

## Теоретическое решение задачи В.

### Алгоритм решения и доказательство его правильности.

Данная задача ставит перед собой следующие задания:

- 1) Хранение ряда из школьников с уже определенным ростом для каждого
- 2) Переворот определенной части ряда
- 3) Подсчет суммарного роста на определенном участке ряда.

В качестве структуры данных, которая будет решать поставленные задачи, мы выбрали объединение декартового дерева по неявному ключу и дерева отрезков.

**Хранение данных.** Каждый ученик, имеет свой узел. Узел состоит из следующих компонентов:

- *Priority* (случайное число). Дает возможность однозначно задать дерево, и при этом задать его приблизительно с высотой  $\log(n)$ . Относительно этого значения дерево формируется в виде двоичной кучи.
- *Value*. Рост ученика.
- *Left*. Ссылка на левое поддерево.
- *Right*. Ссылка на правое поддерево.
- *Sum*. Переменная, которая будет служить для подсчета суммы на отрезке. Элемент дерева-отрезков. Хранит в себе сумму *Value* всех узлов поддерева, в котором узел является корнем.
- *Count*. Переменная, которая будет служить для определения неявного ключа, по которому дерево формируется в виде двоичного дерева. Хранит в себе количество узлов поддерева, в котором узел является корнем.
- *Reverse*. Булева переменная, хранит значение, отвечающее на вопрос: «Нужно ли переворачивать данное поддерево?».

### Используемые методы:

- *Split*. Разбивает дерево по заданному ключу на два новых дерева (первое дерево – лево стоящие ученики от заданного, второе дерево - ученики, стоящие начиная с заданного).
- *Merge*. Склеивает два дерева в одно (соединяет два ряда студентов).
- *Push*. Меняет местами левое и правое поддерево, если значение *Reverse* узла – true, и передает значение *Reverse* – true сыновьям, меняя свое на противоположенное.
- *Insert*. Добавляет элемент в дерево (добавляет ученика в конец ряда). Используется для первоначального построения ряда.

### Нахождение суммарный рост учеников начиная с ученика $l$ по ученика $r$ :

- Разбиваем ряд на два по ученику  $l$  (метод *Split*).
- Разбиваем полученный второй ряд по ученику  $r+1$  (метод *Split*), полученный первый ряд и будет ряд учеников от  $l$  до  $r$ .
- Возвращаем значение *Sum* корня нового дерева, которое хранит суммарный рост всех учеников дерева, в котором благодаря нашим разбиваниям остался только нужный нам отрезок ряда.
- В конце склеиваем 3 полученных ряда (двумя методами *Merge*).

### Зеркальный поворот ряда учеников начиная с ученика $l$ по ученика $r$ :

- Разбиваем ряд на два по ученику  $l$  (метод *Split*).
- Разбиваем полученный второй ряд по ученику  $r+1$  (метод *Split*), полученный первый ряд и будет ряд учеников от  $l$  до  $r$ .

- Придаем корню нового дерева, значение true для переменной *Reverse*.
- Склеиваем ряд  $[0; l)$  с рядом  $[l; r]$  (метод *Merge*). Метод *Merge* рекурсивно сливает два дерева и на каждом шаге рекурсии вызывается метод *Push* который и отвечает за перестановку поддеревьев местами. Таким образом на каждом шаге рекурсии меняются местами левое и правое поддерева узла. И осуществляется переворот ряда  $[l; r]$ .
- В конце к полученному ряду добавляем третий ряд

### Временная сложность.

Наш алгоритм включает такие функции:

1. **Split.** Рекурсивно вызывается одна операция Split для дерева меньшей высоты (хотя бы на 1) и делается еще  $O(1)$  операций. В итоге получаем оценку  $O(h)$ , где  $h$  – высота дерева, которая в среднем случае равняется  $O(\log n)$ , и следовательно время выполнения функции равняется  $O(\log n)$ .
2. **Merge.** Рекурсивно вызывается одна операция Merge для дерева меньшей высоты (хотя бы на 1) и делается еще  $O(1)$  операций. В итоге получаем оценку  $O(h)$ , где  $h$  – высота дерева, которая в среднем случае равняется  $O(\log n)$ , и следовательно время выполнения функции равняется  $O(\log n)$ .
3. **Insert.** Вызывает методы: Split, Merge, Merge, каждый из которых имеет временную оценку  $O(\log n)$ , что в конечном итоге и даст временную сложность  $O(\log n)$
4. **Push.** Все операции - условные или присваивания - выполняются за  $O(1)$ , как и вся функция.

И конечные функции, которые требуются в условии задачи:

1. **Reverse.** Вызывает методы: Split, Split, Merge, Merge, каждый из которых имеет временную оценку  $O(\log n)$ , что в конечном итоге и даст временную сложность  $O(\log n)$
2. **SumFromAToB** Вызывает методы: Split, Split, Merge, Merge, каждый из которых имеет временную оценку  $O(\log n)$ , что в конечном итоге и даст временную сложность  $O(\log n)$

Видим, что все функции реализованы с одной асимптотикой  $O(\log n)$ .

Непосредственно в нашей задаче задается ряд учеников, а для того что бы мы могли делать операции с этим рядом, нам надо заполнить наше декартово дерево. Делаем мы это с помощью вызова  $n$ -го количества метода Insert с оценкой  $O(\log n)$ , что дает асимптотику  $O(n \log n)$ . А также требуется, произвести  $m$  запросов на переворот определенного участка ряда из учеников или получить суммарный рост определенного участка ряда учеников, это выполняется двумя методами: Reverse и SumFromAToB с оценками  $O(\log n)$ , что дает асимптотику  $O(m \log n)$ .

**Общая асимптотика решения задачи  $O(n \log n + m \log n) = O((m+n) \log n)$ .**

### Затраты памяти.

Для реализации описанного выше алгоритма, требуется постоянно хранить только  $n$  узлов декартового дерева каждый из которых хранит в себе количество узлов в поддереве, приоритет, рост одного ученика, сумму роста учеников поддерева, ссылку на левое и правое поддерево и булеву переменную, отвечающую за переворот отрезка

**Таким образом, итоговые затраты памяти =  $O(n)$ .**