АНАЛИЗ НА РЕШЕНИЕТО НА ЗАДАЧА
ZEROSUM

Тази задачка е давана на интервюта за работа в големи софтуерни компании. В оригиналния си вариант се пита само дали има тройка с нулева сума, но за да я направим малко по-трудна, тук вместо това се иска да се намери броя на всички тройки с нулева сума.

Най-простият алгоритъм би бил да въртим три вложени цикъла, като проверим всички тройки. Това е решението със сложност $O(N^3)$ и не би хванало около 30 точки (40, ако се имплементира малко по-хитро, тоест се сортират числата и се break-ват циклите, когато стигнем до момент, в който стане невъзможно да получим нови тройки.
Ако фиксираме две от числата (тоест въртим само два вложени цикъла) които да кажем са A и B, то има ли бърз начин да проверим дали има трето, което е -(A + B)? Всъщност има, даже повече от един такъв. По-стандартният от тях е да сортираме предварително числата (със сложност $O(N * \log N)$, а дори и $O(N^2)$ не разваля сложността), след което фиксираме всички двойки числа и с двоично търсене търсим дали масивът съдържа третото. Тъй като можем да имаме повтарящи се числа (тоест двете фиксирани да образуват тройка с повече от едно друго), трябва да намерим двата края на интервала, с който те образуват тройка с нулева сума. Това не е голям проблем – просто правим два binary search-а вместо един. Този метод води до сложност $O(N * \log N + N^2 * \log N)$, тоест $O(N^2 * \log N)$.

Ако имаме право на O(N) допълнителна памет, можем в хеш таблица (hashmap в случая) да пазим всички числа и търсенето ни за число да става константно, вместо логаритмично по време. Това би довело до решение със сложност $O(N^2)$, но O(N) откъм допълнителна памет.

Най-доброто решение ползва отново $O(N^2)$ време, но O(1) допълнителна памет. То използва една стандартна техника, която често се ползва за оптимизиране на динамични задачи, наричана „оптимизация на вътрешния цикъл" (като в случая се ползва за оптимизиране на brute force решение). Отново е нужно да сортираме елементите. След което въртим цикъл по един от елементите на тройката. Нека този елемент е с индекс i. Фиксираме два указателя (в смисъл на нещо, което ни сочи позиция в масива, не на указател към памет) към елементите с индекси i + 1 и N (които ще наричаме, съответно, "ляв" и "десен" указател). Проверяваме каква е сумата на трите елемента. Ако тя е нула - супер. Ако тя е отрицателна, то, очевидно, е в наш интерес да я увеличим. Тъй като сме фиксирали първия елемент и можем да местим само втория и третия указател, то единственото, което можем да направим, за да увеличим сумата, е да преместим левия указател с един елемент надясно. Ако ли пък сумата е положителна, за да я намалим ще преместим десния указател с един елемент наляво. След всяко такова преместване проверяваме отново каква е сумата на трите елемента (фиксирания и двата, които сочим в момента) и действаме адекватно. Тъй като на всяка стъпка или местим левия надясно, или десния наляво, те рано или късно ще се срещнат. Нещо повече, те ще се срещнат след не повече от N такива премествания. Така ако можем да фиксираме първия елемент по N начина, и за всяко фиксиране правим по не повече от N местения, сложността на този алгоритъм е $O(N^2)$. Защо работи и не пропуска ли той решения? С малко мислене можем да се убедим, че той е еквивалентен на алгоритъма с binary search, само че правим търсенето по-умно и спираме по-рано фиксирането на втория елемент. Не е толкова лесно да се види защо той не изпуска решения. Нека разгледаме едно решение (в сортиран масив). То представлява три числа, които можем да наричаме "ляво", "средно" и "дясно" (тъй като, очевидно, едното ще е най-малко (или равно на (някое от) останалите), другото ще е най-голямо (или равно на (някое от) останалите) и третото ще е между тях (или равно на (някое от) тях)). Фиксираното от нас във външния цикъл число е "лявото" (демек най-малкото). Тъй като в хода на вътрешния цикъл двата индекса

се срещат, то със сигурност на някоя стъпка левият индекс е минал през "средното", а десният индекс е минал през "дясното". Ако някой от индексите е преминал през двете числа, а другият през нито едно от тях, то тогава на някоя стъпка от алгоритъма сме преместили неправилния индекс, което е противоречие с алгоритъма или с това, че разглежданата тройка е решение.
Тъй като алгоритъмът не е особено интуитивен и не е лесен за асимилиране, ще дадем и пример:
Дадени са ни числата {7, -5, 2, 3, -4, -4, 2, 0, 1, -6}.
Като първа стъпка от алгоритъма ги сортираме. Получаваме {-6, -5, -4, -4, 0, 1, 2, 2, 3, 7}.
Въртим цикъл за всяко от тях, като казваме, че то ще е най-малкото ("най-лявото") от тройката. При фиксиране на -6 за такова, ето стъпките, които ще се изпълнят във вътрешния цикъл:
{**-6**, -5, -4, -4, 0, 1, 2, 2, 3, 7}, със сума -4, следователно местим левия индекс.
{**-6**, -5, -4, -4, 0, 1, 2, 2, 3, 7}, със сума -3, следователно местим левия индекс.
{**-6**, -5, -4, -4, 0, 1, 2, 2, 3, 7}, със сума -3, следователно местим левия индекс.
{**-6**, -5, -4, -4, 0, 1, 2, 2, 3, 7}, със сума +1, следователно местим десния индекс.
{**-6**, -5, -4, -4, 0, 1, 2, 2, 3, 7}, със сума -3, следователно местим левия индекс.
{**-6**, -5, -4, -4, 0, 1, 2, 2, 3, 7}, със сума -2, следователно местим левия индекс.
{**-6**, -5, -4, -4, 0, 1, 2, 2, 3, 7}, със сума -1, следователно местим левия индекс.
{**-6**, -5, -4, -4, 0, 1, 2, 2, 3, 7}, със сума -1, следователно местим левия индекс.
Индексите се срещат, следователно няма тройка със сума 0, чието най-малко число е -6.
Продължаваме нататък, като фиксираме следващото число, -5.
{-6, **-5**, -4, -4, 0, 1, 2, 2, 3, 7}, със сума -2, следователно местим левия индекс.
{-6, **-5**, -4, -4, 0, 1, 2, 2, 3, 7}, със сума -2, следователно местим левия индекс.
{-6, **-5**, -4, -4, 0, 1, 2, 2, 3, 7}, със сума +2, следователно местим десния индекс.
{-6, **-5**, -4, -4, 0, 1, 2, 2, 3, 7}, със сума -2, следователно местим левия индекс.
{-6, **-5**, -4, -4, 0, 1, 2, 2, 3, 7}, със сума -1, следователно местим левия индекс.
{-6, **-5**, -4, -4, 0, 1, 2, 2, 3, 7}, със сума 0, намерихме отговор!
Ако усложним задачата, като искаме да преброим колко са всички тройки с нулева сума, трябва леко да модифицираме алгоритъма (но той си остава със сложност $O(N^2)$). Трябва да внимаваме правилно да имплементираме вътрешния цикъл да продължава дори след като сме намерили решение, тъй като може да има повече от една валидна тройка с дадено най-малко число.
{-6, **-5**, -4, -4, 0, 1, 2, 2, 3, 7}, със сума 0, намерихме отговор!
На следващата стъпка указателите се срещат, така че прекратяваме вътрешния цикъл.
Понякога може да имаме повече от един отговор с дадено минимално число дори като не ползваме последователни повтарящи се числа.
{-6, -5, **-4**, -4, 0, 1, 2, 2, 3, 7}, със сума -1, следователно местим левия индекс.
{-6, -5, **-4**, -4, 0, 1, 2, 2, 3, 7}, със сума +3, следователно местим десния индекс.
{-6, -5, **-4**, -4, 0, 1, 2, 2, 3, 7}, със сума -1, следователно местим левия индекс.
{-6, -5, **-4**, -4, 0, 1, 2, 2, 3, 7}, със сума 0, намерихме отговор! 1 != 2 и 2 != 3 (тоест следващото ляво и следващото десно число са различни от, съответно, текущото ляво и текущото дясно), така че можем да преместим и двата индекса.
{-6, -5, **-4**, -4, 0, 1, 2, 2, 3, 7}, със сума 0, намерихме отговор!
На следващата стъпка указателите се срещат, така че прекратяваме вътрешния цикъл.

Автор: Александър Георгиев