

Решение задачи о нахождении НОДа двух чисел

1.1 Размышления

Как вы наверное догадались, задача решается путём переполнения целочисленного типа *int*. Рассмотрим подробнее этот тип в языке C++.

Для целочисленных переменных на 64-х разрядных операционных системах (ОС) в основном выделяется 4 байта (32 бита) в памяти (в самом же стандарте плюсов написано, что размер может меняться в зависимости от системы, но в дальнейшем будем всё-таки считать что 4). Тем самым нетрудно понять, что в 4 байта не влезут довольно большие целые числа. Соответственно мы работаем с неким диапазоном целых чисел $[a, b]$, где a — самое минимальное число, а b — самое максимальное. В языке C++ существуют специальные константы для a и b .

Число $a = \text{INT_MIN} = -2147483648$ — самое минимальное целое число, которое помещается в 4 байта в соответствии со стандартом. Соответственно число $b = \text{INT_MAX} = 2147483647$ — максимальное целое число, помещающееся в 4 байта. Если мы попытаемся число b увеличить на какую-то натуральную константу k , то мы получим так называемое **переполнение**. Оно работает по следующему правилу:

$$b + k = a + (k - 1)$$

Например, $b + 1 = a$. С остальными операциями переполнение работает схожим образом.

Теперь осталось найти участок программы, в котором мы можем что-то изменить переполнением. Несложно догадаться, что я не зря написал эту «пользовательскую» функцию для нахождения НОДа:

```
int Gcd(int a, int b) {
    while (a * b) {
        int rem = a % b;
        a = b;
        b = rem;
    }
    return std::max(a, b);
}
```

Листинг 1.1: «Пользовательская» функция для поиска НОДа чисел

1.2 Ошибочное место алгоритма

Если рассмотреть описанный в листинге 1.2 алгоритм в виде **формальной записи**, то на самом деле он абсолютно корректен. Но дело в том, что математические системы нельзя прямо переносить в любые языки программирования, так как из-за физической природы вычислителя они имеют свои ограничения. Обычно, ошибки в работе информационных систем делят на 4 уровня:

1. Пользовательские
2. Аппаратные
3. Программные
4. Логические

В данном случае имеется программная ошибка. Не было учтено переполнение, в следствие чего среди множества целых чисел появляются специфичные объекты, так называемые «**делители нуля**» — это ненулевые числа, произведение которых равно 0. Данные объекты обычно рассматриваются в рамках алгебраической теории множеств.

То есть, **банальная идея для решения этой задачи** — это перебрать все пары неотрицательных чисел, пока их умножение не даст 0 из-за переполнения целочисленного типа *int*, тем самым мы выйдем из цикла *while* преждевременно: функция сработает неправильно. Из-за того, что делителей нуля будет достаточно много, такой подход очень быстро сойдется (то есть хотя бы одну пару чисел для поломки программы мы сможем найти).

1.3 Формальное описание множества ответов. Полнота

Рассмотрим самый простой перебор для поиска нескольких некорректных входных данных. Он приведён на следующей странице(обратим внимание на то, что при присвоении переменной *j* значение INT_MAX нужно самостоятельно прописать выход из цикла, так как при следующей итерации *j* увеличится на 1, т.е. мы столкнёмся с переполнением, о котором говорили выше):

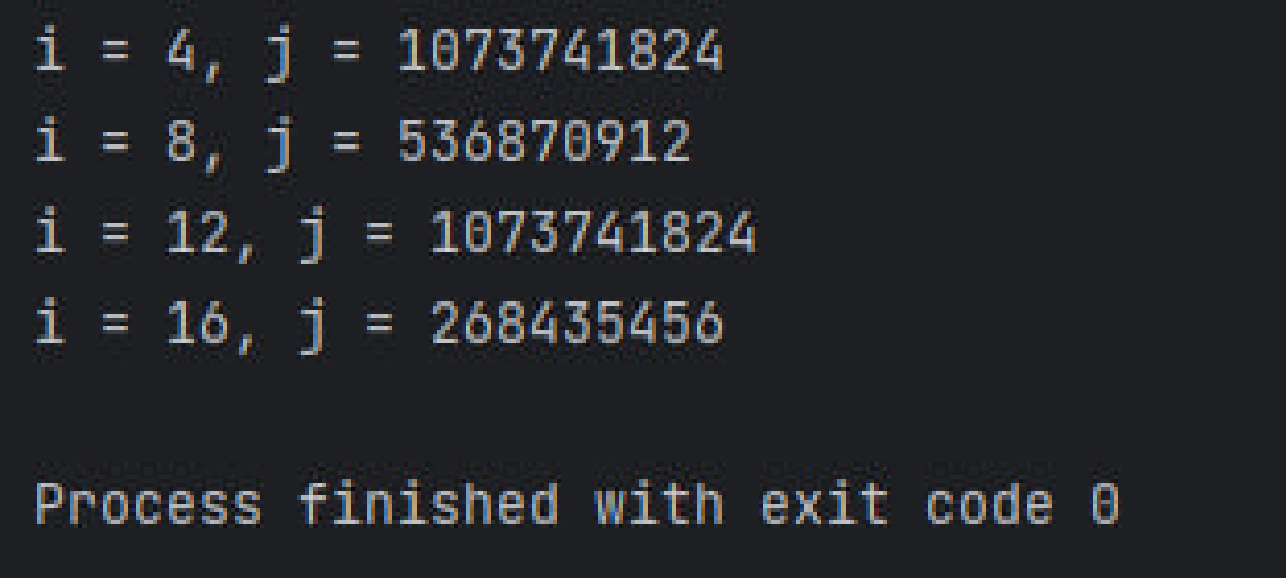
```

for (int i = 1; i <= 20; ++i) {
    for (int j = 1; j <= INT_MAX; ++j) {
        if (i * j == 0) {
            std::cout << "i = " << i << ", j = " << j << "\n";
            break;
        }
        if (j == INT_MAX) {
            if (i * j == 0) {
                std::cout << "i = " << i << ", j = " << j << "\n";
            }
            break;
        }
    }
}

```

Листинг 1.2: Перебор для поиска некорректных входных данных

Запустив код выше, получим следующий вывод:



```

i = 4, j = 1073741824
i = 8, j = 536870912
i = 12, j = 1073741824
i = 16, j = 268435456

Process finished with exit code 0

```

Рисунок 1.1 — Результат работы поиска некорректных входных данных

Сразу заметна закономерность, что одно из чисел должно делиться на 4, причем похоже что для любого числа, делящегося на 4, существует другое число, умножив на которые мы в итоге получаем 0. Попробуем понять с чем это связано.

Для начала, нам нужно избавиться от неприятного перехода в множество отрицательных чисел, так как считать в такой форме какое будет значение

после переполнения не очень удобно. Для этого представим, что все отрицательные числа — это на самом деле положительные числа после INT_MAX , то есть отобразим отрицательные числа в положительные по следующей схеме:

$$\{INT_MIN \rightarrow INT_MAX + 1\}, \dots, \{-1 \rightarrow 2 \cdot INT_MAX + 1\} \quad (1)$$

После такого преобразования, видно, что $\mathbf{a} \cdot \mathbf{b} = \mathbf{0} \Leftrightarrow \mathbf{a} \cdot \mathbf{b} \equiv \mathbf{0}(\bmod \mathbf{M})$, где $\mathbf{M} = 2 \cdot INT_MAX + 2$. Почему это так? Дело в том, что после преобразования отрицательных чисел в положительные наше максимальное значение изменилось — это $M - 1$. Если мы ещё раз к $M - 1$ прибавим единицу, то мы вернёмся в 0 (так как по формуле (1) мы «-1» преобразовали в « $M - 1$ »). Тем самым мы как раз получили определение $\bmod M$.

Раньше наша задача была следующей: «Среди множества положительных чисел типа *int* найти те, что с учетом переполнения в произведении дадут 0». Теперь же мы можем записать её формально: «Среди множества положительных чисел типа *int* найти такие a и b , что $\mathbf{a} \cdot \mathbf{b} \equiv \mathbf{0} \bmod \mathbf{M}$ ». Заметим, что множество ответов при переходе от одной формулировки к другой никак не поменялось.

Рассмотрим более подробно выражение: $\mathbf{a} \cdot \mathbf{b} \equiv \mathbf{0} \bmod \mathbf{M}$. Это означает, что $\mathbf{a} \cdot \mathbf{b} = \mathbf{k} \cdot \mathbf{M}$, $k \in \mathbb{N}$. То есть произведение чисел a и b нацело делится на число M , то есть **остаток от деления будет равен нулю**. А это в свою очередь означает, что суммарное множество делителей чисел a и b содержит множество делителей числа M .

Теперь разложим число M на эти самые делители:

$$M = 2 \cdot INT_MAX + 2 = 2^{32}$$

Тогда с учетом того, что каждое из чисел не превосходит INT_MAX , посмотрим, как могут делиться эти 2-ки между a и b . Минимальным таким делением является 2^2 и 2^{30} , так как иначе одно из чисел становится больше INT_MAX . Этот факт и объясняет что первое число делится на 4 (см. рис. 1.1).

Теперь мы можем формально описать множество ответов:

$$L = \{(a, b) | a : 2^i, b : 2^j, i + j = 32; \max\{a, b\} \leq INT_MAX\}$$

Из приведённого выше конструктивного построения множества L следует его полнота, т.е. оно описывает все возможные решения данной задачи.