Муниципальный этап Всероссийской олимпиады школьников по информатике

в 2016 – 2017 учебном году

Разборы решений и идеи тестов

Муниципальный этап Всероссийской олимпиады школьников по информатике в 2016 – 2017 учебном году

10 класс

Время выполнения задач — 4 часа

Ограничение по времени — 2 секунды на тест

Ограничение по памяти — 64 мегабайта

10.1. «Прибавляем единицу». Имеется четырёхзначное натуральное число, не содержащее в своей десятичной записи цифру 9. Какое число получится, если к цифре в каждом разряде прибавить единицу?

Формат входа: На входе задаётся десятичная запись единственного натурального исходного числа.

Формат выхода: Выдайте десятичную запись натурального числа, получающегося в результате указанной операции.

Пример

Вход:	*Выход:*
1234	2345

10.2. «Упрощение числа». Петя Торопыжкин на математике изучил разложение целого числа на простые множители. Он придумал операцию, которую назвал *упрощение числа*: число раскладывается на простые множители, после чего находится произведение всех его различных простых множителей в первой степени. Напишите программу, которая осуществляет данную операцию.

Формат входа: В единственной входной строке задано натуральное число n ($1 \leqslant n \leqslant 30\,000$).

Формат выхода: Выведите единственное натуральное число — результат применения Петиной операции к числу n.

Пример 1

Вход:	*Выход:*
7	7

Пример 2

Вход:	*Выход:*
500	10

Примечание: В первом примере $7 = 7^1 \to 7$, во втором $500 = 2^2 \cdot 5^3 \to 2 \cdot 5 = 10$.

10.3. «Муравьиный шифр». Первым уроком у Пети Торопыжкина была биология, его класс изучал муравьёв. На втором уроке, на математике, рассказывали про разные шифры. На большом перерыве после математики Петя придумал интересный шифр, которым могли бы воспользоваться муравьи. А именно: берём муравьёв по количеству символов в сообщении, выстраиваем их по росту (от более длинных к более коротким) и сообщаем каждому по порядку очередную

букву из сообщения, после чего вся команда посылается к месту получения сообщения. Там они снова выстраиваются по росту, и каждый по очереди называет свою букву. (Кстати, такой шифр был бы устойчивым к потере небольшого числа посыльных!) На третьем уроке, на информатике, он захотел было написать соответствующую программу, которая по информации о росте посыльных муравьёв и их буквах восстанавливала бы закодированное сообщение, но отвлёкся на другие задачи. Напишите такую программу вы.

Формат входа: В первой строке задано целое число n — количество посыльных муравьёв ($1 \leqslant n \leqslant 10^5$). В следующих n строках задана информация о посыльных муравьях (в каком-то порядке!): через пробел натуральное число l_i — рост i-го посыльного ($1 \leqslant l_i \leqslant 10^9$) — и c_i — символ, который ему сообщили. Считаем, что все c_i — заглавные символы латиницы, а все l_i попарно различны.

Формат выхода: Выведите единственную строку — сообщение, переданное этим набором посыльных.

Пример

Вход:	*Выход:*
3 15 A 253 B 77 C	BCA

10.4. «Ещё один шифр». После уроков Петя Торопыжкин придумал ещё один шифр. Пусть слова исходного сообщения содержат только заглавные символы латиницы. К концу каждого слова зеркально приписываем его само, получая палиндром. Затем, возможно, дописываем в начало и в конец какие-то заглавные буквы латиницы так, чтобы запутать противника, но при этом не испортить ситуации, что полученный на первом шаге палиндром был максимальным по длине среди первых по вхождению (то есть среди всех, начинающихся не правее любых других подпалиндромов). Для раскодирования такой шифровки Пете нужна программа, которая для каждого слова из полученного сообщения найдёт первый по вхождению подпалиндром чётной длины и выведет первую его половину. Или определит, что в данной строке нет подпалиндромов чётной длины (такие слова могут добавляться случайным образом для усложнения взлома). Конечно же, в шифровка считается корректной, и в ней присутствует хотя бы одно слово, которое содержит подпалиндром чётной длины, поскольку какая-то полезная информация в шифровке должна быть.

Формат входа: В первой строке содержится натуральное число n, не превосходящее 1000 — количество слов в шифровке. Каждая из последующих n строк содержит по одному слову из шифровки. Каждое слово есть непустая строка, состоящая только из заглавных символов латиницы. Сумма длин всех строк не превосходит 10^4 символов.

Формат выхода: По одному на строке выдайте слова расшифровки.

Пример

Вход:	*Выход:*
3 GCGOODDOOGB ABCDEFG HGDEEDDEEDWWWWWWWWWW	GOOD DEED

Примечание: В первом слове самый левый палиндром GCG не может быть искомым, так как имеет нечётную длину. В третьем слове палиндром DEED не может быть искомым, так как не имеет наибольшую длину среди всех палиндромов, начинающихся с третьего символа. Палиндром в третьем слове, состоящий из букв W, не может быть искомым, так как имеются палиндромы, входящие левее его.

10.5. «Ломаем и едим». Пете Торопыжкину попалась в руки прямоугольная шоколадка размером $n \times m$ долек. На дольке с координатами (i, j), $i = 1 \ldots, n$ — номер строки, $j = 1, \ldots, m$ — номер столбца, написано число $a_{ij} = (p^i \cdot q^j) \bmod r$, где p, q, r — некоторые различные простые числа. Разглядывая эту шоколадку, он придумал следующую игру: двое по очереди отламывают от шоколадки полоски сверху или справа (со стороны бóльших индексов i или j) и съедают их. Полоски могут включать один или несколько рядов или столбцов долек. При этом долька $(1, 1)$ может быть съедена только одна, а не в составе какой-либо полоски. (В частности, запрещается первому игроку первым ходом съесть всю шоколадку.) Каждый игрок суммирует числа на съеденных им дольках. Петя задался следующим вопросом относительно этой игры: а чему равна сумма результатов первого игрока в такой игре по всем возможным партиям? Напишите программу, которая по данным об игре будет вычислять остаток от деления этой величины на $10^9 + 7$ (остаток — поскольку сама сумма может быть очень большим числом).
Внимание! Ограничение по памяти для этой задачи составляет **256 мегабайт**!

Формат входа: В единственной входной строке через пробел заданы пять натуральных чисел: n, m — размеры шоколадки, количество её столбцов и строк ($1 \leqslant n, m \leqslant 1500$), p, q, r — параметры заполнения шоколадки числами (это простые числа, не превосходящие 10^4).

Формат выхода: Выдайте целое число — остаток от деления на $10^9 + 7$ суммы результатов первого игрока в такой игре по всем возможным партиям.

Пример

Вход:	*Выход:*
2 3 3 2 1009	642

Примечание: В данной игре шоколадка выглядит как

18 ($= 3^2 \cdot 2^1 \bmod 1009$)	**36** ($= 3^2 \cdot 2^2 \bmod 1009$)	**72** ($= 3^2 \cdot 2^3 \bmod 1009$)
6 ($= 3^1 \cdot 2^1 \bmod 1009$)	**12** ($= 3^1 \cdot 2^2 \bmod 1009$)	**24** ($= 3^1 \cdot 2^3 \bmod 1009$)

(жирным выделены числа на дольках; в скобках показано, как эти числа получаются). В такой игре возможно пять различных партий:

1) $\begin{array}{|c|c|c|}\hline 18 & 36 & 72 \\ \hline 6 & 12 & 24 \\ \hline\end{array} \xrightarrow{(1)} \begin{array}{|c|c|}\hline 18 & 36 \\ \hline 6 & 12 \\ \hline\end{array} \xrightarrow{(2)} \begin{array}{|c|}\hline 18 \\ \hline 6 \\ \hline\end{array} \xrightarrow{(1)} \boxed{6} \xrightarrow{(2)} \varnothing$ (114)

2) $\begin{array}{|c|c|c|}\hline 18 & 36 & 72 \\ \hline 6 & 12 & 24 \\ \hline\end{array} \xrightarrow{(1)} \begin{array}{|c|c|}\hline 18 & 36 \\ \hline 6 & 12 \\ \hline\end{array} \xrightarrow{(2)} \begin{array}{|c|c|}\hline 6 & 12 \\ \hline\end{array} \xrightarrow{(1)} \boxed{6} \xrightarrow{(2)} \varnothing$ (108)

3) $\begin{array}{|c|c|c|}\hline 18 & 36 & 72 \\ \hline 6 & 12 & 24 \\ \hline\end{array} \xrightarrow{(1)} \begin{array}{|c|}\hline 18 \\ \hline 6 \\ \hline\end{array} \xrightarrow{(2)} \boxed{6} \xrightarrow{(1)} \varnothing$ (150)

4) $\begin{array}{|c|c|c|}\hline 18 & 36 & 72 \\ \hline 6 & 12 & 24 \\ \hline\end{array} \xrightarrow{(1)} \begin{array}{|c|c|c|}\hline 6 & 12 & 24 \\ \hline\end{array} \xrightarrow{(2)} \begin{array}{|c|c|}\hline 6 & 12 \\ \hline\end{array} \xrightarrow{(1)} \boxed{6} \xrightarrow{(2)} \varnothing$ (138)

5) $\begin{array}{|c|c|c|}\hline 18 & 36 & 72 \\ \hline 6 & 12 & 24 \\ \hline\end{array} \xrightarrow{(1)} \begin{array}{|c|c|c|}\hline 6 & 12 & 24 \\ \hline\end{array} \xrightarrow{(2)} \boxed{6} \xrightarrow{(1)} \varnothing$ (132)

Сумма результатов первого игрока, приведённых в скобках в конце каждой строчки, равна 642.

Муниципальный этап Всероссийской олимпиады школьников по информатике в 2016 – 2017 учебном году
10 класс. Разбор решений и идеи тестов

10.1. «Прибавляем единицу». *Имеется четырёхзначное натуральное число, не содержащее в своей десятичной записи цифру 9. Какое число получится, если к цифре в каждом разряде прибавить единицу?*

Собственно, для решения задачи нужно понять, что ответ есть сумма исходного числа и 1111, и нет нужды производить разложение числа на цифры и другие сложные операции с данным числом.

Идеи тестов:

1–10. Случайные тесты.

10.2. «Упрощение числа». *Петя Торопыжкин на математике изучил разложение целого числа на простые множители. Он придумал операцию, которую назвал* упрощение числа*: число раскладывается на простые множители, после чего находится произведение всех его различных простых множителей в первой степени. Напишите программу, которая осуществляет данную операцию.*

Решение этой задачи несложно, хотя требует некоторой технической аккуратности при работе с числами.

Идея решения состоит в следующем. Начиная с 2, ищем простые делители числа — отдельно проверяем двойку, а затем идём по нечётным числам от 3 и выше. Найдя очередной делитель числа, исключаем его из разложения — делим число на него, пока делится. Результирующая переменная, которая вначале равна 1, умножается на найденный делитель. Процесс заканчивается, когда текущее обрабатываемое число становится равным 1.

Идеи тестов:

1. $n = 1$.
2. $n = 2$.
3. n — степень двойки.
4. n — нечетное простое число.
5. n — степень семерки.
6. n — чётное число, произведение простых чисел в первых степенях.
7. n — нечётное число, произведение простых чисел в первых степенях.
8. n — чётное число, произведение простых чисел в разных степенях (некоторые — в первой, включая двойку).
9. n — чётное число, произведение простых чисел в разных степенях (некоторые — в первой, двойка — не в первой).

10. n — нечётное число, произведение простых чисел в разных степенях (некоторые — в первой).
11. n — чётное число, произведение простых чисел в разных степенях, бóльших 1.
12. n — нечётное число, произведение простых чисел в разных степенях, бóльших 1.
13. $n = 30\,000$.

14–20. Случайные тесты.

10.3. «Муравьиный шифр». *Первым уроком у Пети Торопыжкина была биология, его класс изучал муравьёв. На втором уроке, на математике, рассказывали про разные шифры. На большом перерыве после математики Петя придумал интересный шифр, которым могли бы воспользоваться муравьи. А именно: берём муравьёв по количеству символов в сообщении, выстраиваем их по росту (от более длинных к более коротким) и сообщаем каждому по порядку очередную букву из сообщения, после чего вся команда посылается к месту получения сообщения. Там они снова выстраиваются по росту, и каждый по очереди называет свою букву. (Кстати, такой шифр был бы устойчивым к потере небольшого числа посыльных!) На третьем уроке, на информатике, он захотел было написать соответствующую программу, которая по информации о росте посыльных муравьёв и их буквах восстанавливала бы закодированное сообщение, но отвлёкся на другие задачи. Напишите такую программу вы.*

Данная задача несложна идеологически, но требует написания процедуры сортировки (или использования соответствующей библиотечной процедуры). При этом сортировать надо сложные объекты: пары число–символ (по величине числа). Кроме того, требуется быстрая сортировка, так как на больших тестах (более 14-15 тысяч входных элементах) неоптимальные сортировки (типа пузырьковой) не пройдут по времени. Участники, написавшие неоптимальную сортировку, получат около двух третей полного балла.

Идеи тестов:

1. $n = 1$.

2–13. Случайные тесты, $n \leqslant 10000$.

14. Максимальный тест, $n = 10^5$.

15–20. Случайные тесты, $n \geqslant 20000$.

10.4. «Ещё один шифр». *После уроков Петя Торопыжкин придумал ещё один шифр. Пусть слова исходного сообщения содержат только заглавные символы латиницы. К концу каждого слова зеркально приписываем его само, получая палиндром. Затем, возможно, дописываем в начало и в конец какие-то заглавные буквы латиницы так, чтобы запутать противника, но при этом не испортить ситуации, что полученный на первом шаге палиндром был максимальным по*

длине среди первых по вхождению (то есть среди всех, начинающихся не правее любых других подпалиндромов). Для раскодирования такой шифровки Пете нужна программа, которая для каждого слова из полученного сообщения найдёт первый по вхождению подпалиндром чётной длины и выведет первую его половину. Или определит, что в данной строке нет подпалиндромов чётной длины (такие слова могут добавляться случайным образом для усложнения взлома). Конечно же, в шифровка считается корректной, и в ней присутствует хотя бы одно слово, которое содержит подпалиндром чётной длины, поскольку какая-то полезная информация в шифровке должна быть.

По мнению программного комитета, данная задача имеет сложность выше среднего.

Безусловно, имеется лобовое решение, связанное с прямолинейным перебором подстрок чётной длины в соответствующем порядке и проверкой их на симметричность. Однако такое решение имеет кубическую сложность под длине строки: перебираются $n-1$ позиция середины палиндрома, полудли́ны палиндромов от 1 до $n/4$ (в среднем), и проводится проверка симметричности соответствующих подстрок, что тоже требует в среднем до $n/4$ сравнений. Поэтому при имеющихся ограничениях лобовое решение может не пройти на больших тестах.

Важно обратить внимание на слово «может»! Дело в том, что O-оценки чаще всего связаны с наихудшим случаем. В нашем случае наихудший случай проявляется, когда мы ищем палиндром какой-то длины $2k$, и сравнения первых $(k-1)$-й пары символов удачны, а символы в последней паре различны. Но такая ситуация в «пёстрой» строке, где символы подобраны случайно, вряд ли может произойти стабильно при поиске палиндромов в различных местах. Чаще всего очень быстро будет найдена пара несовпадающих символов, и процесс проверки подстроки на симметричность закончится существенно раньше, чем будет достигнута длина подстроки. Исключение составляют строки, в которых есть подстроки существенной длины, каждая из одинаковых символов. На таких строка проверка симметричности требует полного или почти полного времени. В корпусе тестов есть соответствующие примеры.

Гарантированно более оптимальным является алгоритм, позволяющий за квадратичное время определить максимальные длины палиндромов, имеющих середину в каждой из возможных позиций. Он базируется на том, что всякая подстрока палиндрома, симмеричная относительно позиции его центра, тоже является палиндромом: в палиндроме `ABCDDCBA` подстроки `DD`, `CDDC` и `BCDDCB` также являются палиндромами. Поэтому правильным порядком перебора являются следующих: перебираем возможные позиции центра палиндрома, а дальше последовательными сравнениями символов, расположенных в строке симметрично относительно этой позиции, находим максимальную длину палиндрома с центром в выбранной позиции. Затем для решения нашей задачи по полученной информации выбираем палиндром, начинающийся левее других и имеющий максимальную длину среди

всех, начинающихся в позиции его начала.

Идеи тестов:

1. Все слова — чистые палиндромы без добавления букв, искомый палиндром не содержит подпалиндромов, не центрированных с ним.
2. Все слова — чистые палиндромы без добавления букв, искомый палиндром содержит подпалиндромы, не центрированные с ним, но начинающиеся не с его начала.
3. Все слова — чистые палиндромы без добавления букв, искомый палиндром содержит подпалиндромы, не центрированные с ним и начинающиеся с его начала.
4. Все слова получены добавлением букв в начало, добавленные комбинации не дают новых палиндромов, искомый палиндром не содержит подпалиндромов, не центрированных с ним.
5. Все слова получены добавлением букв в начало, добавленные комбинации не дают новых палиндромов, искомый палиндром содержит подпалиндромы, не центрированные с ним, но начинающиеся не с его начала.
6. Все слова получены добавлением букв в начало, добавленные комбинации не дают новых палиндромов, искомый палиндром содержит подпалиндромы, не центрированных с ним и начинающиеся с его начала.
7. Все слова получены добавлением букв в начало, добавленные комбинации внутри себя содержат палиндромы нечетной длины, искомый палиндром не содержит подпалиндромов, не центрированных с ним.
8. Все слова получены добавлением букв в начало, добавленные комбинации внутри себя содержат палиндромы нечетной длины, искомый палиндром содержит подпалиндромы, не центрированные с ним, но начинающиеся не с его начала.
9. Все слова получены добавлением букв в начало, добавленные комбинации внутри себя содержат палиндромы нечетной длины, искомый палиндром содержит подпалиндромы, не центрированных с ним и начинающиеся с его начала.
10. Все слова получены добавлением букв в начало, добавленные комбинации образуют с началом искомого палиндромы нечетной длины, искомый палиндром не содержит подпалиндромов, не центрированных с ним.
11. Все слова получены добавлением букв в начало, добавленные комбинации образуют с началом искомого палиндромы нечетной длины, искомый палиндром содержит подпалиндромы, не центрированные с ним, но начинающиеся не с его начала.
12. Все слова получены добавлением букв в начало, добавленные комбинации образуют с началом искомого палиндромы нечетной длины, искомый палиндром содержит подпалиндромы, не центрированных с ним и начинающиеся с его начала.

13. Все слова получены добавлением букв в конец, добавленные комбинации не дают новых палиндромов, искомый палиндром не содержит подпалиндромов, не центрированных с ним.
14. Все слова получены добавлением букв в конец, добавленные комбинации не дают новых палиндромов, искомый палиндром содержит подпалиндромы, не центрированные с ним, но начинающиеся не с его начала.
15. Все слова получены добавлением букв в конец, добавленные комбинации не дают новых палиндромов, искомый палиндром содержит подпалиндромы, не центрированных с ним и начинающиеся с его начала.
16. Все слова получены добавлением букв в конец, добавленные комбинации внутри себя содержат палиндромы нечетной длины, искомый палиндром не содержит подпалиндромов, не центрированных с ним.
17. Все слова получены добавлением букв в конец, добавленные комбинации внутри себя содержат палиндромы нечетной длины, искомый палиндром содержит подпалиндромы, не центрированные с ним, но начинающиеся не с его начала.
18. Все слова получены добавлением букв в конец, добавленные комбинации внутри себя содержат палиндромы нечетной длины, искомый палиндром содержит подпалиндромы, не центрированных с ним и начинающиеся с его начала.
19. Все слова получены добавлением букв в конец, добавленные комбинации внутри себя содержат палиндромы четной длины, искомый палиндром не содержит подпалиндромов, не центрированных с ним.
20. Все слова получены добавлением букв в конец, добавленные комбинации внутри себя содержат палиндромы четной длины, искомый палиндром содержит подпалиндромы, не центрированные с ним, но начинающиеся не с его начала.
21. Все слова получены добавлением букв в конец, добавленные комбинации внутри себя содержат палиндромы четной длины, искомый палиндром содержит подпалиндромы, не центрированных с ним и начинающиеся с его начала.
22. Все слова получены добавлением букв в конец, добавленные комбинации образуют с концом искомого палиндромы нечетной длины, искомый палиндром не содержит подпалиндромов, не центрированных с ним.
23. Все слова получены добавлением букв в конец, добавленные комбинации образуют с концом искомого палиндромы нечетной длины, искомый палиндром содержит подпалиндромы, не центрированные с ним, но начинающиеся не с его начала.
24. Все слова получены добавлением букв в конец, добавленные комбинации образуют с концом искомого палиндромы нечетной длины, искомый палиндром содержит подпалиндромы, не центрированных с ним и начинающиеся с его начала.

25. Все слова получены добавлением букв в конец, добавленные комбинации образуют с концом искомого палиндромы четной длины, искомый палиндром не содержит подпалиндромов, не центрированных с ним.
26. Все слова получены добавлением букв в конец, добавленные комбинации образуют с концом искомого палиндромы четной длины, искомый палиндром содержит подпалиндромы, не центрированные с ним, но начинающиеся не с его начала.
27. Все слова получены добавлением букв в конец, добавленные комбинации образуют с концом искомого палиндромы четной длины, искомый палиндром содержит подпалиндромы, не центрированных с ним и начинающиеся с его начала.

28–32. Случайные короткие тесты, основной палиндром состоит из одинаковых символов, слова включают дополнительные буквы как в начале слов, так и в конце.

33–50. Случайные длинные тесты, включающие ситуации, описанные выше.

Примечание: Тесты 1–27 являются короткими, то есть берутся лобовым решением. Во всех тестах присутствуют дополнительные слова, не содержащие палиндромов чётной длины.

10.5. «Ломаем и едим». *Пете Торопыжкину попалась в руки прямоугольная шоколадка размером $n \times m$ долек. На дольке с координатами (i, j), $i = 1 \ldots, n$ — номер строки, $j = 1, \ldots, m$ — номер столбца, написано число $a_{ij} = (p^i \cdot q^j) \bmod r$, где p, q, r — некоторые различные простые числа. Разглядывая эту шоколадку, он придумал следующую игру: двое по очереди отламывают от шоколадки полоски сверху или справа (со стороны бóльших индексов i или j) и съедают их. Полоски могут включать один или несколько рядов или столбцов долек. При этом долька $(1, 1)$ может быть съедена только одна, а не в составе какой-либо полоски. (В частности, запрещается первому игроку первым ходом съесть всю шоколадку.) Каждый игрок суммирует числа на съеденных им дольках. Петя задался следующим вопросом относительно этой игры: а чему равна сумма результатов первого игрока в такой игре по всем возможным партиям? Напишите программу, которая по данным об игре будет вычислять остаток от деления этой величины на $10^9 + 7$ (остаток — поскольку сама сумма может быть очень большим числом).*

Данная задача, безусловно, имеет крайне высокую сложность, связанную как с осознанием условия, так и с разработкой оптимального алгоритма, так и с его реализацией.

Лобовое решение, связанное с последовательным перебором возможных ходов первого игрока, затем второго, затем снова первого, и т.д. имеет показательную сложность и работает лишь только для очень маленьких n и m; реальное ограничение $n + m \approx 15 \div 16$. Как обычно, проблема в том, что некоторые ситуации

анализируются многократно. Что естественным образом приводит к идее использования принципа динамического программирования.

Обозначим величину $10^9 + 7$, по модулю которой должен быть вычислен результат, через M.

В дальнейшем будем считать, что знак «+» обозначает сложение по модулю M, чтобы не загромождать записи операциями взятия модуля. Слово «сумма» будет означать «сумма по модулю M». Кроме того, под «произведением» также будем понимать «произведение по модулю M», которое также будем в выкладках обозначать как обычное произведение.

Здесь кроется один из тонких технических моментов. Если сумму по модулю можно реализовать непосредственно:

```
const modulo = 1000000007;
function addMod_(a,b: integer): integer;
begin
  addMod_ := (a+b) mod modulo;
end;
```

то такое же прямолинейное определение функции произведения по модулю

```
function mulMod_(a,b: integer): integer;
begin
  mulMod_ := (a*b) mod modulo;
end;
```

неправильно, поскольку каждое из чисел может быть порядка 10^9, а их произведение выйдет за пределы стандартного 4-байтного целого типа. Поэтому надо ввести вспомогательные переменные 8-байтного целого типа (`int64` в Паскале, `__int64` в C++), присвоить им значения перемножаемых переменных и только после этого находить произведение и остаток от его деления.

Введем двумерный массив `a[i,j,k]`, `i` $= 1..n$, `j` $= 1..m$, `k` $= 1..2$. Каждая ячейка этого массива хранит две целых величины (взятых/вычисленных по модулю M). Первая `a[i,j,1].value` — сумма чисел, съеденных первым игроком, по всем партиям, где после хода первого игрока от исходной шоколадки остался прямоугольник 1..`i` $\times$ 1..`j` к тому моменту, когда такой прямоугольник получился. Вторая `a[i,j,1].qnt` — количество таких партий. Аналогично, ячейка `a[i,j,2]` хранит информацию о суммарном результате первого игрока по партиям, в которых прямоугольник 1..`i`$\times$1..`j` получился после хода второго игрока, и о количестве таких партий.

В целом, понятно, зачем хранить поля `value` — они накапливают информацию, по которой будет получен искомый результат. Но зачем хранить количество партий? Рассмотрим ситуацию, когда после хода второго игрока остался какой-то прямоугольник 1..`i` $\times$ 1..`j`. Пусть теперь первый игрок делает ход, съедая одну горизонтальную полоску, оставляя прямоугольник 1..(`i` $-$ 1) $\times$ 1..`j` и получая сколько-то очков. И важно понять, что этот ход он мог сделать во всех партиях, в которых был прямоугольник 1..`i` $\times$ 1..`j` после хода второго игрока. Соответствен-

но, вклад этого хода в общий результат есть произведение полученных очков на количество партий, в которых такой ход мог быть сделан. И именно для вычисления этого произведения нужно хранить количество партий, в которых после хода второго игрока мог возникнуть прямоугольник $1..\mathtt{i} \times 1..\mathtt{j}$.

Теперь обсудим начальные условия метода динамического программирования. Это несложно:

```
a[n,m,1].value := 0;
a[n,m,1].qnt := 0;
a[n,m,2].value := 0;
a[n,m,2].qnt := 1;
```

Мы начинаем с шоколадки размера $n \times m$, поэтому положим условия для начального состояния. Начальная ситуация не могла возникнуть после хода первого игрока — перед ходом второго, поэтому `a[n,m,1].qnt := 0`. Но она возникла единственным образом перед ходом первого игрока, то есть как бы после хода второго, поэтому `a[n,m,2].qnt := 1`. В начальный момент первый игрок ещё ничего не съел, поэтому поля `value` инициализируем нулями.

Теперь о пересчете метода динамического программирования. Пусть какие-то ячейки в правом верхнем углу уже просчитаны (какие точно нужны, обсудим чуть попозже). Хочется просчитать информацию в ячейках `a[i,j,1]` и `a[i,j,2]` для каких-то `i` и `j`. Для этого нужно посмотреть, из каких позиций и какими ходами мы могли получить прямоугольник $1..\mathtt{i} \times 1..\mathtt{j}$. Несложно понять, что такой прямоугольник мог получиться из прямоугольника с бóльшим числом строк $\mathtt{i'} > \mathtt{i}$ съедением горизонтальных полосок от $(\mathtt{i}+1)$-й до `i'`-й. Либо из прямоугольника с бóльшим числом столбцов $\mathtt{j'} > \mathtt{j}$ съедением вертикальных полосок от $(\mathtt{j}+1)$-й до `j'`-й. При этом, если ходил первый игрок, то его результат увеличился соответствующим образом, а если ходил второй, то результат первого не увеличился. Таким образом, имеем

$$\begin{aligned}
&\mathtt{a[i,j,1].value} = \\
&\qquad = \sum_{\mathtt{i'}=\mathtt{i}+1}^{n} \Big(\mathtt{a[i',j,2].value} + \mathtt{a[i',j,2].qnt} \cdot \mathrm{Val}\left[(\mathtt{i'},\mathtt{j}) \to (\mathtt{i},\mathtt{j})\right]\Big) + \\
&\qquad + \sum_{\mathtt{j'}=\mathtt{j}+1}^{m} \Big(\mathtt{a[i,j',2].value} + \mathtt{a[i,j',2].qnt} \cdot \mathrm{Val}\left[(\mathtt{i},\mathtt{j'}) \to (\mathtt{i},\mathtt{j})\right]\Big), \quad (1)
\end{aligned}$$

$$\mathtt{a[i,j,1].qnt} = \sum_{\mathtt{i'}=\mathtt{i}+1}^{n} \mathtt{a[i',j,2].qnt} + \sum_{\mathtt{j'}=\mathtt{j}+1}^{m} \mathtt{a[i,j',2].qnt}. \quad (2)$$

То есть для получения результата первого игрока по всем партиям, проходящим после хода первого игрока через позицию $(\mathtt{i},\mathtt{j})$, перебираются все позиции $(\mathtt{i'},\mathtt{j})$ и $(\mathtt{i},\mathtt{j'})$, возникшие после хода второго игрока, суммируются результаты первого игрока в них, а также величины Val, которые будут получены первым игроком, когда он сделает соответствующий ход. Величина $\mathrm{Val}\left[(\mathtt{i'},\mathtt{j}) \to (\mathtt{i},\mathtt{j})\right]$ равна сумме

значений на плитках в прямоугольнике $(\mathtt{i}+1)..\mathtt{i'} \times 1..\mathtt{j}$, а $\mathrm{Val}\left[(\mathtt{i},\mathtt{j'}) \to (\mathtt{i},\mathtt{j})\right]$ — сумме значений на плитках в прямоугольнике $1..\mathtt{i} \times (\mathtt{j}+1)..\mathtt{j'}$. При этом стоимость хода умножается на количество партий, в которых этот ход может быть сделан.

Вторая формула говорит, что количество партий, в которых после хода первого игрока может возникнуть позиция $(\mathtt{i},\mathtt{j})$, равна сумме партий, в который после хода второго игрока возникали позиции $(\mathtt{i'},\mathtt{j})$ и $(\mathtt{i},\mathtt{j'})$.

Ещё раз напомним, что сложение и умножение ведётся по модулю M.

Для учёта ситуаций, когда позиция $(\mathtt{i},\mathtt{j})$ возникла после хода второго игрока применим аналогичные формулы

$$\mathtt{a[i,j,2].value} = \sum_{\mathtt{i'=i+1}}^{n} \mathtt{a[i',j,1].value} + \sum_{\mathtt{j'=j+1}}^{m} \mathtt{a[i,j',1].value}, \qquad (3)$$

$$\mathtt{a[i,j,2].qnt} = \sum_{\mathtt{i'=i+1}}^{n} \mathtt{a[i',j,1].qnt} + \sum_{\mathtt{j'=j+1}}^{m} \mathtt{a[i,j',1].qnt}. \qquad (4)$$

Отличие в том, что стоимости ходов не идут в зачёт первому игроку: ход второго игрока, и дольки съедает он.

Из этих формул видно, что вычисление данных в ячейках можно вести в любом порядке, главное, чтобы к моменту просчёта ячейки $(\mathtt{i},\mathtt{j})$ были уже просчитаны все ячейки, расположенные сверху и справа от неё. В частности, можно вычислять строки в порядке убывания индекса $\mathtt{i}$, а внутри строки — в порядке убывания индекса $\mathtt{j}$.

В конечном итоге нас интересует величина

$$\mathtt{a[1,1,1].value} + \Big(\mathtt{a[1,1,2].value} + v[1, 1 \cdot \mathtt{a[1,1,2].qnt}\Big),$$

состоящая из суммы результатов первого игрока в тех партиях, где последняя долька досталась второму игроку (первое слагаемое), и в тех партиях, где последнюю дольку съел первый (слагаемое в скобках); последняя величина состоит из того, что первый игрок съел до последнего хода и стоимость последней дольки, умноженной на количество партий, в которых первый игрок может её съесть.

Описанная процедура имеет форму, стандартную для метода динамического программирования. Однако основная сложность состоит в том, чтобы придумать её содержание, даже зная, что решение должно опираться на динамическое программирование.

Следует также отметить, что аналогичная процедура может быть реализована и в рамках «ретроспекции»: не от полной плитки в начале к последней левой нижней дольке в конце, а наоборот — от дольки к полной плитке. Все идеи останутся в такой процедуре точно такими же.

Оценим трудоёмкость предложенной процедуры. Во-первых, в основном цикле вычисляются все nm ячеек основного массива. При вычислении информации в

каждой ячейке производится суммирование по всем ячейкам, лежащим справа и сверху от неё; их не более $(n+m)$ штук. Кроме того, для каждой такой ячейки производится вычисление стоимости хода, которое состоит в суммировании стоимостей долек, лежащих в прямоугольнике, имеющем размеры в худшем случае $n \times m$, то есть производится ещё nm действий. Таким образом, всего имеем $O(nm \cdot (n+m) \cdot nm) = O\big(n^2 m^2 (n+m)\big)$, то есть, в целом, пятая степень от размера шоколадки. Это очень много, в указанных ограничениях работа алгоритма такой сложности займёт слишком много времени. За разумное время такой алгоритм будет работать при n, $m \leqslant 40$.

Однако имеется, по крайней мере, возможность оптимизировать вычисление стоимостей хода — вычисление сумм стоимостей долек по всевозможным прямоугольникам. Идея этой оптимизации состоит в следующем. Пусть у нас имеется двумерный массив $\texttt{sumRect}[1..n,\ 1..m]$, в котором в ячейке $(\texttt{i}, \texttt{j})$ хранится сумма стоимостей долек в прямоугольнике $1..\texttt{i} \times 1..\texttt{j}$. Тогда сумма стоимостей долек в каком-либо прямоугольнике $i_1..i_2 \times j_1..j_2$, $i_1 < i_2$, $j_1 < j_2$, может быть найдена как

$$sum[i_1..i_2 \times j_1..j_2] = \texttt{sumRect}[i_2, j_2] - \texttt{sumRect}[i_1 - 1, j_2] - \\ - \texttt{sumRect}[i_2, j_1 - 1] + \texttt{sumRect}[i_1 - 1, j_1 - 1]. \quad (5)$$

Это есть формула сложения-вычитания, хорошо известная в математике. Если $j_1 - 1 = 0$ или $i_1 - 1 = 0$, то есть если $i_1 = 1$ или $j_1 = 1$, полагаем соответствующие слагаемые нулевыми.

Таким образом, при наличии массива $\texttt{sumRect}$ вместо квадратичной сложности при нахождении величин Val имеем сложность константную. Осталось понять, как построить этот массив. Но здесь вычисление идет с использованием этой же формулы:

$$\texttt{v}[i, j] = sum[i..i \times j..j] = \texttt{sumRect}[i, j] - \texttt{sumRect}[i - 1, j] - \\ - \texttt{sumRect}[i, j - 1] + \texttt{sumRect}[i - 1, j - 1],$$

откуда

$$\texttt{sumRect}[i, j] = \texttt{sumRect}[i - 1, j] + \texttt{sumRect}[i, j - 1] - \\ - \texttt{sumRect}[i - 1, j - 1] + \texttt{v}[i, j]. \quad (6)$$

Здесь массив $\texttt{v}$ содержит стоимости долек и вычисляется как

$$\texttt{v}[1, 1] = (p \cdot q) \bmod r, \qquad \texttt{v}[i, j] = \big(\texttt{v}[i - 1, j] \cdot p\big) \bmod r = \big(\texttt{v}[i, j - 1] \cdot q\big) \bmod r.$$

Видно, что отдельные процедуры вычисления массивов $\texttt{v}$ и $\texttt{sumRect}$ какждая имеют сложность $O(nm)$ и не ухудшают общую сложность алгоритма, которая теперь равна $O\big(nm(n+m)\big)$. Описанная процедура работает при n, $m \leqslant 1000$ и, как будет обсуждаться ниже, требует память около $16nm$ байт, что при указанных ограничениях на n и m составляет около 16 мегабайт.

Казалось бы, на этом оптимизация задачи закончена. Однако удивительным образом оказывается, что можно еще на один порядок улучшить сложность алгоритма, а именно оптимизировать суммирования по индексам `i'` и `j'` в пересчётных формулах (1)–(4). Оптимизация формул (2)–(4) идеологически уже несложна: здесь опять надо оптимизировать суммирование некоторых величин по прямоугольникам; пусть даже эти прямоугольники имеют высоту или ширину 1, идея быстрого суммирования та же, что и в процессе оптимизации нахождения суммы стоимостей долек в некотором прямоугольнике, отраженная в формуле (5). То есть, нам нужно дополнительно хранить суммы величин `a[i',j',1/2].value` и `.qnt` для всевозможных прямоугольников $\mathtt{i}..n \times \mathtt{j}..m$. Эти значения можно хранить, дополнив хранилища в ячейках массива `a` ещё двумя полями — для суммы полей `value` и для суммы полей `qnt`. После того, как вычислены величины `a[i,j,1/2].value` и `a[i,j,1/2].qnt`, эти поля находятся по формулам, аналогичным (6).

Остаётся вопрос о том, как оптимизировать суммирование в формуле (1); здесь идея не столь очевидна, так как имеется не просто суммирование величин `qnt`, а они умножаются на стоимости хода.

Для начала заметим, что

$$\mathrm{Val}\left[(i',j) \to (i,j)\right] = \mathrm{Val}\left[(i',j) \to (i+1,j)\right] + \mathrm{Val}\left[(i+1,j) \to (i,j)\right]. \qquad (7)$$

Сумма стоимостей долек на прямоугольнике $(i+1)..i' \times 1..j$ равна сумме стоимостей долек на прямоугольнике $(i+2)..i' \times 1..j$ плюс стоимость долек на полосочке $(i+1) \times 1..j$. Теперь рассмотрим формулу (1):

$$\mathtt{a}[i,j,1].\mathtt{value} = \sum_{i'=i+1}^{n} \Big(\mathtt{a}[i',j,2].\mathtt{value} + \mathtt{a}[i',j,2].\mathtt{qnt} \cdot \mathrm{Val}\left[(i',j) \to (i,j)\right]\Big) + \ldots$$

Второе слагаемое опущено, его оптимизация ведётся аналогичным образом. Сосредоточимся на оптимизации первого слагаемого. Общая цель — получить что-нибудь, связанное с величиной $\mathtt{a}[i+1,j,1].\mathtt{value}$, которая уже просчитана к моменту счёта $\mathtt{a}[i,j,1].\mathtt{value}$. Выделим в сумме слагаемое с $i' = i+1$:

$$\begin{aligned}
&\sum_{i'=i+1}^{n} \Big(\mathtt{a}[i',j,2].\mathtt{value} + \mathtt{a}[i',j,2].\mathtt{qnt} \cdot \mathrm{Val}\left[(i',j) \to (i,j)\right]\Big) = \\
&\qquad = \mathtt{a}[i+1,j,2].\mathtt{value} + \mathtt{a}[i+1,j,2].\mathtt{qnt} \cdot \mathrm{Val}\left[(i+1,j) \to (i,j)\right] + \\
&\qquad\qquad + \sum_{i'=i+2}^{n} \Big(\mathtt{a}[i',j,2].\mathtt{value} + \mathtt{a}[i',j,2].\mathtt{qnt} \cdot \mathrm{Val}\left[(i',j) \to (i,j)\right]\Big)
\end{aligned}$$

Рассмотрим слагаемое с суммой. Разделим величину Val на два слагаемых по формуле (7) и разделим одну сумму на две:

$$\sum_{i'=i+2}^{n} \Big(\mathtt{a}[i',j,2]\,.\,\mathtt{value} + \mathtt{a}[i',j,2]\,.\,\mathtt{qnt} \cdot$$

$$\cdot \big(\operatorname{Val}\big[(i',j) \to (i+1,j)\big] + \operatorname{Val}\big[(i+1,j) \to (i,j)\big]\big)\Big) =$$

$$= \sum_{i'=i+2}^{n} \Big(\mathtt{a}[i',j,2]\,.\,\mathtt{value} + \mathtt{a}[i',j,2]\,.\,\mathtt{qnt} \cdot \operatorname{Val}\big[(i',j) \to (i+1,j)\big]\Big) +$$

$$+ \sum_{i'=i+2}^{n} \Big(\mathtt{a}[i',j,2]\,.\,\mathtt{qnt} \cdot \operatorname{Val}\big[(i+1,j) \to (i,j)\big]\Big) =$$

$$= \sum_{i'=(i+1)+1}^{n} \Big(\mathtt{a}[i',j,2]\,.\,\mathtt{value} + \mathtt{a}[i',j,2]\,.\,\mathtt{qnt} \cdot \operatorname{Val}\big[(i',j) \to (i+1,j)\big]\Big) +$$

$$+ \operatorname{Val}\big[(i+1,j) \to (i,j)\big] \cdot \sum_{i'=i+2}^{m} \mathtt{a}[i',j,2]\,.\,\mathtt{qnt}\,.$$

Мы видим, что первое слагаемое в последнем полученном выражении весьма напоминает первое слагаемой в формуле (1), но только взятое для индекса $i+1$. Получить точно $\mathtt{a}[i+1,j,1]\,.\,\mathtt{value}$ не удалось, но получили половинку этой формулы. Обозначим эту величину через $\mathtt{a}[i+1,j,1]\,.\,\mathtt{vert}$, поле `vert` означает «часть, связанная с вертикальным суммированием». Подставляя полученное выражение назад, получаем

$$\sum_{i'=i+1}^{n} \Big(\mathtt{a}[i',j,2]\,.\,\mathtt{value} + \mathtt{a}[i',j,2]\,.\,\mathtt{qnt} \cdot \operatorname{Val}\big[(i',j) \to (i,j)\big]\Big) =$$

$$= \mathtt{a}[i+1,j,2]\,.\,\mathtt{value} + \mathtt{a}[i+1,j,2]\,.\,\mathtt{qnt} \cdot \operatorname{Val}\big[(i+1,j) \to (i,j)\big] +$$

$$+ \mathtt{a}[i+1,j,1]\,.\,\mathtt{vert} + \operatorname{Val}\big[(i+1,j) \to (i,j)\big] \cdot \sum_{i'=i+2}^{n} \mathtt{a}[i',j,2]\,.\,\mathtt{qnt} =$$

$$= \mathtt{a}[i+1,j,2]\,.\,\mathtt{value} + \mathtt{a}[i+1,j,1]\,.\,\mathtt{vert} +$$

$$+ \operatorname{Val}\big[(i+1,j) \to (i,j)\big] \cdot \sum_{i'=i+1}^{n} \mathtt{a}[i',j,2]\,.\,\mathtt{qnt}\,.$$

Проведя аналогичные выкладки для второго слагаемого в формуле (1) и введя поле `.horz` для его значения, получим соотношение

$$\sum_{j'=j+1}^{m} \Big(\mathtt{a}[i,j',2]\,.\,\mathtt{value} + \mathtt{a}[i,j',2]\,.\,\mathtt{qnt} \cdot \operatorname{Val}\big[(i,j') \to (i,j)\big]\Big) =$$

$$= \mathtt{a}[i,j+1,2]\,.\,\mathtt{value} + \mathtt{a}[i,j+1,1]\,.\,\mathtt{horz} +$$

$$+ \operatorname{Val}\big[(i,j+1) \to (i,j)\big] \cdot \sum_{j'=j+1}^{m} \mathtt{a}[i,j',2]\,.\,\mathtt{qnt}\,.$$

И в итоге получаем новое выражение для формулы (1):

$$\begin{aligned}
\mathtt{a}[i,j,1]\,.\,\mathtt{value} &= \mathtt{a}[i,j,1]\,.\,\mathtt{vert} + \mathtt{a}[i,j,1]\,.\,\mathtt{horz}\,,\\
\mathtt{a}[i,j,1]\,.\,\mathtt{vert} &= \mathtt{a}[i+1,j,2]\,.\,\mathtt{value} + \mathtt{a}[i+1,j,1]\,.\,\mathtt{vert} +\\
&\quad + \mathrm{Val}\left[(i+1,j)\to(i,j)\right]\cdot\sum_{i'=i+1}^{n}\mathtt{a}[i',j,2]\,.\,\mathtt{qnt}\,,\\
\mathtt{a}[i,j,1]\,.\,\mathtt{horz} &= \mathtt{a}[i,j+1,2]\,.\,\mathtt{value} + \mathtt{a}[i,j+1,1]\,.\,\mathtt{horz} +\\
&\quad + \mathrm{Val}\left[(i,j+1)\to(i,j)\right]\cdot\sum_{j'=j+1}^{m}\mathtt{a}[i,j',2]\,.\,\mathtt{qnt}\,.
\end{aligned}$$

В полученных формулах, если $i = n$ или $j = m$, то слагаемые и суммы с индексами $i+1$ и $j+1$ полагаются нулями. Таким образом, для вычисления $\mathtt{a}[i,j,1]\,.\,\mathtt{value}$ надо хранить дополнительно две величины — его «вертикальное» и «горизонтальное» слагаемое. Но такова цена этого последнего оптимизированного показателя степени в оценке сложности. И он того стоит!

Теперь формула (1) пересчёта величины $\mathtt{a}[i,j,1]\,.\,\mathtt{value}$ требует константного времени, а весь алгоритм имеет сложность $O(nm)$. Однако нам требуется достаточно много памяти для того, чтобы хранить все вспомогательные величины. Оценим суммарный потребный объем памяти:

- массив `v`: $n \cdot m$ ячеек 4-байтного целого типа; всего $4nm$ байт;
- массив `sumRect`: $n \cdot m$ ячеек 4-байтного целого типа; всего $4nm$ байт;
- основной массив `a`: $n \cdot m \cdot 2$ ячеек, каждая ячейка хранит 4 поля 4-байтного целого типа — `value`, `qnt`, `valueSum`, `qntSum` (последние две для быстрого суммирования полей `value` и `qnt`); всего $32nm$ байт.
- массив хранения «половинок» величин $\mathtt{a}[i,j,1]\,.\,\mathtt{value}$: $n \cdot m$ ячеек какждая с двумя полями 4-байтного целого типа; всего $8nm$ байт;

Тем самым имеем, что всего требуется $48nm$ байт (без учёта накладных расходов для организации структур данных в памяти), при максимальных ограничениях это составит 108 мегабайт. Процедура, не включающая оптимизацию вычисления $\mathtt{a}[i,j,1]\,.\,\mathtt{value}$, требует втрое меньше памяти — только $16nm$ байт.

Из изложенной выше процедуры решения следует, что от величин p, q, r ход работы процедуры не зависит вообще, поэтому во всех тестах положим $p = 3$, $q = 2$, $r = 1009$, как это было в примере в условии задачи.

<u>Идеи тестов:</u>

1. Минимальный тест, $n = m = 1$.
2. Минимальный тест, $n = 1$, $m = 7$.
3. Минимальный тест, $n = 8$, $m = 1$.

4–7. Тесты на полный перебор $n + m \leqslant 14$.

8–13. Тесты для неоптимальной процедуры $20 \leqslant n$, $m \leqslant 40$.

14–20. Тесты для кубической процедуры $70 \leqslant n$, $m \leqslant 1000$.

21–24. Тесты для квадратичной процедуры $1200 \leqslant n$, $m \leqslant 1400$.

25. Максимальный тест, $n = m = 1500$.