Задача A. Бой с числами

Идея: Иван Казменко
Разработка: Иван Казменко

Заметим, что после переворачивания самая значащая цифра в двоичной записи — а это всегда единица — становится последней. При вычитании эта единица заменяется на ноль. Значит, каждый суперудар Звёздочки уменьшает количество единиц в двоичной записи ровно на 1.

Ответ — количество единиц в двоичной записи числа n.

Задача B. Время магического числа

Идея: Николай Дубчук
Разработка: Николай Дубчук

Два основных подхода в этой задаче:

- можно сделать перебор и проверить каждое время на причастность к 239;
- можно подсчитать математически, сколько будет удачных сочетаний.

Задача C. Инфляция

Идея: Николай Дубчук
Разработка: Николай Дубчук

Можно решить формулой $n \cdot 5.5 \cdot p/100$. Либо циклом: в первый месяц теряем $n \cdot p/1000$, во второй теряем $2 \cdot n \cdot p/1000$, ..., в десятый — $10 \cdot n \cdot p/1000$.

Задача D. Девять из десяти

Идея: Антон Майдель
Разработка: Антон Майдель

Рассмотрим подзадачу максимизации числа успешных экспериментов. Множество успешных экспериментов состоит из двух непересекающихся подмножеств: безумный учёный верно определил успешный эксперимент и безумный учёный неверно определил, что эксперимент неуспешный. Так как эти подмножества не пересекаются, то можно максимизировать их размер независимо друг от друга. По условию задачи, размер множества верно определённых результатов экспериментов равен $\frac{n}{10}$, а по условию, размер множества экспериментов, которые учёный определил как успешные, равен x. Для максимизации числа верно определённых успешных экспериментов мы, пользуясь тем, что эксперименты между собой независимы и неотличимы друг от друга, получим, что результат будет равен $\min(\frac{n}{10}, x)$. Формула для второго подмножества соответственно: $\min(\frac{9n}{10}, n - x)$. Сложив эти два минимума, мы получим формулу для нашей подзадачи: $\min(\frac{n}{10}, x) + \min(\frac{9n}{10}, n - x)$. Вторая подзадача минимизации числа успешных экспериментов эквивалентна задаче максимизации числа неуспешных экспериментов, которая решается аналогично первой подзадаче, и итоговая формула для подзадачи минимизации выглядит так: $n - \min(\frac{n}{10}, 10 - x) - \min(\frac{9n}{10}, x)$.

Задача E. Грядка

Идея: Николай Дубчук
Разработка: Николай Дубчук

Для нахождения наименьшей площади минимальный размер доски нужно умножить на сумму всех остальных. Для наибольшей — решить задачу о рюкзаке (набираем доски так, чтобы было ближе к квадрату). Решать можно полным перебором.

Интересный факт

Если разрешить парные доски ставить, как угодно, а не напротив друг друга, то для наибольшей площади решение можно улучшить.

Пример: (1 11 101 111 200 20)

Можно построить квадрат со стороной 222: $200 + 20 + 1 \cdot 2 = 111 \cdot 2 = 101 \cdot 2 + 20 = 11 \cdot 2 + 200$.

Если парные доски надо ставить напротив друг друга, то доску 111 не добить до стороны 222, так как нет способа набрать 111 подмножеством чисел 1, 11, 101, 20, 200.

Задача F. Наибольшая площадь

Идея: Михаил Иванов
Разработка: Михаил Иванов

Наибольший эллипс, вмещающийся в квадрат — это круг. Чтобы получить ответ для произвольного прямоугольника, надо перевести аффинным преобразованием квадрат в прямоугольник, тогда и круг перейдёт в эллипс. При этом аффинные преобразования сохраняют соотношения площадей, из этого следует, что эллипс получится наибольшей площади.

Из этого рассуждения мы можем получить, что отношение площади наибольшего эллипса к площади содержащего его прямоугольника постоянно. Для квадрата соотношение равно $\frac{\pi}{4}$, значит, решение задачи — посчитать площадь прямоугольника (например, при помощи псевдоскалярного произведения векторов двух его соседних сторон) и домножить её на $\frac{\pi}{4}$.

Задача G. За границы пианино

Идея: Михаил Иванов
Разработка: Михаил Иванов

Представим, что клавиатура бесконечна в обе стороны, пронумеруем её клавиши целыми числами. Заметим, что каждая следующая клавиша может быть однозначно определена своей буквой и предыдущей клавишей. Давайте построим маршрут на этой бесконечной клавиатуре (начиная с любого числа с правильным остатком по модулю 7) и сохраним достигнутые минимальное и максимальное значения. Затем, с прыжками по семь, попытаемся поместить эти минимальное и максимальное значения в отрезок $[0; 51]$: ответ будет «`YES`», если и только если это возможно. Для проверки мы добавим семёрку минимальное количество раз (возможно, отрицательное), чтобы минимальное значение оставалось (всё ещё) неотрицательным, и проверим, что максимальное значение после этого не превышает 51.

Задача H. Результаты соревнования

Идея: Анастасия Григорьева
Разработка: Владислав Макаров

У этой задачи есть очень много решений. Давайте разберём одно из них. Давайте поймём, какое место занял бегун 1. Все, кого он обогнал, имеют больший номер (потому что его номер является минимальным возможным). Следовательно, он занял место $n - a_1$. Запомним этот факт и представим, что бегун 1 в соревновании не участвовал (но номера всех остальных бегунов остались такими же). От этого числа a_j при $j \geqslant 2$ не поменяются: так как 1 — это минимальный возможный номер, бегун 1 не учитывался ни в одном из чисел a_j независимо от своего выступления. В соревновании без бегуна 1 наименьший номер имеет бегун 2. Следовательно, бегун 2 занял место $n - a_2$ в соревновании без бегуна 2. Это рассуждение можно продолжить для всех бегунов в порядке возрастания их номеров.

Как теперь решать задачу? Создадим массив всех возможных мест в порядке возрастания (изначально это числа от 1 до n в порядке возрастания). Когда мы рассматриваем бегуна i, мы говорим, что он занял место, находящееся на позиции $n - a_i$ в этом массиве, и удаляем это место из массива. Другими словами, мы можем сказать, что когда мы притворяемся, что бегун i никогда не участвовал в соревновании, мы позволяем ему «забрать с собой» его место. Например, бегун 2 занял место, находящееся на позиции $n-a_2$ в массиве $(1, 2, \ldots, n-a_1-2, n-a_1-1, n-a_1+1, n-a_1+2, \ldots, n)$. Наконец, мы получили массив, сопоставляющий каждому номеру бегуна его место. Построим обратный массив, сопоставляющий номерам бегунов их места. Это и будет ответ.

Данное решение работает за $O(n^2)$: нужно n раз удалить элемент откуда-то из середины массива, для чего нужно сдвинуть все последующие элементы на одну позицию влево. Есть и более быстрые решения, при этом их очень много. Если эта тема вам интересна, рекомендуем почитать любые материалы по ключевым словам «таблица инверсий».

Задача I. Кто я?

Идея: Никита Гаевой
Разработка: Никита Гаевой

Рассмотрим одного произвольного игрока. На случайном тесте этот игрок угадает свою карточку с вероятностью ровно $1/n$, где n — количество игроков за столом. Нам нужно, чтобы хотя бы один из игроков угадывал ответ с вероятностью 1, следовательно, в правильной стратегии ответ будет угадан ровно одним игроком. Таким образом, стратегии игроков должны иметь некий параметр, различный для всех игроков и принимающий ровно n возможных значений, из правильного значения которого игрок сможет восстановить свою карточку. В качестве такого параметра нам подойдет сумма чисел на всех карточках по модулю n.

Таким образом, на первом запуске мы выберем игрока, номер которого совпадает с суммой чисел на всех карточках, а на втором запуске, имея это число в качестве номера игрока, мы восстановим ответ.

Задача J. Заповедник

Идея: Анастасия Григорьева
Разработка: Владислав Макаров

Утверждается, что задача решается следующим образом: найдём среди наших точек две наиболее удалённые друг от друга и проведём через каждую из них прямую, перпендикулярную отрезку между ними. Понятно, что расстояние между такими прямыми будет в точности равно расстоянию между выбранными точками.

Почему при таком выборе полосы все точки действительно попадут либо внутрь неё, либо на её границу? Давайте это поймём. Пусть наша пара точек — это A и B (если пар, на которых достигается наибольшее расстояние, много, то можно взять любую из них). Пусть через A проходит прямая ℓ_A, перпендикулярная AB. Аналогично, через B проходит прямая ℓ_B, тоже перпендикулярная AB. Пусть какая-то из наших точек (назовём её C) не лежит между ℓ_A и ℓ_B. Не умаляя общности, C и B лежат строго по разные стороны от ℓ_A. Тогда отрезок CB пересекает прямую ℓ_A. Назовём точку пересечения P. Тогда $|BC| > |BP| \geqslant |BA|$. Первое неравенство выполняется, потому что P лежит строго на отрезке BC. Второе неравенство выполняется, потому что отрезок BA является перпендикуляром к ℓ_A и, тем самым, реализует кратчайшее расстояние от B до точки на ℓ_A. Следовательно, точки B и C более удалены друг от друга, чем A и B. Противоречие.

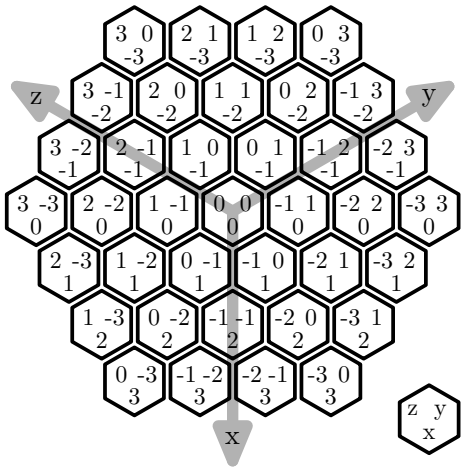
С другой стороны, получить ответ лучше невозможно. Действительно, пусть у нас есть две точки C и D, и через них проходят параллельные прямые ℓ_C и ℓ_D соответственно. Тогда расстояние между ℓ_C и ℓ_D не превосходит $|CD|$, так как отрезок CD соединяет параллельные прямые *как-то* и, следовательно, не длиннее расстояния между ℓ_C и ℓ_D по перпендикуляру. Ну а CD — *какой-то* отрезок между двумя данными точками. Следовательно, он не длиннее отрезка AB. Здесь даже неважно, что все наши точки должны лежать между ℓ_C и ℓ_D (и вообще, множество правильных ответов на задачу не поменяется, если это требование убрать).

Задача K. Операции с гексом

Идея: Иван Казменко
Разработка: Иван Казменко

Для начала научимся более-менее удобно работать с гексом. Вот один из способов.

Введём для клеток однородные координаты (x, y, z) по трём осям с углами в 120 градусов между ними. Для каждой клетки будет верно $x + y + z = 0$. Если условиться, что центр гекса в клетке $(0, 0, 0)$, то клетки гекса — это (x, y, z), для которых $|x|, |y|, |z| < n$.



Вот, например, псевдокод для чтения исходного гекса:

```
m := n - 1
create table a[m + n][m + n]
for x := -m, ..., m:
    for y := -m, ..., m:
        z := -x - y
        if -m <= z and z <= m:
            read a[m + x][m + y]
```

При этом гекс из примера хранится в массиве `a` так:

```
0 0 4 1 8
0 3 5 1 7
2 1 6 1 8
1 7 1 9 0
8 9 9 0 0
```

Нули — неиспользуемые клетки массива.

Любая другая операция с гексом выглядит в коде примерно так же. Вывод делается аналогично. А вот, например, операция «`R`» — поворот на 60 градусов по часовой стрелке:

```
create table b[m + n][m + n]
for x := -m, ..., m:
    for y := -m, ..., m:
        z := -x - y
        if -m <= z and z <= m:
            b[m + x][m + y] := a[m - y][m - z]
```

Операции «`L`» и «`T`» реализуются аналогично.

В задаче есть и вторая часть: как успеть сделать 250 000 операций с гексом размера 500.

Можно заметить, что доступные операции образуют группу преобразований над гексом, в которой всего 12 элементов. Другими словами, у гекса всего 12 различных состояний: 6 возможных поворотов и ещё 6 отражённых случаев.

Научимся вычислять операцию в этой группе, не производя саму операцию над гексом. Например, можно хранить состояние гекса в виде двух чисел: количество поворотов $rotate \in \{0, 1, 2, 3, 4, 5\}$ и отражение $flip \in \{-1, +1\}$. Тогда операция отражения изменяет значение *flip*, а операция поворота прибавляет к *rotate* (или отнимает) значение *flip* (по модулю 6).

В конце сначала применяем все повороты (их будет от 0 до 5), а затем делаем или не делаем отражение.

Задача L. Гипотеза Коллатца и случайные увеличения

Идея: Михаил Иванов
Разработка: Михаил Иванов

Существует множество способов решения этой задачи. Одно из семейств решений выглядит следующим образом: зафиксируем два параметра $w \in \mathbb{Z}_{>0}$ и $k \in [0; 1]$. Попробуем применить функцию Коллатца к текущему числу w раз. Если за это время мы достигли единицы, то применяем функцию Коллатца. Если мы применили функцию Коллатца к нечётному числу не более чем kw раз, то мы также применяем функцию Коллатца (интуитивно, чем меньше мы применяем функцию Коллатца к нечётному числу, тем меньше мы умножаем его на три и тем больше делим на два, таким образом, в среднем мы быстрее уменьшаем число, отображаемое на экране). В противном случае применяем случайную функцию.

Одна из причин, по которой это решение может работать плохо, заключается в том, что оно иногда может увеличить чётное число. С вероятностью 50% оно становится нечётным, и возможность его уменьшить вдвое упускается. Лучше сначала уменьшить число вдвое, а затем применить случайную функцию. Чтобы решить эту проблему, давайте объединим стратегию (w, k) со стратегиями $(1, k), \ldots, (w-1, k)$: то есть, если хотя бы для одного положительного целого числа $v \leqslant w$ в ближайшие v итерациях функции Коллатца мы увеличили число не более чем kw раз, то нам лучше применить Коллатца. Эта обновленная стратегия уже проходит все тесты для $w = 11$, $k = 0.37$. Чтобы получить пару хорошо работающих параметров, можно перебрать случайные пары параметров и проверить производительность каждой из них на многих случайных входных данных.

Еще одна стратегия заключается в том, чтобы попытаться построить (почти) оптимальное решение. Давайте выберем границу N и создадим массив чисел с плавающей точкой $\mathrm{dp}[1..N]$, обозначающий математическое ожидание оценки оптимальной стратегии, начинающейся с каждого числа. Инициализируем его оценкой, достигнутой только при применении функции Коллатца. Затем давайте пройдем по массиву s раз, каждый раз обновляя каждое значение $dp[i]$ минимумом оценки, достигнутой при применении Коллатца (и оптимальным продолжением), или средним значением по $a[3i + 1..6i]$, в зависимости от того, что меньше. Чтобы вычислить среднее значение по $a[3i + 1..6i]$, мы можем воспользоваться алгоритмом скользящей суммы (также известным как алгоритм оконной суммы). Этот алгоритм проходит тесты для $N = 10^7$, $s = 19$ (и, конечно, для многих других пар).

Задача M. Суммы двух

Идея: Иван Казменко
Разработка: Иван Казменко

Это задача про константные оптимизации.

Для начала реализуем то, что сказано в условии. Будем поддерживать массив из булевых значений v, в котором v_k указывает, есть ли в множестве V элемент k. Чтобы посчитать количество пар с суммой k, рассмотрим пары индексов (i, j), для которых $i \leqslant j$ и $i + j = k$. Придётся сделать порядка 10^{11} операций.

Чтобы операции были удобнее, будем поддерживать ещё один массив w, в котором элементы будут храниться в обратном порядке: $w_k = v_{m-k}$, где $m = 999\,982$. Теперь проверку $v_i \& v_{k-i}$ можно заменить на $v_i \& w_{m-k+i}$. Выгода в том, что теперь удобно хранить v_i и w_i в битах 64-битных целых чисел, и делать операцию & сразу с 64 битами, то есть в 64 раза быстрее.

Чтобы ускорить решение ещё в несколько раз, можно воспользоваться векторизацией операций, но уже с 64-битными числами: SSE или AVX. Аккуратной реализации — или ещё более аккуратного использования `bitset` из библиотеки — уже хватало, чтобы оптимизирующий компилятор сам сделал это (например, в C++ для этого нужно включить соответствующие `pragma`-инструкции).

Если этого не хватает, можно дополнительно оптимизировать код. Например, развернуть внутренний цикл вручную (loop unrolling). Или хранить 64 `bitset`-а для сдвигов на $0, 1, \ldots, 63$ бита. Самые быстрые решения жюри на нескольких языках работают в 4–5 раз быстрее, чем требуется в задаче.

Задача N. Разыскивается: второй носок

Идея: Иван Бочков
Разработка: Иван Бочков

Пусть $f(p, m)$ — ответ на задачу. О ней можно думать так: берется случайная перестановка

носков и считается математическое ожидание положения первого такого носка, чья пара уже была. Отметим, что:

1. $f(p,m) = \frac{f(p,m-1)\cdot(2p+m+1)}{(2p+m)}$. В самом деле, давайте выберем один непарный носок и фиксируем порядок остальных носков. Если остальные носки будут зафиксированы, мы сможем вставить фиксированный в каждый промежуток между остальными с равной вероятностью. То есть, переходя к математическому ожиданию, «длина» разрывов умножается на $\frac{(p+2m+1)}{(p+2m)}$.

2. $f(p,0) = f(p-1,1)$, если $p \geqslant 2$. Действительно, давайте обозначим последний носок в перестановке как 1. Тогда мы можем просто удалить этот носок и добавить оставшийся носок типа 1 к «непарным».

Из двух утверждений следует, что $f(p,m) = \frac{(2p+m+1)4^p}{(2p+1)C_{2p}^p}$, и после вычисления факториалов и их обратных ответ может быть вычислен за $O(1)$.