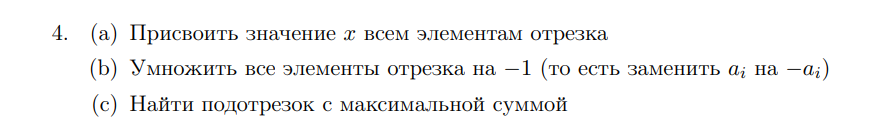
Назаров Рустам М3132 368563

Алгоритмы и структуры данных Практика 4 номер



Здравствуйте, это я, который два раза пытался защитить данную задачу, но «слишком мудрил» с методом построения дерева. В последней попытке я предложил простой способ данной реализации, с которым вы согласились, но, полностью реализовав данный метод, я пришел к выводу, что он не работает. Дальше я попробую показать и доказать, что он не учитывает.

А после доказательства все-таки продемонстрирую **свою** большую, но **рабочую реализацию**. Докажу и покажу весь код. Прошу прочитать полностью. Я написал свои большие случайные миллионные **тесты**, лишний раз доказывающие работоспособность кода.

Если Вы посчитаете, что реализация практики верна, все равно прошу дать шанс моей реализации, ведь я правда потратил много сил и времени и в итоге получилось.

Спасибо

Метод с Практики

Что храним: 1) Префикс

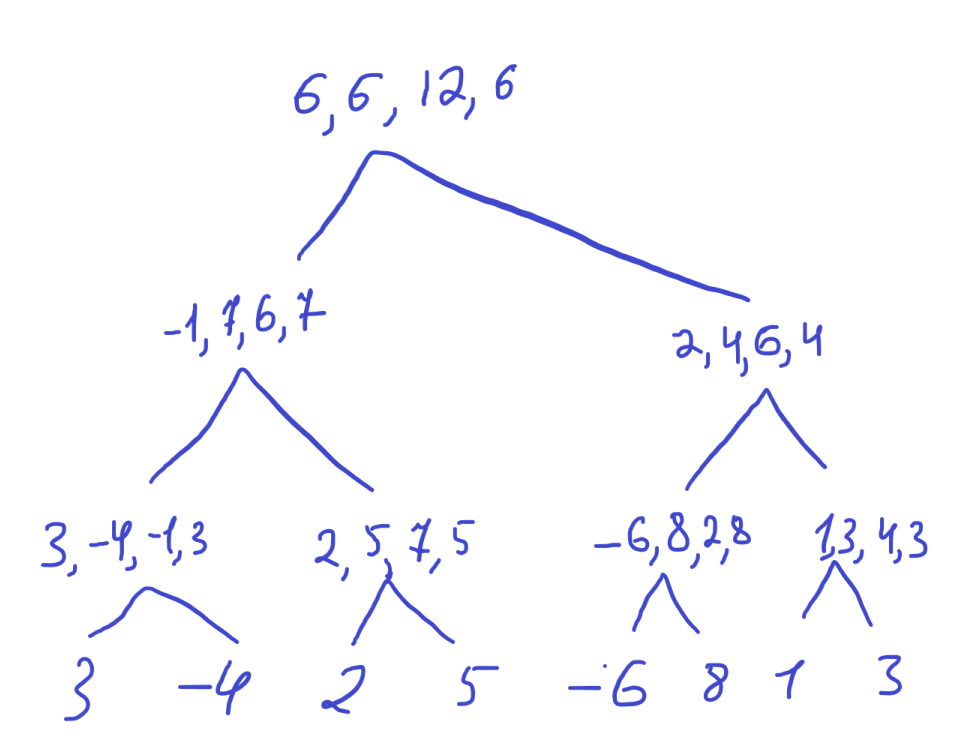
2) Суффикс

3) Сумма на отрезке

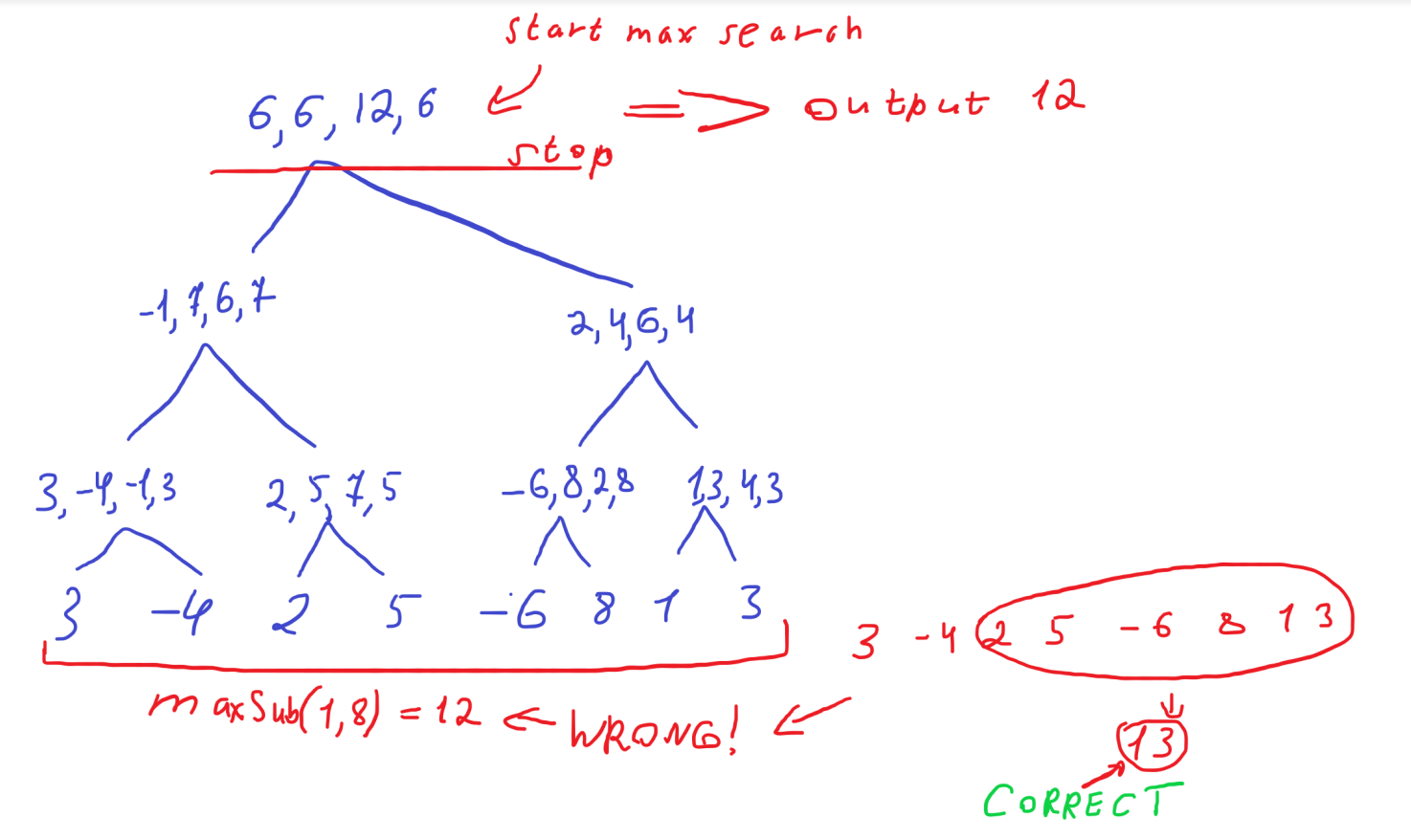
4) Префикс/Суффикс (максимум)

Массив [3, -4, 2, 5, -6, 8, 1, 3]

Результат подсчета:



Теперь отправляю запрос: get maxSubSegment(1, 8) – получить максимальный подотрезок на всем множестве:



Первый родитель полностью входит в наш диапазон, поэтому, остановившись на нем, берем максимум от четырех чисел => Ответ древа: 12!

Несложно увидеть, что в действительности правильный ответ 13:

2 + 5 + -6 + 8 + 1 + 3 = 13

Видим, что дерево не конкатенирует левый подотрезок с правым по частям, только сумму полностью левого и правого.

Попробуем исправить: что если будем хранить: 1) Максимум Префиксов

2) Максимум Суффиксов

3) Сумма на отрезке

4) Префикс/Суффикс (максимум)

Несложно увидеть, что тогда вверху мы получим

(7, 8, 12, 8) => Ответ все равно не будет верным, тут никак не получить 13

Чтобы получить 13 нам надо сложить сумму правого 6 и 7 из левого, так что

Что если будем хранить: 1) Префикс

2) Суффикс

3) Сумма на отрезке

4) максимум Префиксов/максимум Суффиксов

Тогда увидим такое: (6, 6, 12, 8) => и тут не получилось 13

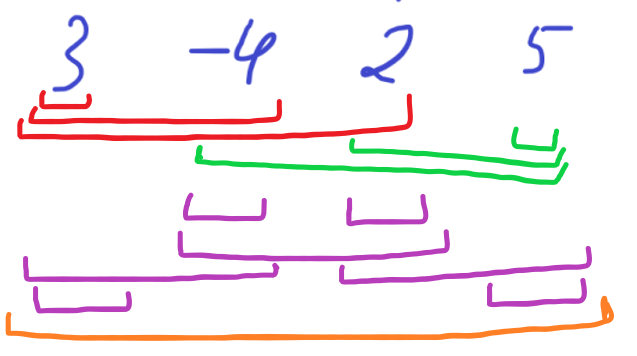
Также, на практике был написан псевдокод, что, если отрезок полностью входит в промежуток надо возвращать его сумму. Но рассмотрев промежуток (4, 7) то есть [5, -6, 8, 1]. Мы видим, что необходимо вернуть 9, то есть в родителе -6 8 надо вернуть не сумму, а цифру 8 и как-то понять, что сложить именно с правым, а с левым нельзя, ведь уже не будет подотрезок.

Но вообще, немного посидев, эту ошибку решить не проблема. Проблема решить первую проблему листом выше.

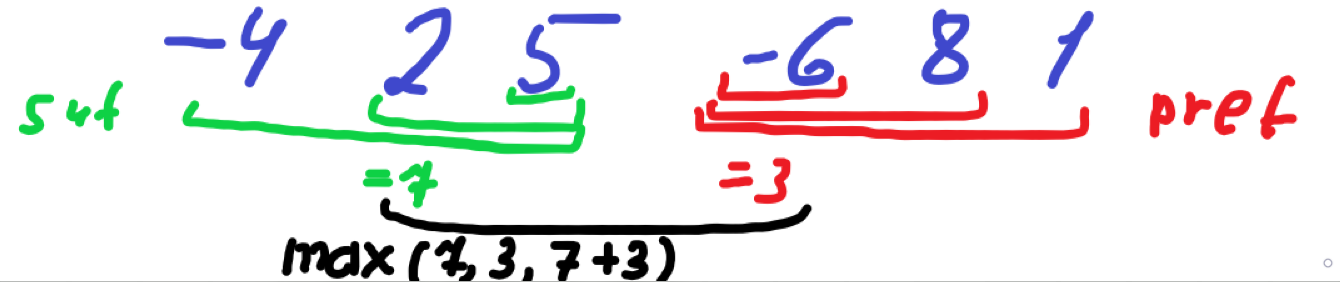
В общем я решил придумать новый способ подсчета.

**Моя реализация**

Раз ключевая проблема в частичном сложении левой и правой веток, то я хочу получить дерево, которое будет выбирать максимумы всех префиксов и префикс плюс префикс суффикса и максимумы всех суффиксов и суффикс плюс суффикс префикса, также хочу рассматривать сумму суффикса префикса и префикса суффикса, то есть внутреннюю сумму. В общем необходимо рассмотреть все.

<= Вот, что необходимо.

“работа СП внутр. Сум.”



 Сумма всего отрезка

Максимальный префикс. Число, которое можно сложить с числом слева, то есть с максималным суффиксом или суммой левого родителя. То есть я готовлю почву, чтобы сложить максимольную сумму начиная от левого влево правого родителя с максимальной с суммой начиная от правого вправо левого родителя. Смотр. Рис “работа СП внутр. Сум.”

Максимальный суффикс. По аналогии префикса. Берем все подотрезки (кроме суммы, ведь она рассматривается в орнаж.), содержащие самое правое число родителя, и берем максмум. Так мы сможем сложить свою максимальную сумму справа с максимальной суммой слева родителя справа.

Максимальный подотрезок отрезка, без самого отрезка. Берем максимум из 4 значений суффикса и 4 префикса. И сравниваем со внутренней суммой: суффикс перфикса + префикс суффикса. Сумма, которую не сложить ни с кем-то левым, ни с кем-то правым. Так рассматриваем последний случай.

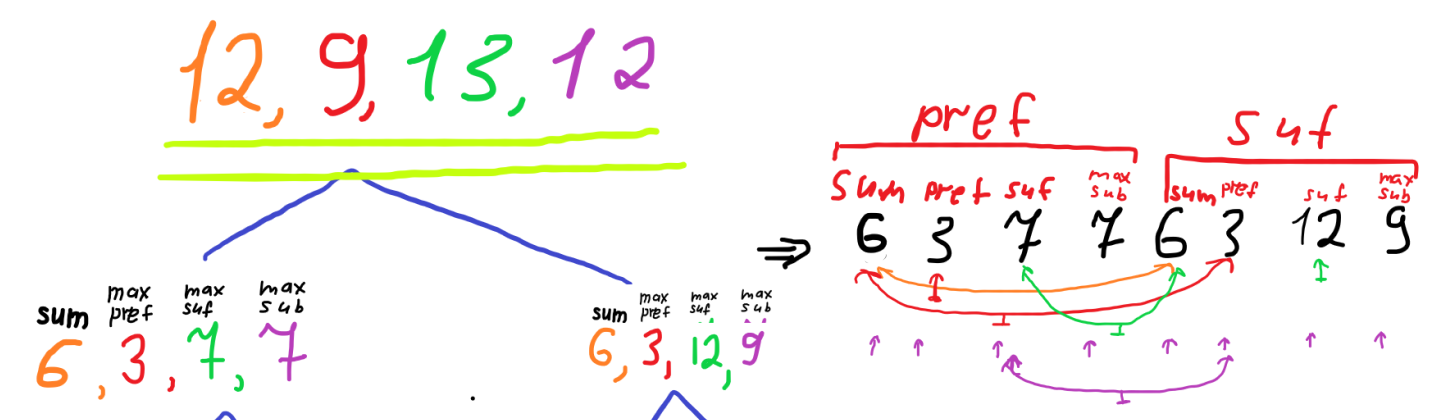
**Что хранить**: 1) Сумма отрезка

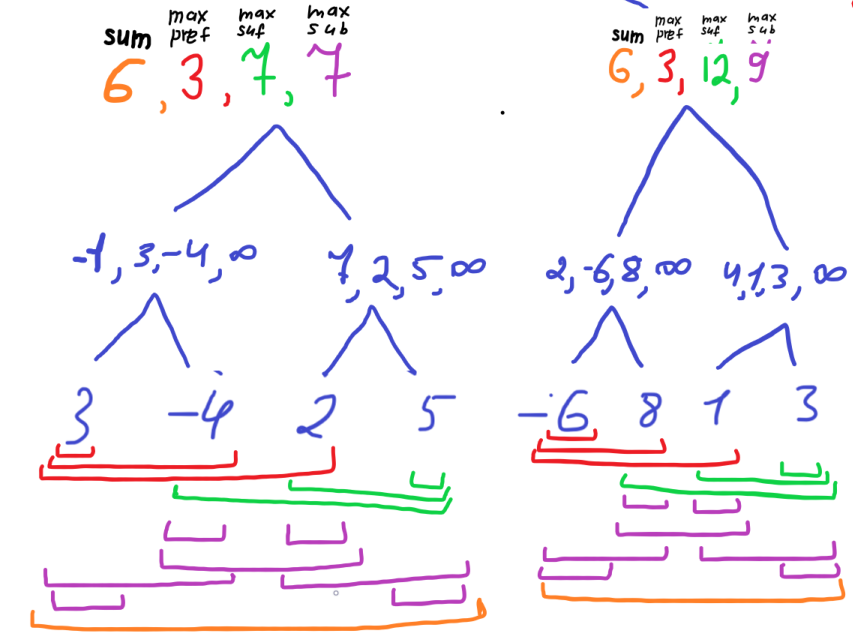
2) Максимальный префикс

3) Максимальный суффикс

4) Максимальный подотрезок

Таким образом получаем голову =>



И тело:

Ну и теперь явно получаем верный ответ на запрос. Просто берем максимум от четырех чисел головы: maxSubSegment(1, 8) = 13

Почему так уверен, что рассматриваешь все случаи?

Итого, что мы получаем каждый раз?

- Сумма

- Максимальный префикс. Максимальная сумма, содержащая самое левое число

- Максимальй суффикс. Максимальная сумма, содержащая самое правое число

- Максимальный подотрезок, рассматривающий и внутреннюю сумму, которую с соседями не объединить, так как не имеет самый левый и самый правый член, но которому ничего не мешает быть наибольшим.

В каждом родителе, имея второй и третий пынкты, мы каждый раз рассматриваем сумму левого суффикса с правым префиксом, где каждый является своим локальным максимумом суффиксов или префиксов. То есть на руках максимум, содержаший правое число левого родителя, и максимум, содержащий левое число правого родителя, то есть максимумы, которые мы имеем право суммировать.

И в четвертом пункте каждый раз получаем максимальный подотрезок, в процессе так же рассматривается внутренний отрезок, который не соединить ни с кем. То есть если мы где-то в самых листьях получили огромное число, то мы всегда будем готовы отнести его в самый вверх, если он и правда больше любой суммы, так как мы сравниваем постоянно. То есть голова всегда знает обо всех своих детях, каждый ребенок может подняться к голове, если большое число.

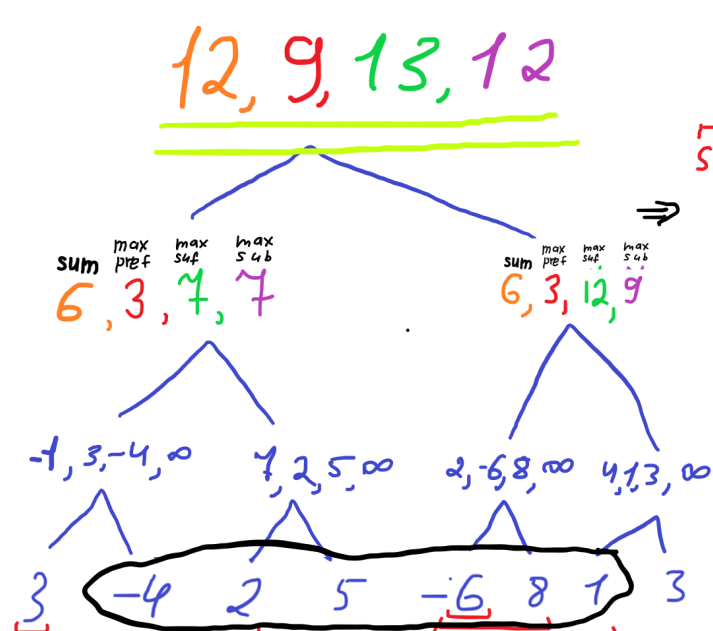
Хорошо, 1-3 уровни понятны, что там выше?

Все то же самое. Слева у нас сумма и максимумы, справа у нас сумма и максимумы. Мы просто точно так же рассматриваем все возможные комбинации и несем выше. То есть снова рассматриваем макс префикс, чтобы сложить с суффиксом родителя слева. Как? Сравниваем макс префикс префикса с суммой префикс и макс префикс суффикса, реально получая левую сумму, как и требуется. Аналогично с макс суффиксом. Что там с максимальным подотрезком, макс 4 тех 4 тех ясно, но что со внутренним? Ничего не поменялось складываем макс суффикс префикса + макс префикс суффикс, получая желаемое.

А что делать когда стык нескольких деревьев?

Тут тоже очень интересно!

Пусть запрос: maxSubSegment(2, 7) от второго до предпоследнего

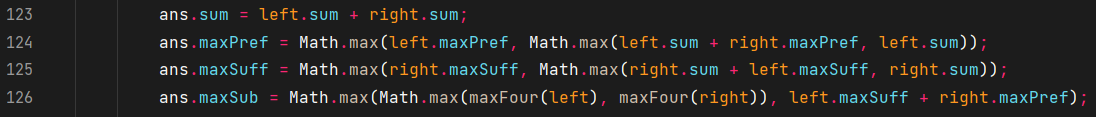


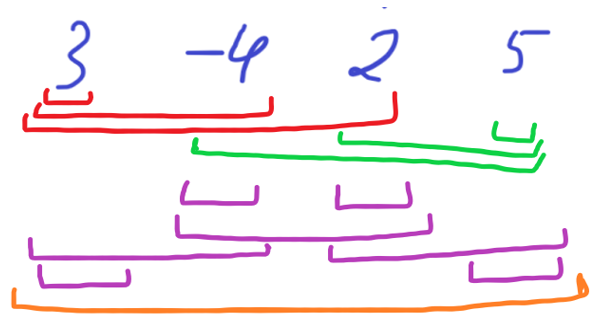


Спускаясь рекурсией до момента, пока не окажемся полностью внутри мы получаем какой-то date, содержащий информацию о себе (ну, 4 числа, что содержим). Например, когда мы в пункте 2, мы запустили рекурсию влево вправо, ожидая 2 результата: два date, это будут 4 и 5 пункты.

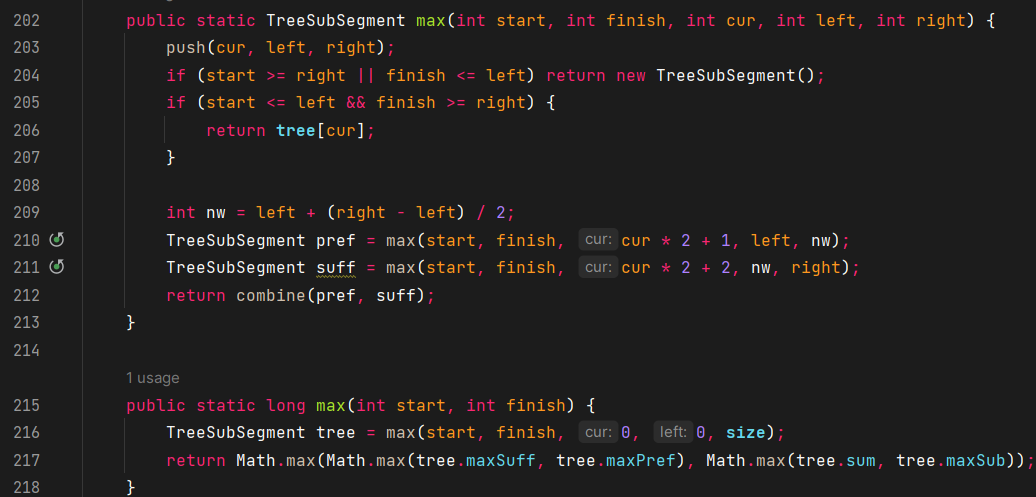
Получили и что дальше?

Строим date из них двоих, делаем combine. Все по тому же правилу.





Сделали один date из двух детей 2 пункта, то же самое один date из двух детей 3 пункта (дети 6, 7). И так как в начале запуска рекурсии были в голове, голова получила два результата date 2 и 3 пунктов. Да, из этих двоих тоже делаем один date. И все, из финального date берем максимум. Вот ответ!



И это работает?

Конечно, ведь мы снова каждый раз берем сумму, макс префикс, суффикс, подотрезок. Складывает все с чем можем. В общем вест принцип тот же все работает.

Звучит кажется нормально, но что по времени?

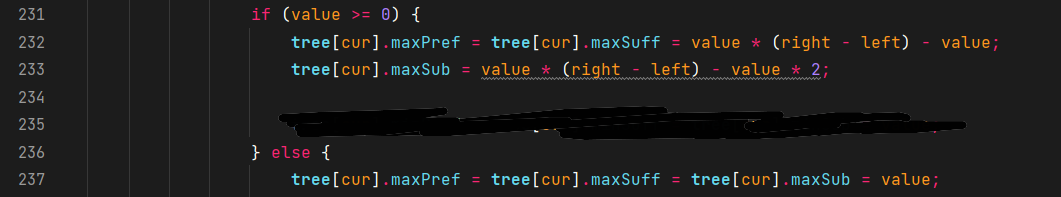
O(const \* lon(n)), константа и есть эти несколько сравнений, чтобы создать date из двух date’ов. А все остальное у нас уже было подсчитано при построении. В общем со временем точно все в порядке, ведь мы не идем туда, куда не просят, к детям, не лежащим в промежутке, или к детям, когда уже полностью в промежутке, потому что все необходимое подсчитано уже при построении.

Это все хорошо, ну а что там с Set(left, right, value)?

Главный плюс set это то, что он все числа под родителем меняет на какое-то одинаковое, то есть нам очень легко понять, что где будет находиться.

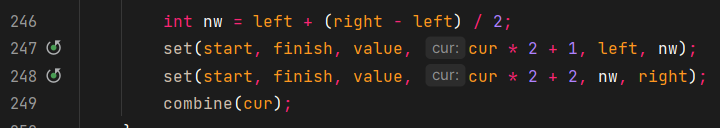
Например, если нам дали положительное число, то очевидно, что сумма это количество детей под родителем, умноженное на value. Макс префикс – это та же сумма, но минус один value. Макс суффикс аналогично. Макс подотрезок – это внутренни: та же сумма но минус 2 \* value.

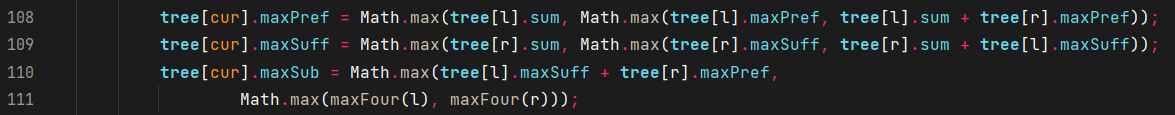
Если отрицательное. Сумма так же. Префикс = суффикс = подотрезок = value

Это, что касается самого родителя, который полностью в промежутке. 

А что с родителями этого родителя, как они узнают и меняются?



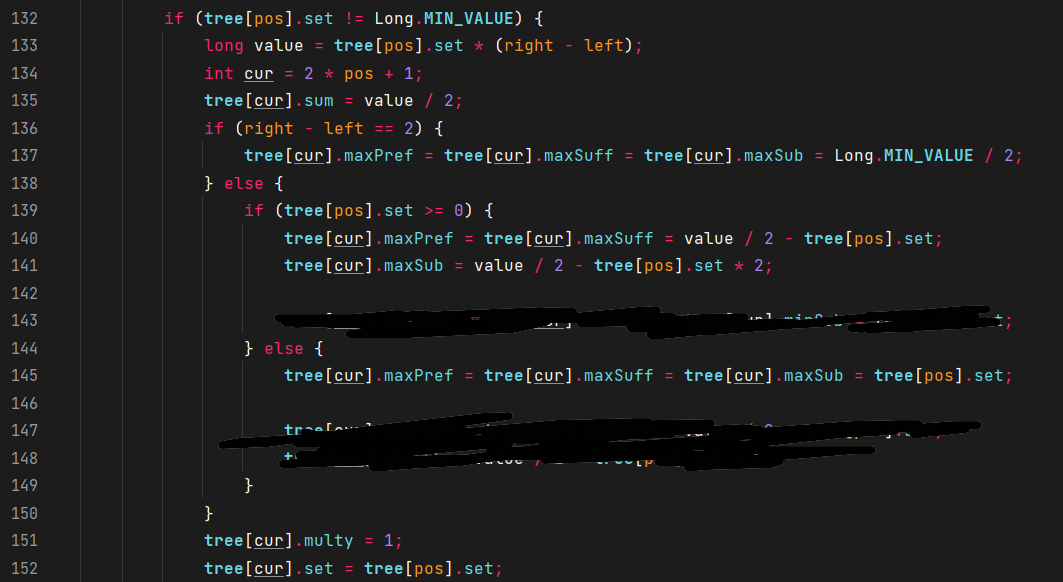


Ну тут как обычно, начиная с головы, мы идем вниз по детям, пока не будем полностью в отрезке. Получается рекурсия, где, как только приходим в желаемую точку меняем значения по вышесказанному. Помемняли значения, метод закончися. Вот закончились рекурсии, но нет, дальше у нас после объявления рекурсии везде стоит combine, то есть пересчет значений, если что-то поменялось. 

То есть родители точно узнают и все пересчитывают. Со временем тоже все в порядке.

Ну допустим, а что там с детьми, они когда получат приказ поменяться?

Ну а это метод push.



(^Зачеркнутое позже) (То же и с правым ребенком)

В Set’е мы запоминали значение value, чтобы в push сообщить и детям. Дети делают то же самое. Если значение .set не равняется бесконечности MINVAL,

Значит меняем двух следующих ребенков. Меняем все по тому же принципу из обычно Set’а если положительное умножаем на количество детей и где-то минус один или два value, и ничего не поменялось и для отрицательного числа. Дальше мы у себя значение set возвращаем в бесконечность, а детям наоборот меняем значение set на value, чтобы потом, если и до них дойдет функция они сделали push дальше. Когда делаю push? В начале каждой функции, которая будет заменять ее (set/multy) или перед max. Но только того, кого касаюсь сейчас, не ниже. В общем работает правильно.

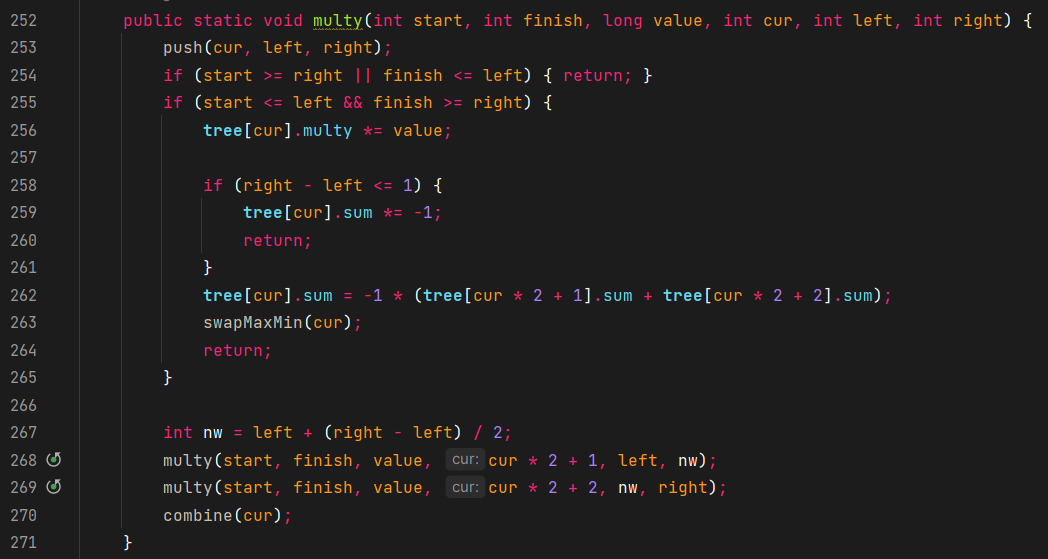
Ну вроде работает, а что с Multy(-1)?

Тут посложней. Что будет с максимами, если их умножить на -1? Они станут минимумами, а что будет с минимумами, если умножить на -1? Они станут максимумами!

Что ж отсуда и вынужденная мера: Храним еще три числа в каждом родителе! Весь принцип тот же, минимальный префикс, минимальный суффикс, минимальный подотрезок. Да, все что было на фотографиях закрашено это был пересчет минимумов. Принцип там точно такой же. Итак, храня и максимумы и минимумы, мы всегда очень быстро сможем узнать какой максимум после умножения на -1. Просто поменяв местами максимумы и минимумы. Ну и также поменять знак каждого из четырех (уже +3, семи!) на противоположный. Почему это работает? Потому что макс и мин и сум это все суммы, то есть (a + b + c), а если сумму умножить на -1, будет (- a - b - c).

Все, очень быстро мы получаем противоположный знак.

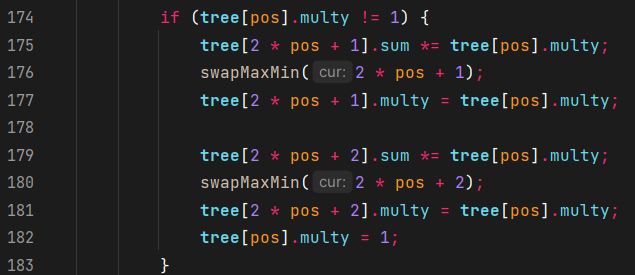
А что там с родителями родителя, который полностью в отрезке?



Все то же самое, в рекурсии поменяли значение, сделали пересчет всего. (В пересчете пересчитываются и минимумы, просто были закрашены)

Ну а что с детьми?

Тот же push



Если перестал multy равняться 1, то умножаем все. И свапаем значения (внутри свапа тоже умножаем на -1).

Стоп, что ты вообще в коде хранишь в каждом родителе?

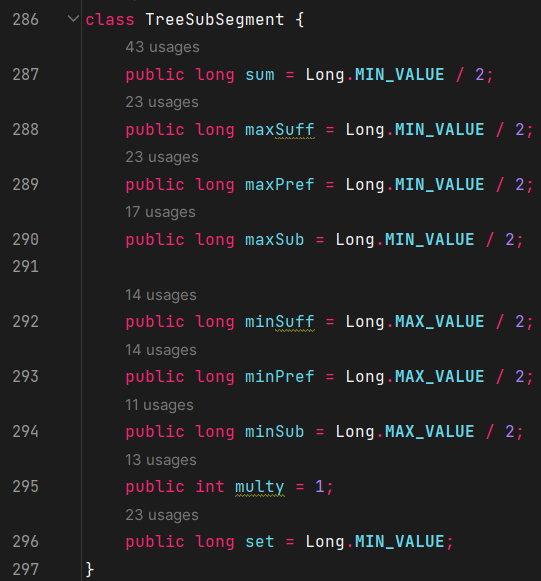
Да, храню я многое, но все это необходимо.

Сумма, макс суффикс, макс префикс, макс подотрезок, которые поумолчанию бесконечнось (Long.MIN\_VALUE / 2).

Мин суффикс, мин префикс, мин подотрезок, по умолчанию (Long.MAX\_VALUE / 2).

Multy, сигнал, что было умножение (по умолчанию 1)

Set, сигнал, что был set (по умолчанию Long.MIN\_VALUE)



Ну допустим, а у тебя не пересекаются set и multy?

Нет.

Если Set был раньше, то все работает как надо, так как multy никак не опускает set, а проверка на set раньше, то есть мы сначала сделаем set, потом multy, так как set выше он спуститься на уровень появления multy, и ничего не сломает. Если set был выше. Если ниже, то тоже работает, ведь multy опуститься на уровень set, и потом сначала сделает set, после multy.

А если set был позже, то тоже работает, так как set опускает multy в своем push, и он раньше проверяется.

Ну вроде да, а какой build?

Стандарт

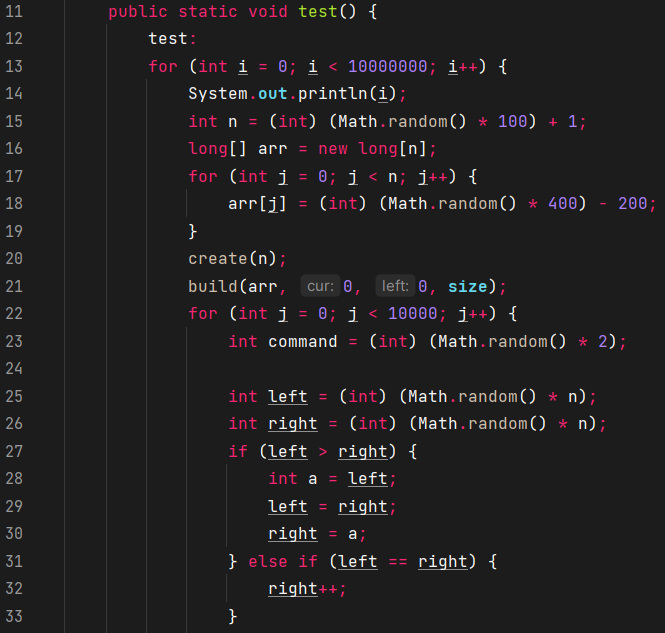


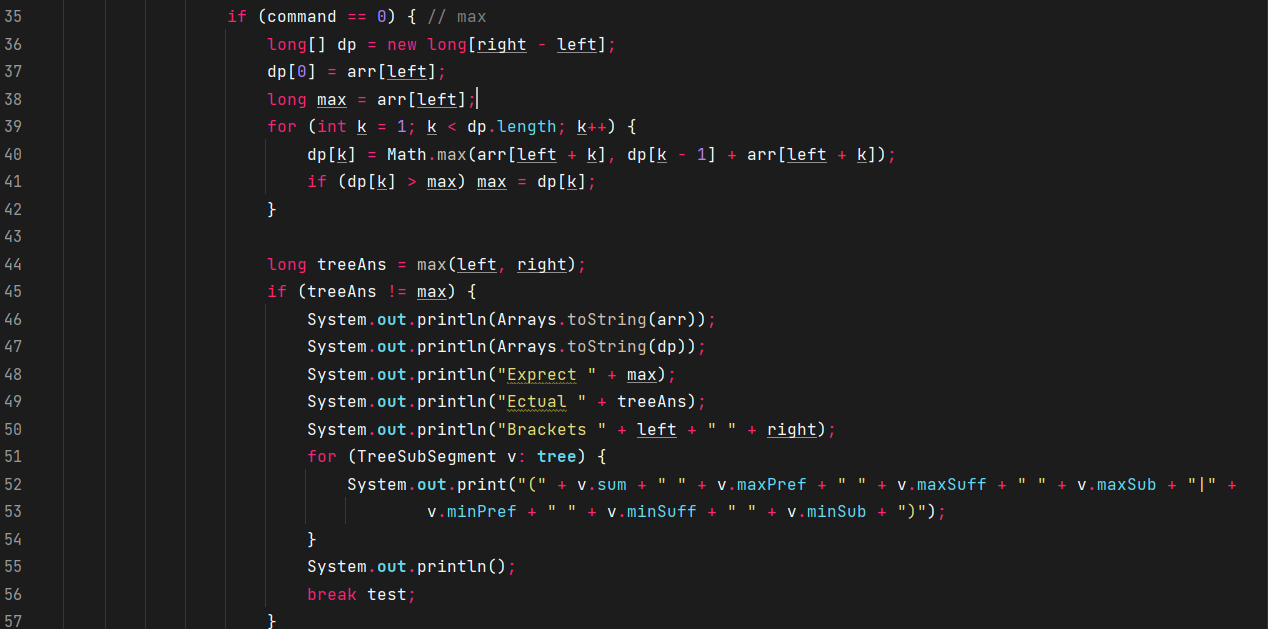
**Ну раз так, то что там по теста уже?**

ТЕСТЫ

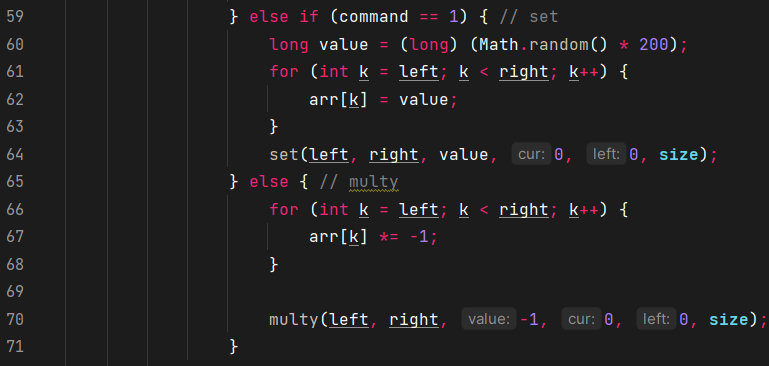
Тесты генерируются полностью случайным образом. Я поставил 10млн тестов, где каждый раз генерируется массив слуйной длины до 100, и случайных чисед от -200 до 200. Создаю дерево. И 10000 раз генерирую случайну команду. Если это max(), то создаю простой массив dp[], который считает максимальную сумму подотрезка. Он работает правильно, так же проходил тестирование.

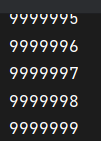
Итак, первая фото, это и есть создание и генерация для старта каждого теста



Дальше метод max где и создаю dp, подсчитаю его ответ и ответ дерева и сравниваю, если что вызываю вывод ошибки и стоп тестов.

Ну и просто вызов двух функций, если ошибочны, то потом в max будет ошибка.





Вот так и выглядит мое дерево. По времени все как надо. Да и по реализации на самом-то деле несложно, просто с нуля придумывать неприятно.

Судя по случайным тестам и моим ручным все правда работает, поэтому прошу оценить работу. Так же предлагаю Вам протестировать мое произведение. Надеюсь Вы не считаете, что слишком замудрено все, ведь по-моему мнению это самый правильный способ. Буду рад предложению еще какого-то способа для решения данной задачи. Но, так как данная реализация тоже является рабочей, прошу не браковать мои труды.

Спасибо огромное, что прочли все!