Задача A. Парад планет

Скорость, с которой увеличивается угол между планетами, равна abs(v2 - v1). Нам нужно, чтобы угол стал равен 360 градусов. Это случится ровно через 360 / abs(v2 - v1) дней. Целая часть от этой дроби плюс единица и даёт искомый номер дня. Итак, ответ на задачу равен 1 + 360 div abs(v2 - v1).

Задача B. Парад планет - 2

Найдём скорости, с которыми меняется угол между каждой парой планет: $d_{12} = abs(v_2 - v_1)$, $d_{13} = abs(v_3 - v_1)$, $d_{23} = abs(v_3 - v_2)$.

Угол между планетами 1 и 2 равен нулю в моменты времени 0, $360/d_{12}$, $2 \cdot 360/d_{12}$, $3 \cdot 360/d_{12}$ ... и так далее. Аналогично будет для пары планет 1 и 3, а также 2 и 3.

Ближайшее положительное время, когда угол между каждой парой планет станет равен нулю – это такое минимальное число t, что t делится нацело на дроби $360/d_{12}$, $360/d_{13}$ и $360/d_{23}$. *Примечание: в каком-то смысле, нам нужно найти наименьшее общее кратное дробей (если ввести для дробей такое понятие).*

Очевидно, что искомое t – рациональное число. Будем искать его в виде $t = a/b$. Итак, нам нужна минимальная дробь $a/b > 0$, что значения $(a/b)/(360/d_{12})$, $(a/b)/(360/d_{13})$ и $(a/b)/(360/d_{23})$ являются целыми числами. Другими словами, $a \cdot d_{12}$ делится на $b \cdot 360$, $a \cdot d_{13}$ делится на $b \cdot 360$, и $a \cdot d_{23}$ тоже делится на $b \cdot 360$.

Наименьшее положительное a, чтобы $a \cdot d_{12}$, $a \cdot d_{13}$ и $a \cdot d_{23}$ делились на 360, можно найти так: $a = 360/gcd(d_{12}, d_{13}, d_{23}, 360)$, где функция gcd – это наибольший общий делитель.

Наибольшее положительное b, при котором $a \cdot d_{12}$, $a \cdot d_{13}$ и $a \cdot d_{23}$ делятся на $b \cdot 360$, находится так: $b = gcd(a \cdot d_{12}/360, a \cdot d_{13}/360, a \cdot d_{23}/360)$.

Ответ на задачу равен $1 + a \ div \ b$.

Задача C. Прыжки со скакалкой

Первый способ решения. Пусть x — сколько полных циклов (по k раз) сделал спортсмен. Сумма по большим прыжкам: 1+2+...+x = (1+x)*x/2. Сумма по всем прыжкам: (k-1)*x. Получаем неравенство: (k-1)*x + (1+x)*x/2 <= n. Требуется найти максимальное целое x, при котором неравенство верно.

Превратим это неравенство в уравнение: (k-1)*x + (1+x)*x/2 = n. Решим это квадратное уравнение и возьмём его неотрицательный корень x. Итак, у нас уже есть x*k прыжков, дающих сумму cur = (k-1)*x + (1+x)*x/2. Осталось определить, сколько ещё прыжков осталось сделать до n. Если (n - cur) < k, то осталось сделать (n - cur) прыжков, а иначе — k прыжков, так как последний прыжок будет большим.

Второй способ решения. Можно просто промоделировать процесс, только одиночные прыжки добавлять не по одному, а сразу группой (иначе будет предел времени).

Задача D. Наименьшая сумма

При n=2 ответом будет ‘1+2’. Во всех остальных случаях можно получить сумму 1 или 2. Если сумма всех чисел от 1 до n нечётна, то всегда можно получить число 1. Если сумма всех чисел от 1 до n чётна, то получить 1 невозможно, поскольку чётность суммы не меняется при изменении ‘+’ на ‘-’ или ‘-’ на ‘+’. В этом случае всегда можно получить 2.

Рассмотрим два способа решения.

В первом способе выделим несколько случаев.

- n – чётное. Ответ строится так: 1+2+3-4(+5-6)(-7+8)(+9-10)(-11+12)(+13-14)... Начиная с 5, все числа разбиваются на пары, а знаки ставятся так, чтобы суммы в парах чередовались: -1, 1, -1, 1...
- n – нечётное, (n-1) делится на 4. Ответ строится так: 1-2+3(+4-5)(-6+7)(+8-9)(-10+11)(+12-13)... Начиная с 4, все числа разбиваются на пары, а знаки ставятся так, чтобы суммы в парах чередовались, начиная с -1: -1, 1, -1, 1...

- n – нечётное, (n-1) не делится на 4. Ответ строится так: 1-2+3(-4+5)(+6-7)(-8+9)(+10-11)(-12+13)(+14-15)... Начиная с 4, все числа разбиваются на пары, а знаки ставятся так, чтобы суммы в парах чередовались, начиная с 1: +1, -1, +1, -1...

Ещё более простой способ решения — жадный алгоритм. Пусть S — сумма всех чисел от 2 до n. С помощью жадного алгоритмы построим набор чисел от 2 до n, в сумме дающих S div 2. Для этого будем идти по убыванию от n до 2 и добавлять очередное число к текущей сумме, если она ещё не превышает S div 2. Перед взятыми числами будем ставить знак '-', перед остальными — знак '+'.

Задача E. Игра

Когда может выиграть первый?

Если $a < b$, то он точно выигрывает. Если $n <= 2a$, то он выигрывает любым первым ходом. Если $n > 2a$, то выигрышная стратегия следующая. Отступим a клеток от левого края, создав тем самым себе «резерв», и зачеркнём клетки от $a + 1$ до $2a$. Теперь, как бы ни сходил второй игрок, первый либо тоже найдет, куда сходить справа от резерва, либо он сходит в оставленный резерв (но тогда второму ходить дальше будет некуда).

Если $a > b$, то в некоторых случаях первый игрок всё равно может выиграть. Один из вариантов – если у него есть возможность первым ходом сходить так, чтобы второму ходить было некуда. Для этого первым ходом зачеркнём центр полоски. Если после этого и слева, и справа осталось меньше b клеток, то первый выиграл.

Если и это условие не выполнилось, то у первого всё ещё остаются шансы выиграть. Для этого нужно, чтобы после вычёркивания центра осталось и слева, и справа от a до $2b - 1$ клеток. Тогда, если второй игрок сходит влево, то первый третьим ходом сходит вправо, и наоборот. После этого второму игроку ходить будет некуда.

Задача F. Пропущенная цифра

При $n < 11$ ответ можно найти «в лоб». Не забываем частный случай: для строки «*20» ответ находится однозначно: он равен 720, а не 120, так как по условию задачи $n \geqslant 6$.

При $n \geqslant 11$ воспользуемся признаком делимости на 11: сумма цифр в чётных позициях минус сумма цифр в нечётных позициях дожна делиться на 11.

Задача G. Отрезки

Задача решается следующим жадным алгоритмом. Первым берём отрезок с самым маленьким правым концом. На каждом следующем шаге берём отрезок с самым маленьким правым концом среди тех оставшихся отрезков, которые пересекаются с последним взятым. Если никакой из оставшихся отрезков не пересекается с последним взятым, то решений нет.

Рассмотрим два способа обоснования корректности алгоритма.

Способ 1 – вычислительный эксперимент.

Переберём все возможные варианты взаимного расположения отрезков во входных данных и проверим, что вышеописанный алгоритм во всех случаях работает верно. Реализация перебора выглядит так. Будем генерировать отрезки в порядке неубывания правого конца, а для одинаковых правых концов – в порядке неубывания левых.

Заметим, что можно не рассматривать случай, когда отрезки пересекаются точно концами. По условию задачи, у нас нет отрезков-точек. Следовательно, если во входных данных какие-то отрезки пересекаются в концах, то можно немного подвигать их концы влево-вправо, и результатом пересечения будет уже не точка, а небольшой отрезок.

Тогда перебор выглядит так. Пусть первый отрезок имеет вид (0.0, 1.0). При желании результат несложно перевести в целочисленные координаты. У каждого следующего отрезка возьмём правую границу на 1 больше, чем у предыдущего. А варианты расположения левой границы переберём. Во-первых, это будет координата на единицу левее самой левой из имеющихся. Во-вторых, переберём все промежуточные точки. Например, для второго отрезка получатся следующие варианты: (-1.0, 2.0), (0.5, 2.0), (1.5, 2.0). И это все возможные варианты взаимного расположения двух отрезков. Аналогично переберём левый конец для третьего отрезка, и так далее.

Получив очередной вариант входных данных, найдём ответ вышеописанным алгоритмом. Если решение нашлось, то всё хорошо. А если оно не нашлось, то переберём все перестановки этих отрезков, чтобы убедиться, что решения действительно нет.

Такой подход позволяет за разумное время проверить где-то 8-9 отрезков. Однако, можно оптимизировать перебор. Например, можно применить динамическое программиование по подмножествам. Пусть у нас уже построено какое-то начало цепочки. Если мы когда-то ранее выяснили, что для подмножества оставшихся отрезков цепочки не существует, то перебор можно не продолжать (сделать отсечение). Этот способ позволил нам перебрать 11 отрезков за несколько часов.

Конечно, описанным способ не является полным доказательством. Но вероятность, что до 11 отрезков алгоритм работает верно, а для большего количества вдруг найдётся специфический частный случай, крайне небольшая.

Способ 2 – математическое доказательство.

Пусть какое-то решение существует. Будем на каждом шаге удалять из него очередной отрезок, выбранный согласно описанному алгоритму (и добавлять его в конец ответа). Но перед удалением будем изменять это решение так, чтобы после удаления отрезка оно для оставшихся отрезков осталось корректным. Покажем, что при этом гарантированно каждый раз будет находиться отрезок, пересекающийся с предыдушим, пока все отрезки не закончатся.

Поясним суть с помощью рисунка.

```
         --
k-1    ___
k  __________
k+1         __
           ____
q_1 ______________
q  ** (удалённый на прошлом шаге)
q+1 ____________
       --
```

Сверху вниз показана цепочка из оставшихся отрезков, образуюшая какое-то решение. Обозначения:

- q – позиция в ней отрезка, который был удалён на предыдущем шаге,
- k – позиция отрезка с самым маленьким правым концом, который пересекатся с q. Он запланирован к удалению на текущем шаге.

Рассмотрим случай k<q (противопожный случай аналогичен).

Заметим, что если просто удалить отрезок k, то решение для осташихся отрезков можеть стать некорректным, поскольку отрезки k-1 и k+1 могут и не пересекаться. Когда это может произойти?

Посмотрим, как может отрезок k располагаться относительно q. Его правый конец может быть как больше правого конца q, так и меньше или равен. Но, если правый конец k меньше или равен правому концу q, тогда отрезки k-1 и k+1 обязательно пересекаются. Они могли бы не пересекаться, только когда отрезок k-1 или k+1 (или оба) кончается раньше отрезка k. Но, если какой-из них пересекается с q, то он бы и был выбран вместо k. А если он не пересекается с q, значит, кончается раньше начала q. Но такой отрезок был бы взят на предыдущих шагах алгоритма. Итак, если k <= правого конца q, то удаление k оставляет нам корректное решение – отрезки k-1 и k+1 пересекаются.

Теперь рассмотрим ситуацию, когда k кончается правее, чем q. Теперь отрезки k-1 и k+1, действительно, могут не пересекаться – это показано на рисунке выше.

Если у нас как раз такой случай, то будем действовать так: возьмём цепочку отрезков от k до q-1, и развернём её. При этом решение осталось корретным. Действительно, отрезки k и q+1 пересекаются, так как они оба пересекаются с q и кончаются не раньше, чем q. Отрезки q-1 и k-1 тоже пересекаются. Это следует из того, что отрезок q-1 начинается не позже конца отрезка q и кончается не раньше конца отрезка k, поэтому он по-любому пересекается с k-1, который кончается не раньше конца q и пересекается с k.

Осталось заметить, что, поскольку решение каждый раз восстанавливается, то у нас на каждом шаге гарантированно будет находиться очередной отрезок - как минимум, предыдущий или следующий в цепочке перед отрезком, что был удалён.

Задача H. Отрезки - 2

Задача отличается от задачи «Отрезки» только увеличенными ограничениями. Напомним жадный алгоритм решения. Вначале берётся отрезок с самым маленьким правым концом. На каждом следующем шаге берём отрезок с самым маленьким правым концом среди тех оставшихся отрезков, которые пересекаются с последним взятым. Если никакой из оставшихся отрезков не пересекается с последним взятым, то решения нет. Обоснование корректности смотрите в разборе задачи «Отрезки».

Описанный алгоритм имеет квадратичную сложность. Чтобы её уменьшить, нужно придумать, как реализовать поиск очередного отрезка более эффективно. Возможный вариант – использовать дерево отрезков.

Чтобы построить дерево отрезков, вначале выполним сжатие координат. Возьмём множество координат начал и концов всех отрезков. Самую маленькую из них заменим на 0, следующую по величине — на 1, и так далее. В результате все координаты не будут превышать $2 \cdot n$.

Создадим вектор из векторов byleft, где byleft[i] содержит вектор отрезков с началом $= i$, упорядоченный по убыванию правого конца.

Построим дерево отрезков с функцией минимума. Индексами будут начала отрезков, а значениями — концы. Если имеется несколько отрезков с одинаковым началом i, то из них берётся с наименьшим концом (а это как раз последний элемент вектора byleft[i]).

Реализуем функцию поиска минимума на отрезке так, чтобы она возвращала не сам минимум, а его позицию. Искомый отрезок тогда лежит в byleft[imin].back(). Добавляем этот отрезок в ответ. Убираем его из byleft: byleft[imin].pop_back(). Обновляем значение в дереве: если byleft[imin] стал пустым, то в в позицию imin ставим бесконечность, иначе ставим туда byleft[imin].back().right.

В результате мы можем находить отрезок с наименьшим правым концом, пересеающийся с предыдущим, за $O(log(n))$. Сложность всего решения — $O(n \cdot log(n))$.

Задача I. Неделящиеся числа

Возможный ответ: 2*n-1, 2*n-2, ..., n.

Понятно, что никакое из этих чисел не делится ни на какое другое. Чтобы доказать, что максимальное число не может быть меньше 2*n-1, докажем сначала следующее вспомогательное утверждение.

Среди всех целых чисел от 1 до k можно выбрать не более (k+1) div 2 чисел, попарно не делящихся друг на друга. Это, например, числа от k div 2 + 1 до k включительно. Для доказательства разобьём все числа от 1 до k на классы эквивалентности. Два числа относятся к одному классу, если одно можно получить из другого домножением на степень двойки. Например, 1, 2, 4, 8 – один класс, 3, 6, 12 – другой класс, и так далее.

Все числа в интервале от k div 2 + 1 до k относятся к разным классам эквивалентности. Любое же число, меньшее или равно k div 2, относится к одному из этих классов (поскольку умножением его на степень двойки мы всегда можем попасть в интервал от k div 2 + 1 до k). Поскольку представитель каждого класса эквивалентности может присутствовать в решении только в одном экземпляре, то решения большего размера не существует.

Из этого утверждения непосредственно следует, что в исходной задаче максимальное число не может быть меньше, чем 2*n-1.

Задача J. Красивое слово

Разобьём кубики на три типа: содержащие только гласные буквы, только согласные, и оба вида. В каждом кубике первого типа оставим только самую маленькую гласную букву, второго типа — самую маленькую согласную, третьего типа — самую маленькую гласную и согласную.

Пусть v, c и b — количество кубиков первого, второго и третьего типа. Найдём максимальную длину слова, которую можно получить. Рассмотрим случай, когда слово начинается с гласной (с

согласной делается аналогично). Если b >= abs(v-c), то искомая длина равна v+c+b. В противном случае кубики третьего типа будем считать гласными, если гласных меньше, и согласными, если наоборот. То есть, если v < c, то v += b, иначе c += b. Теперь, если v <= c, то длина слова равна 2 * v, иначе она равна 2 * c + 1.

Осталось построить слово буква за буквой. Пусть мы уже построили некоторый префикс, и пусть он заканчивается согласной буквой (для гласной — аналогично). Тогда нам надо подобрать следующую гласную букву. Перебираем гласные буквы по возрастанию. Если имеется кубик первого типа с такой буквой, то сразу кладём его в ответ. Если кубика первого типа нет, но есть третьего, то проверяем, а получится ли достроить остаток слова до нужной длины из оставшихся кубиков, если сейчас убрать этот. Такую проверку мы умеем делать за константу, как описано выше. Если удаётся, то кладём кубик в ответ. Заметим ещё, что если есть несколько подходящих кубиков третьего типа с такой гласной буквой, то надо брать тот из них, что с максимальной согласной.

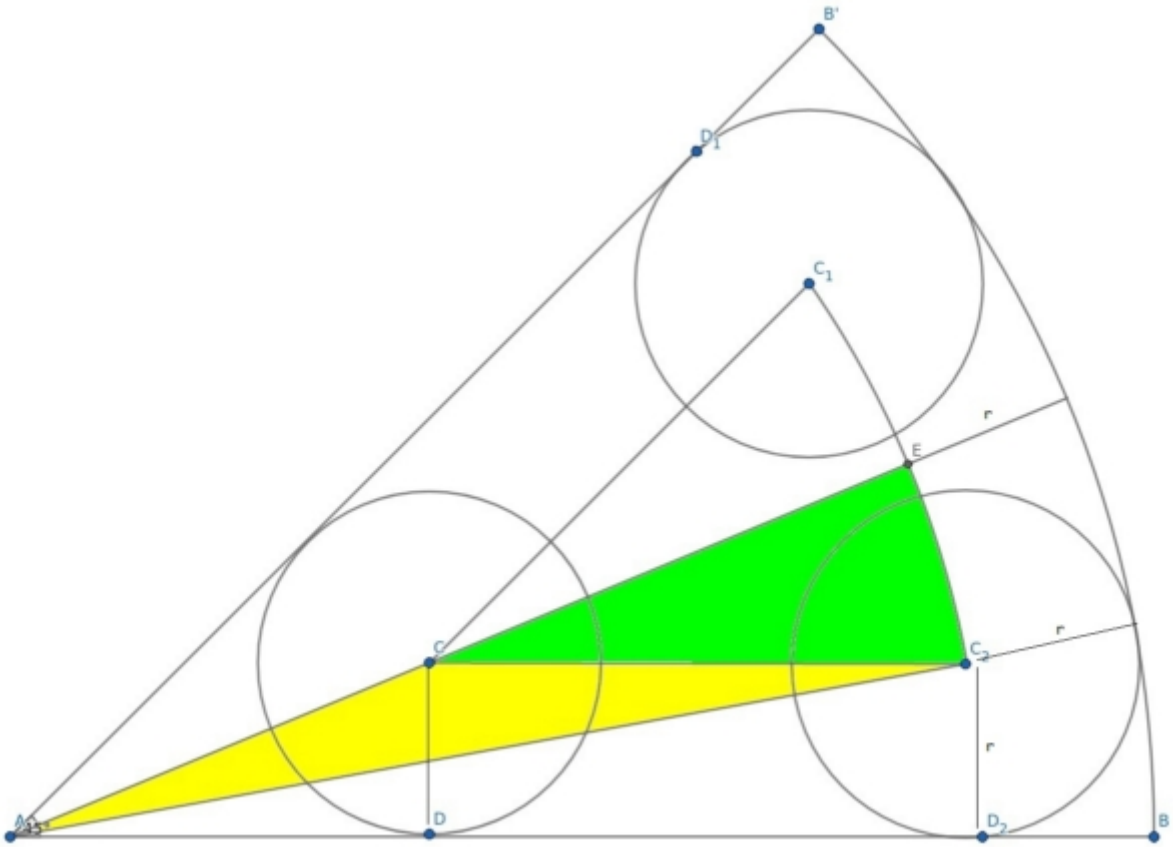
Можно отдельно найти лучшее слово, начинающееся с гласной и с согласной, а затем выбрать наилучшее из них.

Задача K. Пепперони

При таком случайном выборе точек, как описано в условии, они будут распределены равномерно по всей площади окружности. Следовательно, задачу можно свести к отношению площадей.

Пусть good_area – площадь такой части фигуры, что при попадании в неё центра ломтика он не пересекает разрезов и края окружности. Пусть total_area – площадь всей пиццы. Тогда ответ равен (total_area - good_area) / total_area.

Нахождение good_area проиллюстрировано на рисунке – это площадь зелёной фигуры, умноженная на 16. Возможный подвох: зелёная фигура похожа на сектор круга, но сектором не является: дуга EC_2 имеет центр в точке A, а не в точке C.



Чтобы найти площадь зелёной фигуры, можно из площади сектора AEC_2 вычесть площадь треугольника ACC_2. Возможная последовательность действий:

- находим AC, зная угол $\pi/8$ и катет $CD = r$. Если $AC + r \geqslant R$, то ответ равен 1.0.

- находим AD по теореме Пифагора.
- находим $AC_2 = R - r$.
- находим AD_2 по теореме Пифагора (заметим, что AD_2 не равно $R - r$).
- находим $CC_2 = DD_2 = AD_2 - AD$.
- находим площадь треугольника ACC_2, зная все три его стороны.
- находим угол треугольника ACC_2 у вершины A по теореме косинусов.
- находим площадь сектора AEC_2.
- находим площадь зелёной фигуры как разность площадей сектора AEC_2 и треугольника ACC_2.