

Задача А. Два знака

В задаче нужно было расставить знаки арифметических операций в две позиции так, чтобы в результате вычисления выражения получилось максимальное число. Наиболее простым способом было вычислить все возможные шесть вариантов, и выбрать из них максимальное число. Предварительно нужно было разбить число на разряды: можно это было сделать с помощью арифметики (последняя цифра — это остаток от деления на 10, предпоследняя — это остаток от деления на 10 целой части от деления на 10, а первая — это целая часть от деления на 100), а можно с помощью представления числа в виде строки: считать строку и обратиться к нулевому, первому и второму элементам, преобразовав их в числа.

Правильное решение на языке Python:

```
s = int(input())
a = s // 100
b = s // 10 % 10
c = s % 10
ans = max(a * b + c, a * b - c, a + b * c, a + b - c, a - b * c, a - b + c)
print(ans)
```

Задача В. Аугментация

Для решения задачи её необходимо внимательно прочитать, в условии упоминаются четыре языка программирования: C, C++, Python и Pascal.

Задача С. Экскурсия

При небольших n можно перебрать все возможные варианты выбрать несколько делегаций из списка, и из всех вариантов, в которых суммарное количество участников экскурсии $\leq k$, выбрать вариант с наибольшим количеством делегаций. Однако, таких вариантов 2^n , и при больших n перебрать их за отведённое время невозможно.

Чтобы уложиться в ограничение по времени, оптимизируем перебор. Отсортируем делегации по возрастанию их размеров. Пусть a_i — i -я делегация в порядке сортировки. Тогда нужно будет перебрать только префиксы массива a : $(\emptyset)$, (a_1) , (a_1, a_2) , (a_1, a_2, a_3) , $\dots$, $(a_1, a_2, a_3, \dots, a_n)$. Их гораздо меньше — всего $n + 1$, и их можно перебрать за отведённое время. Докажем, что нельзя выбрать несколько делегаций не с префикса a и получить ответ лучше. Пусть мы выбрали x делегаций $n_1 < n_2 < \dots < n_x$, и суммарное количество человек в них $a_{n_1} + a_{n_2} + \dots + a_{n_x}$ оказалось $\leq k$, но при этом суммарное количество человек в делегациях с префикса a длины x ($a_1 + a_2 + \dots + a_x$) оказалось $> k$. Но тогда $n_1 \geq 1$, $n_2 \geq 2$, $\dots$, $n_x \geq x$, и так как a отсортирован, то $a_{n_1} \geq a_1$, $a_{n_2} \geq a_2$, $\dots$, $a_{n_x} \geq a_x$, и сумма $a_1 + a_2 + \dots + a_x \leq a_{n_1} + a_{n_2} + \dots + a_{n_x}$. Но по предположению первая сумма $> k$, а вторая $\leq k$. Получили противоречие, значит, выбрать такие $n_1, n_2, \dots, n_x$ нельзя.

Задача D. Ипотека длиной в жизнь

Так как округление в задаче до копеек, переведём рубли в копейки и будем работать с ними (далее считаем, что p и s представлены в копейках).

Пусть d_i — долг к концу i -го года, тогда в $i + 1$ -м году сначала начислятся проценты, и долг станет равным $\lceil d_i \cdot (1 + \frac{r}{100}) \rceil$, а затем произойдёт погашение, то есть долг уменьшится на p . Таким образом, $d_{i+1} = \max(0, \lceil d_i \cdot (1 + \frac{r}{100}) \rceil - p)$.

В задаче необходимо проверить, что $d_n = 0$, что можно сделать с помощью цикла. Также нужно аккуратно реализовать решение, так как значения d_i могут достигать 10^{200} , из-за чего целочисленные типы данных в некоторых языках могут переполниться.

Задача Е. Подарок для Нанналли

Пусть день рождения сестры приходится на m_s -й месяц и d_s -й день, день рождения брата на y_b -й год, m_b -й месяц и d_b -й день, а побег С.С. на y_c -й год, m_c -й месяц и d_c -й день.

Рассмотрим все даты (тройки чисел (y, m, d)), в которые можно было начать осуществлять план. Чтобы приготовления к захвату были окончены вовремя, необходимо, чтобы $\min((y_b + n, m_b, d_b); (y_c, m_c, d_c)) \leq (y, m, d)$ (1).

А чтобы Лелуш успел завершить план — $(y + 1, m, d) \leq (y_b + k, m_b, d_b)$ (2). Сравнивать даты следует сначала по году, затем, при их равенстве, по номеру месяца, а если равны и они, то по номеру дня.

Также $m = m_s$ и $d = d_s$, так как необходимо, чтобы захват Британии начался ровно за год до дня рождения сестры, а значит, день и месяц даты начала захвата будут такими же, как и день и месяц рождения сестры, а значит, задача свелась к перебору y и проверке условий (1) и (2), или к формуле.

Задача F. Весёлые замены

Определим для строк s и t , можно ли получить t из s при помощи весёлых замен. Если длина строк различается, то ответ — нет, так как при помощи этих замен нельзя поменять длину строки. Если длины строк совпадают, то попытаемся приравнять i -й символ одной строки к i -му символу другой:

1. Если у одной строки i -й символ — согласная, то у другой i -м символом должна быть такая же согласная, так как при помощи весёлых замен нельзя ни заменить её, так как весёлую замену нельзя применить к согласной, ни передвинуть на её место другую букву, так как весёлые замены не меняют порядок букв.
2. Если же у одной строки i -й символ — гласная, то у другой i -м символом тоже должна быть гласная, но не обязательно такая же, а любая, так как её можно заменить на нужную при помощи весёлой замены.

Если для всех i эти условия выполнены, то при помощи весёлых замен можно приравнять все символы одной строки к соответствующим символам другой и тем самым сделать строки равными, иначе — нельзя. Теперь осталось перебрать все строки t_i из набора и сравнить их при помощи этой проверки с s . Чтобы упростить проверку, можно заменить все гласные во всех строках на какой-нибудь одинаковый символ, например, $*$. Тогда останется просто проверить строки на равенство.

Задача G. Hokage

Заметим, что в одном множестве должны находиться параллельные прямые. Если мы рассмотрим две прямые $y = k_1x + b_1$ и $y = k_2x + b_2$, то они будут параллельны, если $k_1 = k_2$ (не забываем также про важное условие, что все прямые различны). Тогда, на самом деле, нам нужно найти количество различных угловых коэффициентов k среди данных нам прямых, это и будет ответом на задачу. Требуемое количество можно найти с помощью структуры данных «множество» или «словарь».

Задача H. Поликарп и malware

Эту задачу можно решать по-разному. Главное — реализовать решение с асимптотикой лучше, чем $O(L^2)$, где L — это длина текста письма.

Один из возможных подходов: сохраним все опасные слова в множество (в C++ и Python подходящая структура данных называется `set`). Затем пройдемся по тексту письма, разобьём его на слова, из каждого слова удалим цифры (это можно сделать за $O(len)$, где len — это длина слова, если вместо непосредственного удаления цифр из слова с помощью встроенных в язык строковых методов, создать новое пустое слово и, проходя по старому, добавлять в новое слово текущий символ, если он не является цифрой), переведём его буквы в нижний регистр и поищем слово в множестве опасных. Затем, если оно нашлось в множестве опасных, добавим его в множество *найденных в тексте* опасных слов. После того, как все слова в тексте проверены, сравним множество опасных слов и множество найденных в тексте опасных слов. Если эти множества равны, то ответ YES, иначе — NO.

Задача I. ALL IN

Заметим, что можно сначала взять все нужные карты в руку, а только потом начать их выкладывать, от этого количество сделанных ходов не увеличится. Также заметим, что выложить карты в возрастающей последовательности длины k можно тогда и только тогда, когда в руке есть хотя бы k

карт с различными числами. Теперь задача сводится к тому, чтобы взять минимальное количество карт так, чтобы в руке оказалось k карт с различными числами. Для этого можно использовать `set` или `unordered set`, и идти по массиву чисел на картах, пока в `set` не окажется k элементов (то есть из взятых карт найдется k с различными числами). Решение за $\mathcal{O}(n \log n)$ или $\mathcal{O}(n)$.

Задача J. Everything, Black and White, No Grey

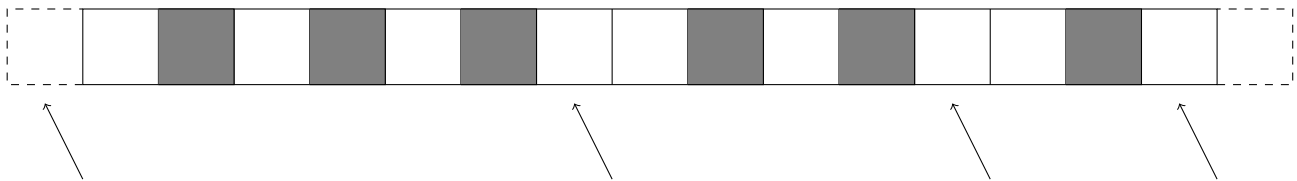
Так как в строке нет подстроки 01, то сначала в ней должны идти все 1, а затем все 0. Давайте пойдём по строке слева направо и будем поддерживать количество cnt нулей на префиксе и текущий ответ ans .

Если $s_i = 0$, то мы просто увеличиваем количество нулей ($cnt = cnt + 1$) и идём дальше, так как этот ноль идет в конце рассмотренного нашего префикса и не меняет ответ.

Если же $s_i = 1$, то мы изменяем ans ($ans = ans + cnt$), так как эту единицу нам нужно будет поменять местами с cnt нулями перед ней.

Задача K. Пилот

Предположим есть такое i , что ничего не подключено к розеткам $(i-1)$, i и $(i+1)$. Тогда можно за одно действие включить новый прибор в розетку i и это, очевидно, будет оптимально. Иначе таких i нет. Как выглядят занятые розетки в таком случае? Утверждается, что тогда соседние позиции i от 0 до n включительно такие, что i -я и $(i+1)$ -я розетки свободны, будут находиться на нечётном расстоянии большем 1. На рисунке ниже приведён пример для $n = 15$. Занятые розетки отмечены серым. Позиции, о которых говорится двумя предложениями ранее — стрелками.



Давайте найдём две соседние позиции l и r , отмеченные стрелками, такие, что $r - l$ минимально. Сдвинем все приборы между ними на 1 влево и на освободившееся место подключим устройство $m + 1$. Сколько операций мы потратим? $r - l$.

Теперь покажем, что это оптимальный ответ. Пусть есть ответ с $x < r - l$ операциями. Его всегда можно представить следующим образом: $y = \frac{x-1}{2}$ раз отключаем приборы, y раз подключаем их обратно в другие розетки, подключаем $(m + 1)$ -й прибор. Значит достаточно доказать, что нет ответов с $y < \frac{r-l-1}{2}$. От противного. Пусть такой ответ есть, тогда он должен удалять по крайней мере все приборы между какими-то двумя позициями, отмеченными стрелками (иначе новое место не появится), но тогда $y \geq \frac{r-l-1}{2}$.

Найти все позиции, где можно сделать такой сдвиг, и выбрать среди них две ближайшие можно за $\mathcal{O}(n)$. Всё остальное решение тривиально линейное, итоговая асимптотика — $\mathcal{O}(n)$.

Задача L. Ягами Лайт и страницы из Тетради смерти

Если опустить всю легенду, то в задаче требуется для каждого из q массивов сказать, является ли он подпоследовательностью данного массива a .

Рассмотрим алгоритм проверки одного списка t длиной m . Будем идти указателем i по массиву a слева направо, поддерживая указатель j на элемент t со следующим инвариантом: максимальный префикс t , который встречается в качестве подпоследовательности в префиксе a длиной i , заканчивается в элементе j . Изначально $i = 0$, $j = 0$ (мы работаем в 1-индексации). При переходе от i к $i + 1$ проверим, совпадает ли $a[i + 1]$ с $t[j + 1]$, если да — прибавим к j единицу, иначе ничего делать не будем. Если в какой-то момент j станет равен m , то t является подпоследовательностью a , иначе — нет.

Теперь поймем, что можно решать задачу одновременно для нескольких запросов. Изначально скажем, что ответ на каждый из них — NO. Теперь, также как и в предыдущем случае, будем идти указателем i слева направо по a , но будем поддерживать уже q указателей. Для каждого $1 \leq k \leq q$ указатель j_k смотрит на элемент массива из k -го запроса (далее будем называть этот массив b_k).

Тогда, чтобы корректно сделать переход от i к $i + 1$, нужно пройти по всем j_k и, если $a[i + 1] = b_k[j_k + 1]$, то прибавить к j_k единицу. Заметим, что это можно делать, проходя только по k , для которых нужно передвинуть соответствующий указатель. Для этого будем поддерживать массив cnt , в ячейке $cnt[x]$ будем хранить список всех индексов k , для которых $x = b_k[j_k + 1]$. Тогда переход $i \rightarrow i + 1$ выглядит так: для каждого элемента k из списка $cnt[a[i + 1]]$ увеличиваем j_k на единицу. Если после этого j_k не смотрит на конец b_k , то добавляем в $cnt[b_k[j_k]]$ значение k , а иначе говорим, что ответ на k -й запрос — YES. После не забываем удалить из массива $cnt[a[i + 1]]$ элемент k .

Задача М. Чёрная металлургия

В задаче есть два массива a и b , а также три типа запросов:

- $a[i] = a[i] + x$ для всех $i \in [L, R]$,
- $b[i] = b[i] + x$ для всех $i \in [L, R]$,
- вычислить $\sum_{i=L}^R a_i \cdot b_i$.

Будем использовать дерево отрезков с массовыми операциями. В вершине v , которой соответствует полуинтервал $[tl, tr)$ исходного массива, будем хранить:

- $A_v = \sum_{i=tl}^{tr-1} a_i$;
- $B_v = \sum_{i=tl}^{tr-1} b_i$;
- $T_v = \sum_{i=tl}^{tr-1} a_i \cdot b_i$.

Ещё в вершине нужно будет хранить отложенные операции прибавления, как в обычном дереве отрезков с массовыми операциями.

При прибавлении x ко всем a_i для $i \in [tl, tr)$ меняются A_v и T_v :

- $A_v = A_v + x \cdot (tr - tl)$,
- $T_v = T_v + x \cdot B_v$.

Прибавление x к b_i делается аналогично.

Разберёмся чуть подробнее с обновлением T_v . Изначально $T_v = \sum_{i=tl}^{tr-1} a_i \cdot b_i$. При прибавлении x ко всем a_i сумма становится следующей $T_v = \sum (a_i + x) \cdot b_i = \sum (a_i \cdot b_i + x \cdot b_i) = \sum a_i \cdot b_i + x \cdot \sum b_i = \sum a_i \cdot b_i + x \cdot B_v$. Из итогового выражения видно, что к T_v просто прибавилось $x \cdot B_v$.

Все требуемые запросы делаются за $\mathcal{O}(\log n)$, поэтому итоговая асимптотика решения — $\mathcal{O}(q \log n)$.

Задача Н. Игра в солдатиков

Для решения задачи можно использовать динамическое программирование. Заведём динамику $dp_{i,j}$, которая будет обозначать минимальное количество выкидываний солдатиков на префиксе солдатиков $[1, i]$ таким образом, что последняя набираемая возрастающая последовательность заканчивается ровно на солдате с номером i , и её длина ровно j . Заметим, что пересчёт $dp_{i,j}$ будет выглядеть следующим образом:

- Для всех t : $1 \leq t < i$, $a_t < a_i$, $j > 1$: $dp_{i,j} = \min(dp_{i,j}, dp_{t,j-1} + (i - t - 1))$,
- Для всех t : $1 \leq t < i$, $j \geq k$: $dp_{i,1} = \min(dp_{i,1}, dp_{t,j} + (i - t - 1))$.

В формулах пересчёта $(i-t-1)$ обозначает, что мы выкинули всех солдатиков с номерами между t и i . Если пересчитывать всё наивным образом, можно получить решение за $O(n^3)$ или $O(n^2 \cdot k)$, но этого не хватает, чтобы уложиться в ограничения по времени.

Заметим, что для пересчёта во втором случае можно запоминать минимальное значение $dp_{t,j}$ для всех t и $j \geq k$. Потом, когда мы хотим посчитать значение $dp_{i,1}$ можно пробежаться по всем $t < i$ и взять запомненное значение $+(i-t-1)$ и найти минимум по всем таким значениям, что приводит к асимптотике $O(n^2)$.

Для пересчёта в первом случае нужно заметить, что для фиксированных i и j минимум будет браться по $dp_{t,j-1}+(i-t-1)$, где $t < i$ и $a_t < a_i$. Можно вынести константу из минимума, получим, что нас интересует минимум по $dp_{t,j-1}-t$ (затем к нему нужно добавить $i-1$, чтобы получить корректное значение). Так как все $1 \leq a_x \leq n$, для таких запросов можно использовать дерево отрезков для каждого j , в котором в листах для каждого значения от 1 до n мы будем поддерживать минимальное $dp_{t,j}-t$ по всем t , где a_t равно значению, за которое отвечает лист. Будем постепенно добавлять в дерево отрезков новые значения в порядке возрастания t параллельно с подсчетом динамики (чтобы при запросе выполнялось $t < i$). Теперь пересчёт для рассматриваемого случая свёлся к запросу минимума к нужному дереву отрезков на отрезке $[1, a_i-1]$, а затем обновлению в точке a_i значением $dp_{i,j}-i$. Такое решение даст асимптотику $O(n^2 \log n)$ или $O(nk \log n)$ (в зависимости от того, создавать k или n деревьев отрезков (так как на самом деле $j \geq k$ можно аккуратно рассматривать как один случай $j = k$)).

Для восстановления ответа нужно в вершинах деревьев отрезков поддерживать не только минимум, но и i , из которого он оказался достижим (где i — индекс в массиве солдатиков, который дал лучшее значение динамики $+i$), и в самой динамике поддерживать состояние, из которого мы получили оптимальный результат.

Итоговое решение за $O(nk \log n)$.

Задача О. Автодополнение

Заметим, что последовательность строк, которые выводятся на экран телефона Кена, удобно представлять в виде бора: пусть сейчас набрана строка Q . Тогда нажатие на клавишу с буквой — это спуск по соответствующей букве, нажатие на **backspace** — переход в предка (если он есть), а автодополнение — «прыжок» на несколько вершин вниз.

Построим бор на слове w , которое нам нужно набрать, и всех строках из списка L . Теперь для каждой вершины v научимся считать go_v : список пар, хранящий значения вида (u, cnt) , где u — вершина бора, а cnt — количество вариантов за одно действие перейти из v в u . Но если строить бор честно, такой список не будет однозначным, так как он будет зависеть от действия, за счёт которого Кен попал в v . Поэтому давайте скажем, что для каждой вершины v переход по **backspace** ведет не в её предка, а в новую вершину del_v , которая будет почти полной копией предка. Единственное, что их будет отличать — список go .

Тогда go для каждой вершины можно построить однозначно: для обычных вершин нужно будет объединить информацию о переходах в боре с допустимыми вариантами автодополнения (терминальными вершинами из поддерева v), а для вершин вида del_v нужно сначала добавить в go_{del_v} переходы, которые есть в go_{parent_v} , но нет в go_v , а потом те, которые есть и в go_{parent_v} , и в go_v ($parent_v$ — предок вершины v в боре).

Осталась последняя проблема: Кен, стоя в любой вершине, может набрать букву, перехода по которой в боре нет. Чтобы обрабатывать такие случаи, для каждой вершины v создадим $\frac{A}{2}$ дополнительных (назовем их $extra_v$), i -я из них будет соответствовать случаю, когда Кен набрал букву, перехода по которой из v нет, а потом еще i каких-то (не обязательно за i действий). Мы не будем честно строить переходы для этих букв, зато построим go .

1. Если v — обычная вершина бора, то всё понятно: в go_v добавим пару вида $(extra_v[0], 26 - child_v)$, где $child_v$ — количество детей v в боре без учёта переходов по **backspace**, $go_{extra_v[0]} = \{(v, 1), (extra_v[1], 26)\}$, а для каждого $i > 0$ $go_{extra_v[i]} = \{(extra_v[i-1], 1), (extra_v[i+1], 26)\}$.

2. Если же v — это копия какой-то вершины u , которую мы создали для перехода по **backspace** ($v = del_l$, $u = parent_l$), то после того, как мы, находясь в v , наберём несколько букв, перехода по которым нет в боре, и сотрём их, список слов для автозаполнения будет таким же, как у вершины u . В терминах нашего бора это будет означать, что из $extra_v[0]$ при нажатии **backspace** мы должны перейти в u . Чтобы рассмотреть этот случай, назначим $extra_v = extra_u$. Так как $extra_u$ уже построены, достаточно добавить в go_v пару $(extra_u[0], 26 - child_v)$.

Теперь мы свели задачу к тому, что у нас есть бор, для каждой пары вершин известно количество вариантов перехода между ними, и нам нужно посчитать количество вариантов попасть в одну из отмеченных вершин за A действий. Для этого напомним динамику $dp[v][cnt]$, в которой будем хранить количество способов за cnt действий попасть в вершину бора v . База, переходы и ответ в таком формате становятся тривиальными.