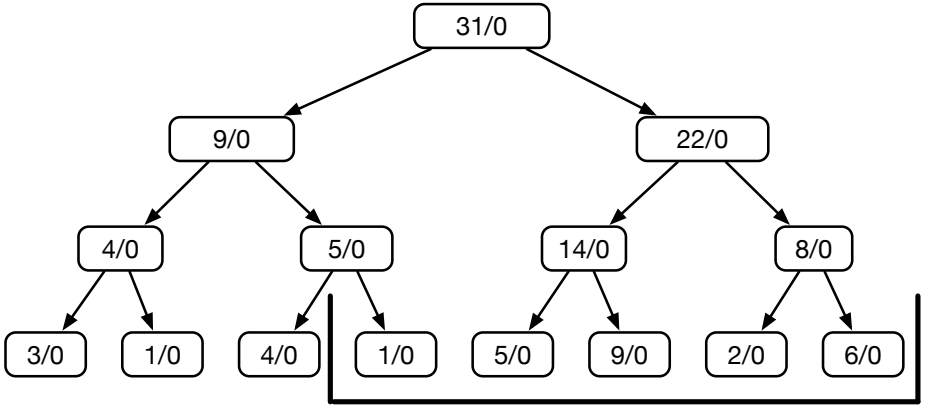
Присвоение на подмассиве

- Будем хранить в каждой вершине кроме суммы sum для второго запроса хранит ещё и параметр операции первого запроса — d.
- Добавим ещё флаг $pending$ — имеются ещё не актуализированные изменения.
- Именно в данной задаче можно было бы обойтись условием $d = 0$.

Присвоение на подмассиве: подопытный кролик

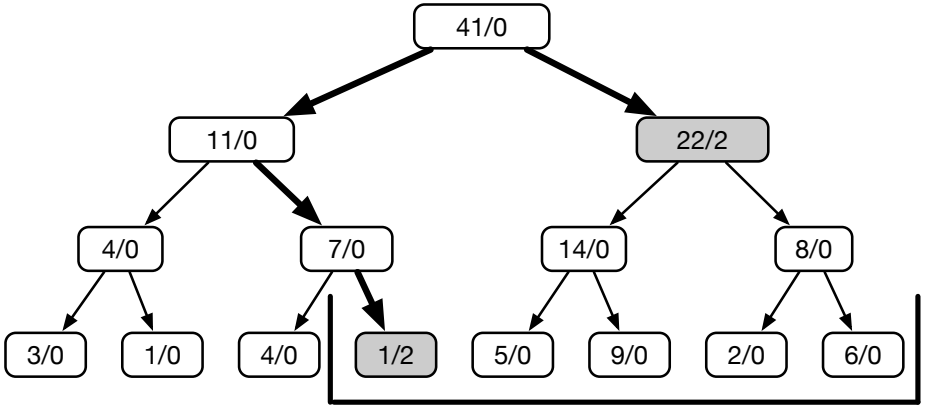


Присвоение на подмассиве: увеличиваем на 2 подотрезок



Присвоение на подмассиве: операция увеличения на $[l, r]$

- Если диапазон текущей вершины $[dl, dr]$ вложен в $[l, r]$, то изменяем sum на d.
- Если $[dl, dr]$ и $[l, r]$ пересекаются частично, запускаем рекурсию и после этого актуализируем sum.



Присвоение на подмассиве: операция нахождения суммы на $[l, r]$

- При заходе в вершину, проверяем наличие в ней ленивого обновления.
- Если оно есть, обновляем текущую sum, проводим обновление по пересекающимся дочерним вершинам, сбрасываем d.

Прибавление арифметической прогрессии на подмассиве

Задача 4. Имеется массив $a_1, a_2, \ldots, a_n$. Требуется исполнять операции двух типов:

1. Прибавить арифметическую прогрессию с первым элементом b и шагом d на отрезке массива $[l, r]$.
2. Вычислить сумму элементов на отрезке массива $[l, r]$.

Прибавление арифметической прогрессии на подмассиве

- Теперь каждая вершина кроме суммы для второго запроса хранит ещё и параметры операции первого запроса — b и d.
- В первой задаче суммы перевычислялась как $sum+ = d + (r - l + 1)$.
- Здесь добавляется сумма $sum+ = (b + (b + d) + (b + 2d) + ...$

Задача 5. Дан массив $a_1, a_2, \ldots, a_n$.
Нужно исполнять запросы: по заданным l, r, x, y сообщать, сколько чисел из подмассива $a_l, \ldots, a_r$ лежат в диапазоне $[x, y]$.

Задача поиска чисел в поддиапазоне на отрезке

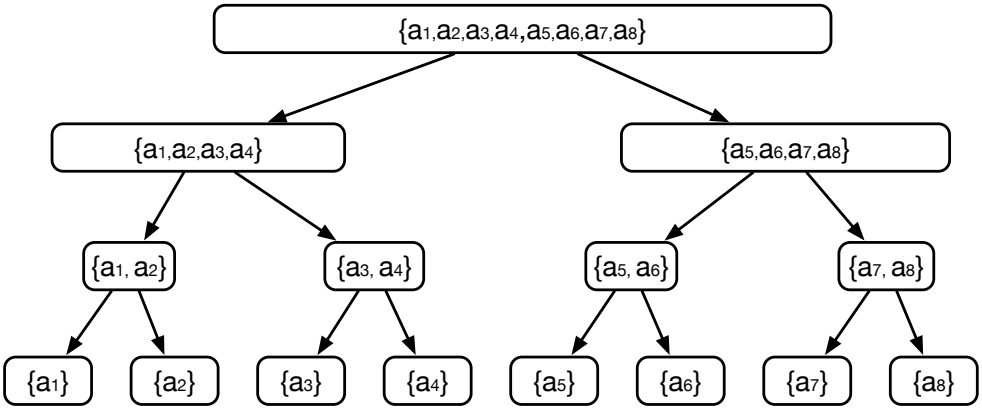
- Разобьём задачу на две:

1. Сколько чисел из $a_l, \ldots, a_r$ не больше $x - 1$.
2. Сколько чисел из $a_l, \ldots, a_r$ не больше y.

- Тогда ответ будет $f(y) - f(x - 1)$.
- Аналогично можно упростить и с границами l и r.

Задача поиска чисел в поддиапазоне: решение 1.

- Сделаем дерево отрезков на ключах, представленных деревьями поиска.
- В узлах будем хранить деревья, построенные на подмножествах.

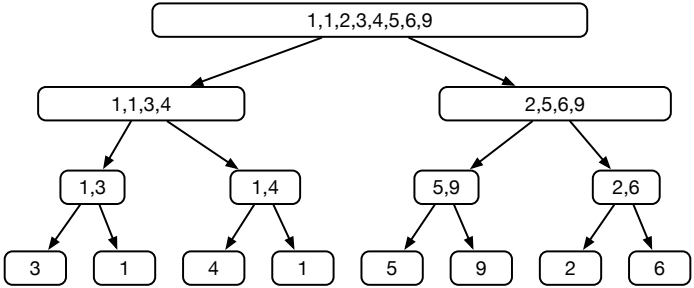


Задача поиска чисел в поддиапазоне: решение 1.

- Как найти количество чисел на отрезке $[4, 8]$, не больших, чем y?
- Отрезок $[4, 8]$ управляется отрезками $[4]$ и $[5, 8]$, которые содержат ключи $\{a_4\}$ и $\{a_5, a_6, a_7, a_8\}$ соответственно.
- В каждом таком узле можно за $\log |t|$ найти количество чисел, не больших y.
- Общее решение находим бинарным спуском от корня.
- Мы разбили отрезок на логарифмическое число деревьев поиска и хорошо бы уметь их сливать.
- Асимптотика $O(\log^2 N)$.

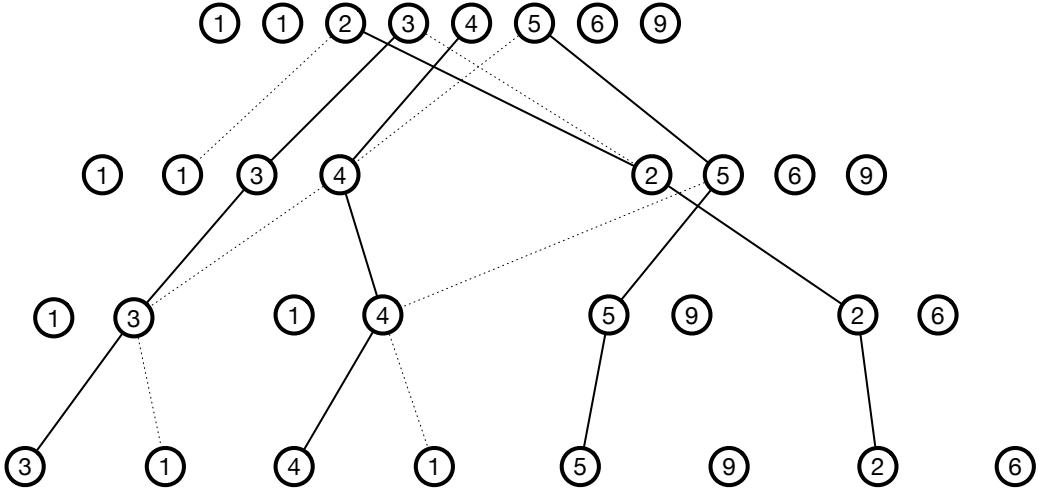
Задача поиска чисел в поддиапазоне: решение 2.

- Давайте хранить в каждом узле не дерево поиска, а отсортированный список ключей.



- Сортировка слиянием отлично подходит для слияния ключей.
- Только давайте её модифицируем для того, чтобы помнить для каждого элемента из узла, тот узел, из которого он пришёл (первый линк).
- Будем помнить и второй — в максимальный элемент оставшихся в другом поддереве, который не больше первого (второй линк).

Задача поиска чисел в поддиапазоне: решение 2.



Задача поиска чисел в поддиапазоне: решение 2.

- Ответим на запрос: сколько чисел на подотрезке $[4, 8]$ не больше 3.
- Бинарным поиском находим 3 в верхнем узле.
- Там отсортировано, есть ответ на вопрос: сколько элементов вообще не больше 3.
- Для неполных отрезков нужно спускаться в детей.
- Бинарный поиск в подузле наивен.
- Опускаемся вниз по первому линку и смотрим, откуда пришла 3.
- Она пришла из левого ребёнка. Что мы там видим: 3-ка — 3-я порядковая статистика в левом ребёнке.
- А в правом ребёнке все элементы по второму линку меньше её, а справа от линка — не меньше её.
- Так, спускаясь до терминальных узлов получаем результат.

Персистентные деревья.

Задача поиска чисел в поддиапазоне: решение 3.

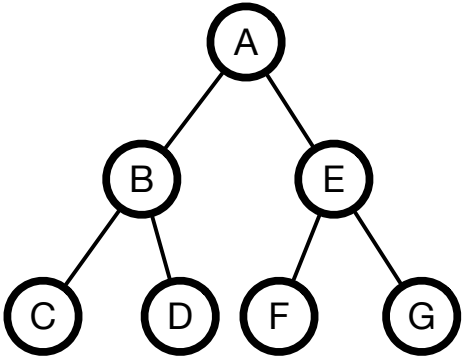
- Используем *персистентное дерево отрезков*.
- Изменим представление дерева отрезков с дерева на массиве на классическое дерево с указателями.
- Когда мы впервые формируем вершину с доминирующим диапазоном, мы создаём все поддеревья для неё.

```
it_node *it_create(int dl, int dr) {
    it_node *t = malloc(sizeof *t);
    t->dl = dl; t->dr = dr;
    if (dl < dr) {
        int dm = (dl + dr) / 2;
        t->left  = it_create(dl, dm);
        t->right = it_create(dm+1, dr);
    }
    return t;
}
```

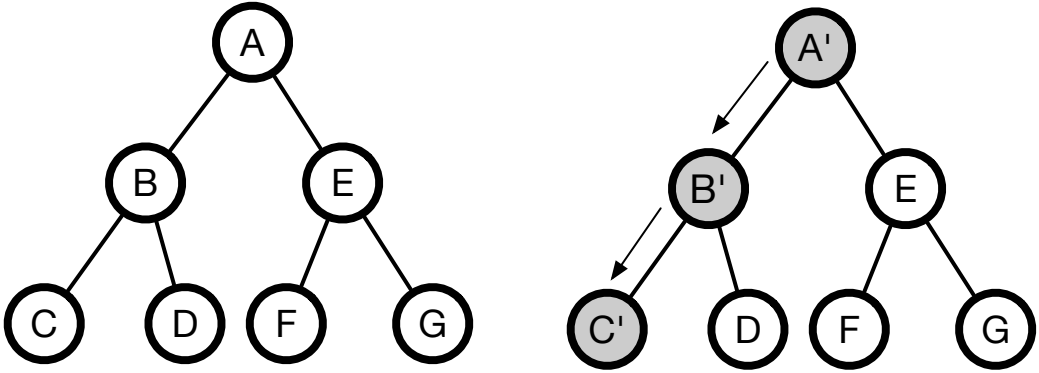
Задача поиска чисел в поддиапазоне: решение 3.

- Идея перститентных деревьев в том, что мы собираемся хранить всю историю их изменений.
- Наивный способ — клонировать дерево перед изменением и работать с клоном.
- Но у нас есть указатели.

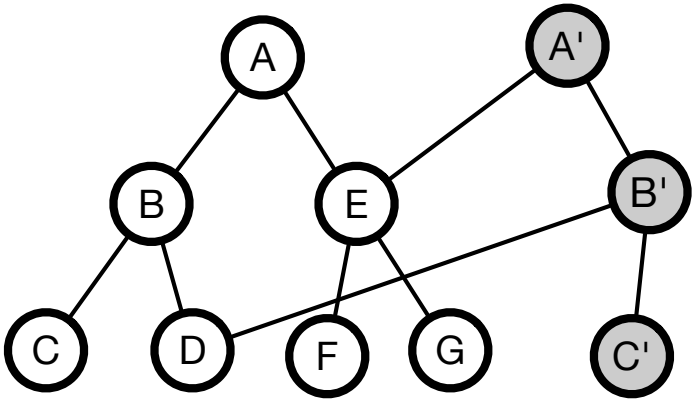
Наивное изменение деревьев отрезков: до



Наивное изменение деревьев отрезков: после



Персистентное изменение деревьев отрезков: после



Персистентное дерево отрезков

- Идея в том, что изменений в дереве $\log N$.
- q поколений изменений потребуют $q \log N$ памяти.
- Когда меняется что-то на пути от вершины к корню, клонируется только текущий узел, с сохранением всех указателей.
- Если при обновлении мы пошли направо — создали клон правого сына, налево — левого.
- При изменении дерева вначале клонируем корень и при посещении клонируем подузлы в нужных направлениях.
- Запросы могут проводиться к любому корню.

Задача поиска чисел в поддиапазоне: решение 3.

- Нам всё ещё нужно определить количество чисел на подотрезке $[l, r]$ массива a, не больших y.
- Создадим массив пар: значение элемента и его номер и отсортируем по значению. Назовём его b.
- Заведём персистентное дерево отрезков, в котором положим все элементы массива b и пометим их неактивными.
- Будем помещать в персистентное дерево отрезков все элементы массива b, делая их активными. Каждое добавление создаёт новый корень.
- В каждой вершине будем хранить количество активных вершин на поддеревьях.
- Теперь решение задачи: находим бинпоиском индекс дерева с наименьшим j таким, что $b[j] \geqslant y$.
- В этом дереве считаем активные элементы на поддиапазоне.
- Асимптотика — $O((N + Q) \log N)$ по времени.

Разреженные таблицы (sparse table)

Задача RMQ

Задача 6. Имеется статический массив $a_1, a_2, \ldots, a_N$. К нему поступают запросы: найти минимум на отрезке $[l, r]$ — Range Minimum Query.
Решение: Несколько решений уже есть: декартовы деревья, деревья отрезков.
Все требуют $O(\log N)$ времени на запрос.
Ценой дополнительной памяти в $O(N \log N)$ можно отвечать на запрос за $O(1)$.

Задача RMQ

- Пусть у нас будет массив $v[L][N]$ такой, что $v[r][i]$ — минимальное число на отрезке длины 2^r, начинающемся в $i-$й позиции.
- L — максимальное из чисел, для которых $2^L \leqslant N$, т. е. $L = \lfloor \log_2 N \rfloor$.
- Нулевой слой содержит элементы самого массива.
- Первый слой содержит минимумы от пар, начинающихся в i.
- Второй слой — минимумы от четвёрок, начинающихся в i.

0	2	7	1	8	2	8	5	4
1	2	1	1	2	2	5	4	
2	1	1	1	2	2			
3	1							

- Как поможет такая таблица?
- Ищем минимум на отрезке $[l, r]$.
- Найдём такое максимальное k, что $2^k \leqslant r - l + 1$.
- Элемент $v[k][l]$ содержит минимум из $a[l]..a[l + 2^k - 1]$.
- Элемент $v[k][r - 2^k + 1]$ содержит минимум из $a[r - 2^k + 1]..a[r]$.
- Эти отрезки пересекаются.
- Операция $\min$ коммутативная, ассоциативная и идемпотентная.
- $\min$ от предложенных элементов есть $\min$ на всём подмассиве $[l, r]$.
- Асимптотика: $N \log N$ на построение, $O(1)$ на каждый запрос, если предподсчитать все $\log_2[1..N]$.