Указания к решениям задач муниципального этапа Всероссийской олимпиады школьников по информатике 2018/2019 учебного года. 9 - 11 классы.

Краснодарская краевая предметно-методическая комиссия
Всероссийской олимпиады школьников по информатике

При разборе задач настоятельно рекомендуется подробно рассматривать не только алгоритмическую составляющую решений, но и математическую, в тех случаях, когда она не тривиальна. Математический аспект решения задачи, несмотря на свою простоту, ниже разобран очень подробно с достаточно высокой степенью формализма, что иногда может препятствовать восприятию школьником. При разборе очень желательно добиться сочетания строгости и доступности изложения.

Задача 1: Поиск купе

Вычислим порядковый номер купе Миши, считая от начала поезда:

$$c_1 = [(s_1 - 1)/4] + 1 + (w_1 - 1) \cdot 9,$$

здесь $[\cdot]$ - целая часть от деления, $[(s_1 - 1)/4]$ - количество купе, предшествующих Мишиному купе в Мишином вагоне, $(w_1 - 1) \cdot 9$ - количество купе во всех вагонах, предшествующих Мишиному вагону.

Аналогично, порядковый номер купе Саши вычисляется следующим образом:

$$c_2 = [(s_2 - 1)/4] + 1 + (w_2 - 1) \cdot 9.$$

Теперь осталось вычислить абсолютное значение разности этих порядковых номеров - это и будет ответом в задаче:

$$|c_1 - c_2|.$$

Решения, построенные на итерировании по номерам мест не укладываются в отведённое время и набирают неполный балл.

Задача 2: Секретное сообщение

Для решения первой подзадачи достаточно воспользоваться методом полного перебора: для каждого $i = 1, ..., n$ рассмотреть все участки подряд идущих элементов с началом в i-ом числе и среди тех из них, которые имеют сумму, кратную k, выбрать максимальный по длине (для равных длин выбирать участок с меньшим индексом начала). Если не пересчитывать для каждого участка сумму, а прибавлять очередное число к сумме участка с тем же началом, но на единицу меньшей длины, то мы получаем решение с количеством действий $O(n^2)$, что достаточно для ограничений первой подзадачи.

Для полного решения задачи необходимо поступить следующим образом. Вычислим частичные суммы по модулю k на всех префиксах исходного массива A (индексация массива с 1):

$$S[i] = \left(\sum_{j=1}^{i} A[j] \right) mod\ k.$$

При последовательном вычислении частичных сумм на это у нас потребуется порядка n действий:

$$S[i+1] = (S[i] + A[i+1])\ mod\ k.$$

Добавим частичную сумму $S[0] = 0$. Сумма участка от i-го до j-го элемента по модулю k может быть легко вычислена с использованием частичных сумм:

$$(A[i] + A[i+1] + \ldots + A[j-1] + A[j]) \mathit{mod}\, k = (S[j] - S[i-1]) \mathit{mod}\, k.$$

Теперь задача сводится к следующей: найти две одинаковые частичные суммы с максимальной разницей индексов. Это легко сделать одновременно с их вычислением, если в массиве длины k (в силу ограничения на возможное значение k такой массив может быть создан) отмечать минимальные индексы данной частичной суммы: $m[t]$ - минимальное такое i, что $S[i] = t$ ($t = 0, \ldots, k-1$, но $m[0] = 0$, поэтому нулевой элемент можно не использовать). Получаем общее количество действий O(n).

Задача 3: Дружественные цепочки

Найдём для каждого числа из данного множества сумму его собственных делителей. Чтобы найти количество собственных делителей числа N можно перебирать все числа меньшие N (или, как небольшая оптимизация, меньшие $[N/2]$) и учитывать те из них, на которые делится число N. Такой алгоритм потребует порядка N действий, и с его использованием можно построить решение только подзадач 1 и 2. Для решения подзадач 3 и 4 требуется более эффективный алгоритм вычисления количества собственных делителей: перебирается только меньший из делителей, он, очевидно, не превосходит $\left[\sqrt{N}\right]$. Вместе с меньшим делителем учитываетя и парный к нему - тот, произведение на который даёт N. Отдельно нужно обработать ситуацию, когда N - точный квадрат, в этом случае делитель $\left[\sqrt{N}\right]$ нужно учесть только один раз. Количество действий этого алгоритма, очевидно, $O\left(\sqrt{N}\right)$.

Сравнивая количество собственных делителей с самим числом, можно получить решение первой подзадачи. Для решения второй подзадачи достаточно вычислить для каждого числа сумму его собственных делителей и сравнить его со всеми числами. Если находим равенство с другим числом, то для этого числа вычисляем сумму собственных делителей и сравниваем с исходным числом. Если снова равенство - мы нашли пару дружественных чисел. Количество действий при вычислении суммы собственных делителей за $O(N)$ действий - $O\left(n \cdot N + n^2\right)$, где N - ограничение на величину чисел, что позволяет получить решение только второй подазадачи. Если же воспользоваться более эффективным алгоритмом вычисления суммы собственных делителей за $O\left(\sqrt{N}\right)$ действий, получаем общее количество действий $O\left(n \cdot \sqrt{N} + n^2\right)$, что позволяет решить третью подзадачу. Кроме того, данное решение можно немного ускорить (примерно вдвое), если отсортировать числа и воспользоваться бинарным поиском для нахождения числа, потенциально парного данному. Кроме того, саму сумму собственных делителей для каждого числа имеет смысл вычислить только один раз и сохранить в массиве. Количество действий: $O\left(n \cdot \sqrt{N} + n \log n\right)$.

Для полного решения задачи произведём следующее построение. Пусть $A[i]$ - отсортированный массив исходных чисел. Построим следующий массив B: $B[i] = j$, если $\sigma(A[i]) = A[j]$, где $\sigma(a)$ - сумма собственных делителей числа a, если такого j не существует, то $B[i]$ положим равным, например, -1. Для вычисления значений массива B для каждого элемента массива A вычислим сумму собственных делителей и воспользуемся бинарным поиском для определения соответствующего индекса j. После вычисления массива B осталось произвести поиск циклов: стартуем с очередного не пройденного ранее при поиске циклов индекса и проходим по элементам массива $B : i, B[i], B[B[i]], B[B[B[i]]], \ldots$ пока не зайдём в тупик, либо не придём в ранее посещённый индекс. Причём, если мы попадём в индекс, посещённый ранее при старте с другого индекса, то это не даёт нам ничего нового: это либо тупиковая ветвь, либо ранее найденный цикл. А если мы попадём в индекс, посещённый в текущий проход по цепочке, то мы нашли цикл. Осталось только найти его длину - надо снова стартовать с найденного индекса и двигаться, пока не дойдём до него же. Чтобы отличать индексы, исследованные в предыдущих проходах от индексов, пройденных в текущем проходе, необходимо отмечать в массиве посещений индексов индексы при помощи стартового индекса. При обнаружении очередного более длинного цикла необходимо запоминать его длину и любой его элемент, с которого впоследствии можно восстановить последовательные индексы цикла (а соответственно и сами числа).

Выпишем алгоритм:

1. Сортируем массив исходных чисел A.

2. Для каждого $A[i]$ вычисляем $\sigma(A[i])$ и при помощи бинарного поиска определяем такое j, что $\sigma(A[i]) = A[j]$, либо определяем, что такого не существует и кладём $j = -1$. Кладём $B[i] = j$.

3. Инициализируем массив признаков рассмотренных чисел: $V[i] = -1$ для всех i, что означает, что i-е число ещё не рассмотрено.

4. **Цикл** по всем $V[i]$:

   (a) Отмечаем, что данный индекс уже рассмотрен: кладём $V[i] = i$.

   (b) Переходим к возможному следующему индексу в цепочке: кладём $j = B[i]$.

   (c) **Пока** $j \neq -1$ И $V[j] = -1$:

      i. Отмечаем, что данный индекс уже рассмотрен: кладём $V[j] = i$ (важно, что он рассмотрен именно при старте с индекса i).

      ii. Переходим к возможному следующему индексу в цепочке: кладём $j = B[j]$.

   (d) **Если** $j \neq -1$ И $V[j] = i$, то есть мы нашли новую цепочку:

      i. Вычисляем длину цикла: кладём $t = B[j]$ и в цикле проводим операцию $t = B[t]$, пока $t \neq j$.

      ii. Если длина найденной цепочки больше предыдущей максимальной длины, запоминаем эту новую максимальную длину и j как индекс одного из элементов цепочки.

5. Выводим найденную максимальную длину и восстанавливаем цепочку этой максимальной длины, начиная с запомненного индекса одного из её элементов.

На языке C код центральной части алгоритма выглядит следующим образом:

```
maxlength = 0;
maxchainstart = -1;

for(i = 0; i < n; i++)
{
        j = i;
        while(j != -1 && V[j] == -1)
        {
                done[j] = i;
                j = B[j];
        }
        if(j != -1 && V[j] == i)
        {
                t = B[j];
                curlength = 1;
                while(t != j)
                {
                        t = B[t];
                        curlength++;
                }
                if(curlength > maxlength)
                {
                        maxlength = curlength;
                        maxchainstart = j;
                }
        }
}
```

Задача 4: Конное путешествие

Основу решения задачи составляет известный алгоритм обхода графа в ширину: обход начинается в одной вершине, после этого исследуются все её дочерние вершины, после этого все их дочерние вершины и так далее. Классическая реализация данного алгоритма основана на использовании очереди обрабатываемых вершин. Однако для решения первой подзадачи можно реализовать обход в ширину и без использования очереди: отмечаем на данном прямоугольнике в клетках расстояния, измеряемое в ходах коня, от исходной клетки. Для этого отметим сначала саму исходную клетку числом 0. После этого отметим все клетки, в которые из неё можно попасть числом 1. Далее просматриваем весь прямоугольник и если находим число 1 - отмечаем все ранее не отмеченные клетки, в которые можно попасть из данной, числом 2. И так далее: на k-ом шаге полностью просматриваем прямоугольник и, когда находим клетку с числом k, отмечаем все ранее не отмеченные клетки, в которые из неё можно попасть, числом $k+1$. Максимальное расстояние ограничивается общим числом клеток в прямоугольнике, поэтому общее количество действий при таком подходе $O(M^2 \cdot M^2) = O(M^4)$.

Для решения второй подзадачи необходимо научиться определять, принадлежит ли клетка территории Чессляндии. Для этого можно построить карту страны на прямоугольнике $[0, M-1] \times [0, M-1]$ - отметить все клетки, принадлежащий територии страны числом 1, а не принадлежащие - числом 0. В остальном достаточно реализовать приведённый выше алгоритм. Количество действий по-прежнему $O(M^4)$.

В третьей подзадаче для контроля принадлежности клетки территории страны можно поступить точно таким же образом. При этом, чтобы уложиться в ограничения по памяти, двумерный массив с картой должен состоять из однобайтовых целых. На составление карты потребуется $O(n \cdot m^2)$ действий. Для решения этой подзадачи реализацию обхода в ширину необходимо делать уже с помощью очереди (отсылаем в этом месте к учебной литературе, например А. Шень "Программирование. Теоремы и задачи"). Это потребует ещё $O(n \cdot m^2)$ действий, поэтому общая ассимптотика количества операций $O(n \cdot m^2)$.

В четвёртой подзадаче составить карту на двумерном массиве уже не получится в силу возможного большого разброса координат. Однако самих прямоугольников не очень много, поэтому определять принадлежность клетки территории Чессляндии можно простым перебором всех прямоуголников, из которых состоит территория страны. При реализации обхода в ширину при помощи очереди это сумарно потребует $O(n \cdot S)$ действий, где S - площадь территории страны.

Осталось отметить, что для совместного решения подзадач 3 и 4 необходимо реализовать оба подхода и в случае $n \leq 100$ применять алгоритм подазадачи 4, а иначе - алгоритм подзадачи 3 (будучи точно уверенным, что входные данные соответствуют ограничениям 3-ей подзадачи).