Муниципальный этап Всероссийской олимпиады школьников по информатике в 2017 – 2018 учебном году

Разборы решений и идеи тестов

Муниципальный этап Всероссийской олимпиады школьников по информатике в 2017 – 2018 учебном году

10 класс

Время выполнения задач — 4 часа

Ограничение по времени — 2 секунды на тест

Ограничение по памяти — 256 мегабайт

10.1. «Верным путём идём...». Петя Торопыжкин может пойти в школу двумя разными путями. Первый он проходит за t_1 минут, а второй — за t_2 минут. На обдумывание одной интересной идеи он тратит d минут. Во время движения Петя думает постоянно, начиная обдумывать первую идею сразу по выходу из дома и по окончанию обдумывания предыдущей идеи сразу начинает обдумывать следующую. Когда интересная идея обдумана целиком, он получает f единиц удовольствия, а если к моменту окончания пути обдумывание какой-то идеи началось и не завершилось, он теряет l единиц удовольствия. Каким маршрутом нужно ему пойти, чтобы получить наибольшее удовольствие? Укажите номер маршрута и получаемое количество единиц удовольствия за одно прохождение по нему.

Формат входа: В единственной строке через пробел перечислены пять целых чисел: t_1, t_2, d, f, l — время прохождения по первому и второму маршруту, время обдумывания одной идеи, единицы удовольствия, получаемые за полностью обдуманную идею и теряемые за идею, не обдуманную до конца ($1 \leqslant t_1, t_2 \leqslant 10^4$; $1 \leqslant d \leqslant 100$; $0 \leqslant f, l \leqslant 1000$).

Формат выхода: Выведите через пробел два целых числа: номер маршрута (1 или 2), при движении по которому Петя получит наибольшее удовольствие, и количество единиц этого удовольствия. Если оба маршрута дают одинаковое удовольствие, укажите первый.

Пример 1		**Пример 2**	
Вход:	*Выход:*	*Вход:*	*Выход:*
`15 10 2 3 8 2 15`		`15 10 2 3 6 1 15`	

10.2. «Поиск ключа». Петя Торопыжкин постоянно забывал свои пароли от разных сайтов. Для более лёгкого запоминания он всегда выбирал пароли в виде неубывающих последовательностей цифр, больших и малых символов латиницы. При этом порядок на символах соответствовал порядку их кодов в таблице ASCII: цифры меньше больших букв, которые меньше маленьких букв. Буквы внутри наборов возрастают в алфавитном порядке. Чтобы спрятать эти пароли, он написал программу, которая формировала строку длиной до 10^5 символов, в которой введённый Петей пароль был единственной максимальной по длине неубывающей подстрокой. Теперь надо помочь ему написать процедуру, которая из такой строки выделяла бы пароль.

Формат входа: В единственной строке имеется непустая последовательность цифр, больших и малых символов латиницы, такая, что среди всех неубывающих (в смысле указанного порядка символов) подстрок существует единственная максимальной длины. Количество символов в последовательности не превосходит 10^5.

Формат выхода: Выведите максимальную по длине неубывающую подстроку.

Пример

Вход:	*Выход:*
`A01ABab0123`	`01ABab`

10.3. «Словарь братьев по разуму». В фантастическом романе, который пишет Петя Торопыжкин, инопланетные существа используют алфавит, состоящий из двух символов ⩒ и ⩔, которые в рабочем варианте текста Петя представляет заглавными буквами `F` и `G` (для простоты). Петя даже составил словарь языка этих существ. Для быстроты поиска по словарю он выбрал целое число p и сопоставил каждому слову $w = \alpha_0\alpha_1 \ldots \alpha_k$ целое число $h(w) = \sum_{i=0}^{k} a_i \cdot p^i$, где коэффициент a_i равен 0, если $\alpha_i =$ `F`, и 1, если $\alpha_i =$ `G`. Однако такое число может быть большим, поэтому Петя запоминает остаток от деления $h(w)$ на некоторое другое целое число D.

Такое число называется *хешем* слова w, а правило вычисления хеша — *хеш-функцией*. Вычислив один раз хеши слов из словаря, дальше очень просто проверять их на несовпадение: если хеши двух слов различаются, то и сами слова совпадать не могут. А вот если хеши двух слов совпадают (такую ситуацию называют *коллизией*), тогда для точной проверки эти слова надо сравнивать посимвольно.

Для быстрой работы со словарём надо написать программу, которая ищет в нём слова, чей хеш совпадает с заданным значением.

Формат входа: В первой строке через пробел заданы два целых числа p и D, определяющие хеш-функцию ($1 \leqslant p \leqslant 10^9$, $1 \leqslant D \leqslant 2 \cdot 10^9$). Во второй строке задано целое число H — требуемое значение хеша ($0 \leqslant H < D$). В третьей строке задано целое число n — количество слов в словаре ($1 \leqslant n \leqslant 10^3$). В следующих n строках заданы слова — непустые последовательности заглавных символов латиницы `F` и `G` длиной не более 10^3 символов.

Формат выхода: Если слова с указанным хешем найдены, выведите в первой строке «`FOUND`», а затем найденные слова по одному в строке (в каком-либо порядке). Если таких слов не найдено, выведите в первой строке «`NOT FOUND`», а во второй через пробел наименьшее и наибольшее значения хешей слов словаря.

Пример 1

Вход:	*Выход:*
7 100 1 3 FFFFG FFG G	FOUND G FFFFG

Пример 2

Вход:	*Выход:*
7 100 10 3 FFFFG FFG G	NOT FOUND 1 49

10.4. «Перебираем числа». Простая прямолинейная формулировка: есть набор из n целых положительных чисел. Нужно найти наименьшее общее кратное наибольших общих делителей всевозможных троек чисел из этого набора (в тройке числа берутся из разных позиций в наборе). Так как эта величина может быть очень большой, выдайте остаток от её деления на $10^9 + 7$.

Формат входа: В первой строке задано целое число n — количество чисел в наборе ($3 \leqslant n \leqslant 13000$). Во второй строке через пробел перечислены числа a_i из набора ($1 \leqslant a_i \leqslant 3 \cdot 10^6$).

Формат выхода: Выведите единственное целое число — остаток от деления искомого НОК на $10^9 + 7$.

Пример

Вход:	*Выход:*
4 12 1100 44 242	44

Примечание: Для этих чисел: НОД$(12, 1100, 44) = 4$, НОД$(12, 1100, 242) = 2$, НОД$(12, 44, 242) = 2$, НОД$(1100, 44, 242) = 22$ и НОК$(4, 2, 2, 22) = 44$.

10.5. «Ещё о паролях». Петя Торопыжкин нашёл свой старый архив, но не может вспомнить пароль, чтобы распаковать его. Всё, что он помнит: пароль непуст, состоит только из заглавных символов латиницы, имеет длину не более l символов, не содержит двух одинаковых символов подряд и в смысле лексикографического порядка расположен между двумя известными строками s_1 и s_2, строго больше s_1 и строго меньше s_2. Ему нужно посчитать, сколько таких строк существует (чтобы понять, а стоит ли вообще пытаться перебирать их). Посчитайте и вы. Но поскольку это число может быть очень большим, выдайте остаток от деления его на $10^9 + 7$.

Формат входа: В первой строке задано натуральное число l — максимальная длина пароля ($1 \leqslant l \leqslant 10^5$). Во второй и третьей строках заданы строки s_1 и s_2 — непустые строки из заглавных символов латиницы длины не более l символов, $s_1 < s_2$.

Формат выхода: Выдайте остаток от деления на 10^9+7 количества строк указанного вида.

Пример

Вход:	*Выход:*
3 A BB	677

Примечание: Эти 677 строк состоят из строк вида: `A?` — 25 штук (на месте `?` может стоять любой из 25 символов, отличных от `A`), `A??` — $25^2 = 625$ (вместо первого `?` стоит любой из 25 символов, отличных от `A`, вместо второго — любой из 25 символов, отличных от символа, стоящего вместо первого `?`), `B` — 1, `BA` — 1, `BA?` — 25 строк: $25 + 625 + 1 + 1 + 25 = 677$.

Муниципальный этап Всероссийской олимпиады школьников по информатике в 2017 – 2018 учебном году
10 класс. Разбор решений и идеи тестов

10.1. «Верным путём идём...». *Петя Торопыжкин может пойти в школу двумя разными путями. Первый он проходит за t_1 минут, а второй — за t_2 минут. На обдумывание одной интересной идеи он тратит d минут. Во время движения Петя думает постоянно, начиная обдумывать первую идею сразу по выходу из дома и по окончанию обдумывания предыдущей идеи сразу начинает обдумывать следующую. Когда интересная идея обдумана целиком, он получает f единиц удовольствия, а если к моменту окончания пути обдумывание какой-то идеи началось и не завершилось, он теряет l единиц удовольствия. Каким маршрутом нужно ему пойти, чтобы получить наибольшее удовольствие? Укажите номер маршрута и получаемое количество единиц удовольствия за одно прохождение по нему.*

Задача имеет уровень утешительной. Решение требует небольших математических размышлений, которые приводят к следующему алгоритму. Сначала для каждого пути вычисляем
1) количество идей, которые будут обдуманы целиком: t_i `div` d;
2) количество удовольствия, которое будет получено за завершение обдумывания этих идей: $(t_i$ `div` $d) \cdot f$;
3) если t_i `mod` $d \neq 0$, то есть на i-м пути имеется идея, не обдуманная до конца, то из полученной величины надо вычесть l.

Затем сравниваем итоговые числа и выбираем наибольшее, аккуратно обрабатывая ситуацию их совпадения.

Идеи тестов:

1–7. Случайные тесты, первый путь лучше.
8–14. Случайные тесты, второй путь лучше.
15–20. Случайные тесты, пути одинаково хороши.

10.2. «Поиск ключа». *Петя Торопыжкин постоянно забывал свои пароли от разных сайтов. Для более лёгкого запоминания он всегда выбирал пароли в виде неубывающих последовательностей цифр, больших и малых символов латиницы. При этом порядок на символах соответствовал порядку их кодов в таблице ASCII: цифры меньше больших букв, которые меньше маленьких букв. Буквы внутри наборов возрастают в алфавитном порядке. Чтобы спрятать эти пароли, он написал программу, которая формировала строку длиной до 10^5 символов, в которой введённый Петей пароль был единственной максимальной по длине неубывающей подстрокой. Теперь надо помочь ему написать процедуру, которая из такой строки выделяла бы пароль.*

Задача представляется, в целом, достаточно простой, хотя имеет несколько тонких моментов в своём решении.

Идея весьма прямолинейна: выделяем из данной строки неубывающие подстроки и ищем из них максимальную в лексикографическом порядке. Алгоритм выделения неубывающей подстроки, в целом, тоже несложен: в цикле перебираем индекс i от 2 до длины строки `str`; если $\texttt{str}[i-1] > \texttt{str}[i]$, то на символе с индексом $i-1$ закончилась очередная неубывающая часть. Тонкость заключается в том, что нужно не забыть обработать неубывающую подстроку, кончающуюся вместе с обрабатываемой строкой.

Соответственно, алгоритм сравнения строк должен на вход принимать индексы начал сравниваемых подстрок и их длины. То есть, если есть желание сэкономить на копировании подстрок, то процедуру сравнения надо реализовать самому.

Заметим, что сложность разумного алгоритма — $O(n)$: каждый символ один раз проверяется на неубывание при формирование очередной неубывающей строки и один раз сравнивается при сравнении текущей накопленной неубывающей подстроки с текущим максимумом.

Идеи тестов:

1. Односимвольная строка.
2. Двухсимвольная строка, символы совпадают.
3. Двухсимвольная строка, символы убывают.
4. Двухсимвольная строка, символы возрастают.
5. Трёхсимвольная строка $a_1a_2a_3$: $a_1 > a_2 > a_3$.
6. Трёхсимвольная строка $a_1a_2a_3$: $a_1 < a_2 > a_3$, максимум a_1a_2.
7. Трёхсимвольная строка $a_1a_2a_3$: $a_1 < a_2 > a_3$, максимум a_3.
8. Трёхсимвольная строка $a_1a_2a_3$: $a_1 > a_2 < a_3$, максимум $a2a3$.
9. Трёхсимвольная строка $a_1a_2a_3$: $a_1 > a_2 > a_3$, максимум a_1.
10. Трёхсимвольная строка $a_1a_2a_3$: $a_1 = a_2 = a_3$.
11. Алфавит в обратном порядке
12. Много букв, максимум в начале строки, это самая длинная из неубывающих подстрок.
13. Много букв, максимум в начале строки, это не самая длинная из неубывающих подстрок.
14. Много букв, максимум в середине строки, это самая длинная из неубывающих подстрок.
15. Много букв, максимум в середине строки, это не самая длинная из неубывающих подстрок.
16. Много букв, максимум в конце строки, это самая длинная из неубывающих подстрок.
17. Много букв, максимум в конце строки, это не самая длинная из неубывающих подстрок.
18. Много букв, вся строка неубывает.

19–25. То же, что в тестах 12–18, длина строки 10^5.

10.3. «Словарь братьев по разуму». *В фантастическом романе, который пишет Петя Торопыжкин, инопланетные существа используют алфавит, состоящий из двух символов* ⧜ *и* ⧝*, которые в рабочем варианте текста Петя представляет заглавными буквами* `F` *и* `G` *(для простоты). Петя даже составил словарь языка этих существ. Для быстроты поиска по словарю он выбрал целое число p и сопоставил каждому слову $w = \alpha_0\alpha_1\ldots\alpha_k$ целое число $h(w) = \sum_{i=0}^{k} a_i \cdot p^i$, где коэффициент a_i равен 0, если $\alpha_i =$* `F`*, и 1, если $\alpha_i =$* `G`*. Однако такое число может быть большим, поэтому Петя запоминает остаток от деления $h(w)$ на некоторое другое целое число D.*

Такое число называется хешем слова w, а правило вычисления хеша — хеш-функцией. Вычислив один раз хеши слов из словаря, дальше очень просто проверять их на несовпадение: если хеши двух слов различаются, то и сами слова совпадать не могут. А вот если хеши двух слов совпадают (такую ситуацию называют коллизией), тогда для точной проверки эти слова надо сравнивать посимвольно.

Для быстрой работы со словарём надо написать программу, которая ищет в нём слова, чей хеш совпадает с заданным значением.

Задача представляется программному комитету весьма технической и, как следствие, имеющей сложность чуть ниже среднего.

Собственно, первая техническая тонкость заключается в том, что нужно постоянно считать степени числа p. Если каждый раз честно производить умножение, то общая сложность такого алгоритма будет $O(nl^2)$, где l — длина строк: для каждого слова, которых n штук, l раз надо возвести число p в степень до l. Однако это вычисление можно упростить, даже несколькими способами.

1) Можно предпросчитать один раз все степени вплоть до максимально возможной, а затем просто извлекать эти данные из массива.

2) Можно при обработке каждого слова хранить предыдущую степень p и на её основе вычислять следующую.

3) Можно применить алгоритм *быстрого возведения в степень*:

$$a^n = \begin{cases} n = 2k, & (a^2)^k, \\ n = 2k+1, & (a^2)^k \cdot a. \end{cases}$$

4) Можно применить схему Горнера вычисления значения многочлена в точке:

$$\begin{aligned} P(x) &= \alpha_m x^m + \alpha_{m-1}x^{m-1} + \alpha_{m-2}x^{m-2} + \ldots + \alpha_2 x^2 + \alpha_1 x + \alpha_0 = \\ &= (\alpha_m x^{m-1} + \alpha_{m-1}x^{m-2} + \alpha_{m-2}x^{m-3} + \ldots + \alpha_2 x + \alpha_1)x + \alpha_0 = \\ &= \big((\alpha_m x^{m-2} + \alpha_{m-1}x^{m-3} + \alpha_{m-2}x^{m-4} + \ldots + \alpha_2)x + \alpha_1\big)x + \alpha_0 = \\ &= \ldots = \Big(\ldots\Big(\big((0\cdot x + \alpha_m)x + \alpha_{m-1}\big)x + \alpha_{m-2}\Big)x + \ldots\Big)x + \alpha_0. \end{aligned}$$

То есть, чтобы вычислить значение многочлена в точке, можно, начав с нулевого значения аккумулятора, перебирать коэффициенты от старших к младшим (в нашем случае — с конца очередной строки к началу) и проводить итеративную процедуру: предыдущее значение аккумулятора умножается на значение x (в нашем случае p) и к результату прибавляется очередной коэффициент.

Вторая техническая тонкость заключается в том, что все вычисления надо проводить по модулю D: $u + v \to (u + v) \bmod D$, $u \cdot v \to (u \cdot v) \bmod D$. А поскольку значение D достаточно большое, то вычисления следует проводить в 8-байтном целом типе `__int64` (или его аналогах в других языках).

Наконец, третья особенность заключается в том, что разумно не делать два прохода по словам, проверяя, есть ли слова с требуемым хешем, а сразу начать считать минимальное и максимальное значение хеша, а в случае нахождения строки с требуемым значением переключаться на отбор слов с заданным хешем.

Идеи тестов:

1. Одно короткое слово (при вычислении хеша нет выхода за 4-байтный целый тип), хеш у него требуемый.
2. Одно короткое слово, хеш не совпадает с требуемым.
3. Одно длинное слово (при вычислении хеша есть выход за 4-байтный целый тип), хеш у него требуемый.
4. Одно длинное слово, хеш не совпадает с требуемым.
5. 100 коротких слов, есть одно слово с требуемым хешем, это слово первое в списке.
6. 100 коротких слов, есть одно слово с требуемым хешем, это слово не первое в списке.
7. 100 коротких слов, есть много слов с требуемым хешем, первое из таких слов первое в списке.
8. 100 коротких слов, есть много слов с требуемым хешем, первое из таких слов не первое в списке.
9. 100 коротких слов, нет слов с требуемым хешем.

10–14. То же, что в тестах 5–9, только слова длинные.

15–24. То же, что в тестах 5–14, только слов 1000.

25–29. То же, что в тестах 10–14, только слова полной длины 1000.

30–34. То же, что в тестах 24–29, только слов 1000.

35–42. Случайные тесты с короткими словами.

43–50. Случайные тесты с длинными словами.

10.4. «Перебираем числа». *Простая прямолинейная формулировка: есть набор из n целых положительных чисел. Нужно найти наименьшее общее кратное наибольших общих делителей всевозможных троек чисел из этого набора (в тройке числа*

берутся из разных позиций в наборе). Так как эта величина может быть очень большой, выдайте остаток от её деления на $10^9 + 7$.

Задача является достаточно сложной, поскольку лобовое решение, связанное с нахождением всех НОДов, а после — их НОКа, работает только для небольших количеств чисел в наборе — до 500-600. Однако, реализуется оно достаточно просто: $\text{НОД}(a, b, c) = \text{НОД}\big(\text{НОД}(a, b), c\big)$, $\text{НОК}(a_1, a_2, \ldots, a_n) = d \cdot (a_1/d) \cdot (a_2/d) \cdot \ldots \cdot (a_n/d)$, где $d = \text{НОД}(a_1, a_2, \ldots, a_n)$; НОДы находятся алгоритмом Евклида. При этом умножения при вычислении НОКа требуется делать по модулю $10^9 + 7$, что подразумевает использование 64-битного целого типа.

Для получения оптимального по скорости решения следует воспользоваться идеей разложения целого числа на простые множители. Если

$$a = p_1^{\alpha_1} \cdot p_2^{\alpha_2} \cdot \ldots \cdot p_m^{\alpha_m} \quad \text{и} \quad b = p_1^{\beta_1} \cdot p_2^{\beta_2} \cdot \ldots \cdot p_m^{\beta_m},$$

то

$$\text{НОД}(a, b) = p_1^{\min(\alpha_1, \beta_1)} \cdot p_2^{\min(\alpha_2, \beta_2)} \cdot \ldots \cdot p_m^{\min(\alpha_m, \beta_m)}$$

и

$$\text{НОК}(a, b) = p_1^{\max(\alpha_1, \beta_1)} \cdot p_2^{\max(\alpha_2, \beta_2)} \cdot \ldots \cdot p_m^{\max(\alpha_m, \beta_m)}.$$

Формулы не изменяются для вычисления НОД и НОК большего количества чисел.

Из этих формул видно, что действия ведутся независимо с каждым из простых делителей. Поэтому дальше будем рассматривать работу с одним делителем. Пусть задан набор чисел $\{a_i\}$, и простой делитель p у числа a_i имеет кратность α_i. Тогда у наибольших общих делителей $\text{НОД}(a_i, a_j, a_k)$ троек чисел делитель p будет иметь степень $\min(\alpha_i, \alpha_j, \alpha_k)$. А у НОДа всех НОКов делитель p будет иметь степень $\max\limits_{i,j,k} \min(\alpha_i, \alpha_j, \alpha_k)$. Так как рассматриваются все тройки чисел, то в частности будут рассмотрена такая тройка, у чисел которой показатели степени p — три наибольших величины среди всех чисел a_i. Отсюда понятно, что $\max\limits_{i,j,k}$ будет равен третьей величине в упорядоченном по убыванию ряду степеней делителя p среди всех чисел a_i. Впрочем, для нахождения этой величины не требуется сортировка, достаточно модифицировать алгоритм поиска максимума.

Теперь встаёт вопрос разложения чисел на простые множители и хранения информации о степенях тех или иных множителей в этих разложениях. Первым полезным классическим наблюдением здесь является то, что если $a = b \cdot c$ и $b \leqslant c$, то $b \leqslant \sqrt{a} \leqslant c$. Соответственно, для поиска младших делителей нужно перебирать простые числа от 2 до $\sqrt{3 \cdot 10^6} \approx 1732$. Таких чисел 272 штуки, их можно предпросчитать и просто вписать в текст программы. Тогда для каждого числа a_i эти простые делители перебираются, для каждого находится его степень в числе a_i; если после выделения всех делителей, не превосходящих 1742, остаётся число отличное от 1, то это гарантированно простое число, превосходящее 1742, и его тоже нужно учесть.

Для хранения троек наибольших степеней можно прямолинейно использовать массив байтов размером $3\,000\,000 \times 3$ (байтов, поскольку степень простых делителей не

может быть больше 22: $2^{22} = 4194304 > 3000000$ — и для её хранения достаточно байта). При этом, конечно, большинство ячеек, соответствующих составным числам использованы не будут. Также для хранения этой информации можно использовать словарь, в котором ключом является целое число, простой делитель, а значением — тройка его наибольших степеней.

После того, как все числа обработаны и вычислены тройки наибольших степеней каждого из возможных простых делителей, вычисляем НОК: идём по всем ячейкам массива (или элементам словаря) и для каждого числа, у которого третья из наибольших степеней отлична от нуля, умножаем (по модулю $10^9 + 7$) результат на соответствующий простой множитель, возведённый в эту степень. Здесь для некоторого ускорения работы можно применить алгоритм быстрого возведения в степень.

В итоге имеем алгоритм со сложностью $O(n)$, правда, операция с каждым числом (разложение на простые множители) весьма времяёмкая.

Идеи тестов:

1. Три совпадающих небольших простых числа.
2. Три совпадающих больших простых числа.
3. 100 совпадающих небольших простых числа.
4. 100 совпадающих больших простых числа.
5. 13000 совпадающих небольших простых числа.
6. 13000 совпадающих больших простых числа.

7–12. Те же идеи, что и в тестах 1–6, только числа *полупростые*, являющиеся произведением двух простых.

13–18. Те же идеи, что и в тестах 1–6, только имеются три полупростых числа, полученных разной комбинацией из трёх простых делителей.

19–26. Случайные тесты: 100 чисел до 2000.

27–34. Случайные тесты: 100 чисел до до $3 \cdot 10^6$.

35–42. Случайные тесты: до 13000 чисел до 2000.

43–50. Случайные тесты: до 13000 чисел до до $3 \cdot 10^6$.

10.5. «Ещё о паролях». *Петя Торопыжкин нашёл свой старый архив, но не может вспомнить пароль, чтобы распаковать его. Всё, что он помнит: пароль непуст, состоит только из заглавных символов латиницы, имеет длину не более l символов, не содержит двух одинаковых символов подряд и в смысле лексикографического порядка расположен между двумя известными строками s_1 и s_2, строго больше s_1 и строго меньше s_2. Ему нужно посчитать, сколько таких строк существует (чтобы понять, а стоит ли вообще пытаться перебирать их). Посчитайте и вы. Но поскольку это число может быть очень большим, выдайте остаток от деления его на $10^9 + 7$.*

Данная задача задумывалась программным комитетом как сложная. Идея решения — лобовой подсчёт количества требуемых строк — возникает здесь достаточно несложно, однако детальная разработка алгоритма является нетривиальной.

Обозначим символы строки s_1 через α_i, а s_2 — β_j: $s_1 = \alpha_1\alpha_2\ldots\alpha_n$, $s_2 = \beta_1\beta_2\ldots\beta_m$; n — длина s_1, m — длина s_2.

Можно разделить две различных ситуаций:

1) s_1 и s_2 имеют общее начало;

2) s_1 и s_2 не имеют общего начала.

Разделение этих случаев и нахождение длины общего начала s_1 и s_2 (если оно есть) производится в процессе их лексикографического сравнения.

Первый случай сводится ко второму отбрасыванием общего начала s_1 и s_2 и соответствующим уменьшением максимальной длины l рассматриваемых строк.

Ответ во втором случае складывается из трёх компонент:

а) строки, имеющие общее начало с s_1 и бóльшие, чем s_1;

б) строки, начинающиеся с символов, бóльших, чем первый символ s_1, но меньших, чем первый символ s_2 (таких строк может вообще не быть, если $\beta_1 = \alpha_1 + 1$);

в) строки, имеющие общее начало с s_2 и меньшие, чем s_2.

Ответ в пункте б) есть

$$A \cdot (1 + 25 + 25^2 + \ldots + 25^{l-1}) = A \cdot S(l-1),$$

где A — количество возможных первых символов, то есть символов, больших α_1, но меньших β_1, а

$$S(k) = \sum_{i=0}^{k} 25^i, \qquad 0 \leqslant k \leqslant l.$$

Величины $S(k)$ будут нужны и для дальнейшей части решения, поэтому разумно их предпросчитать перед началом алгоритма.

Появление числа 25 поясняется в примечании к примеру в условии задачи. Величина $S(k)$ есть сумма геометрической прогрессии, однако нельзя воспользоваться формулой, получив выражение $(25^k - 1)/(25 - 1)$, поскольку все вычисления производятся по модулю числа $10^9 + 7$ и осуществление операции деления в этом случае затруднено.

Ответ в пункте а) складывается из строк, которые имеют всю строку s_1 своим началом (они автоматически больше, чем s_1), и строк, имеющих с s_1 общее начало длины, меньшей длины s_1. Строк первого вида с длиной $n + 1$ существует 25 штук, с длиной $n + 2$ — 25^2 штук и т.д., то есть всего $25 + 25^2 + \ldots + 25^{l-n} = S(l-n) - 1$ (при $n = l$ как раз получается ноль).

Теперь посчитаем количество строк, имеющих с s_1 общее начало с длиной, меньшей длины s_1. Каждое значение индекса i от 2 до n определяет первый символ, который будет различаться у s_1 и конструируемой строки. Проводя рассуждение, аналогичное пункту б), получаем, что число таких строк равно

$$B_i \cdot (1 + 25 + 25^2 + \ldots + 25^{l-i}) = B_i \cdot S(l-i),$$

где B_i — количество символов, больших α_i.

Итак, ответ к пункту а) есть

$$\big(S(l-n)-1\big)+\sum_{i=2}^{n} B_i \cdot S(l-i).$$

Ответ к пункту в) ищется, исходя из похожих рассуждений. Строка, начинающаяся с символа β_1 и меньшая s_2, должна для какого-то i, $2 \leqslant i \leqslant m$, иметь i-й символ, меньший соответствующего символа β_i строки s_2 и произвольный хвост без повторяющихся символов. Ответ получается

$$\sum_{i=2}^{m} C_i \cdot S(l-i),$$

где C_i — количество символов, меньших β_i.

Итак, в третьем случае общий ответ вычисляется как

$$\big(S(l-n)-1\big)+\sum_{i=2}^{n} B_i \cdot S(l-i) \quad + \quad A \cdot S(l-1) \quad + \quad \sum_{i=2}^{m} C_i \cdot S(l-i),$$

где $A = \texttt{ord}(\beta_1) - \texttt{ord}(\alpha_1) - 1$, $B_i = \texttt{ord('Z')} - \texttt{ord}(\alpha_i)$, $C_i = \texttt{ord}(\beta_i) - \texttt{ord('A')}$; $\texttt{ord}(\cdot)$ — функция взятия кода символа (для Паскаля) или сам символ (для C/C++/Java/C#). Из этой формулы следует, что сложность алгоритма есть $O(n+m)$.

Все вычисления в рамках этой формулы и при нахождении $S(k)$, конечно, ведутся по модулю 10^9+7.

Идеи тестов:

1. Короткие строки, общего начала нет, $s_2 > s_1$, $\beta_1 = \alpha_1 + 1$.
2. Короткие строки, общего начала нет, $s_2 > s_1$, $\beta_1 > \alpha_1 + 1$.
3. Короткие строки, есть общее начало, $s_2 > s_1$, $\beta_i = \alpha_i + 1$.
4. Короткие строки, есть общее начало, $s_2 > s_1$, $\beta_i > \alpha_i + 1$.
5. Короткие строки, s_1 — начало s_2, $s_2 > s_1$.

6–10. То же, что в тестах 1–5, строки длиной 1000.

11–15. То же, что в тестах 1–5, строки длинные.

16–25. Случайные тесты.