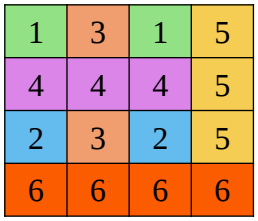
A. Неполный API

Автор задачи: Вова Мшанецкий
Количество сдавших команд: 7 из 31

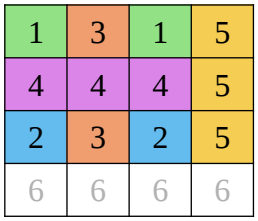
Рассмотрим решение задачи на следующем примере:



Предположим, что ответ существует, то есть существует такая последовательность из Q команд, пронумерованных от 1 до Q, что в результате их выполнения получится картинка, идентичная образцу. Тогда, в этой последовательности есть последняя команда, имеющая номер Q. Поскольку она — последняя, после её выполнения картинка стала идентична образцу и больше уже не менялась. Следовательно, если мы посмотрим на образец, мы должны увидеть в нём результат выполнения этой последней команды — линию (строку или столбец), полностью закрашенную одним цветом. Действительно, в нашем примере есть такая линия, это — строка 4, закрашенная цветом 6. Итак, мы нашли последнюю команду:

$$Q \rightarrow \texttt{hor 4 6}$$

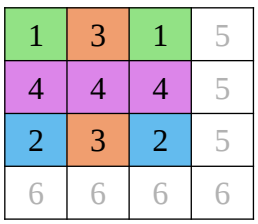
Перерисуем наш образец, отметив, что мы уже нашли команду, которая рисует эту линию:



Теперь попробуем найти команду $Q-1$. Поскольку мы по-прежнему исходим из предположения, что ответ существует, мы также должны иметь возможность найти результат выполнения этой команды в образце. Есть лишь одно различие: после команды $Q-1$ выполняется ещё команда Q, которая может перекрасить какие-то клетки, которые покрасила команда $Q-1$. Следовательно, мы должны найти в образце линию, в которой все клетки одного цвета, за исключением клеток, которые перекрашивает команда Q. Действительно, в нашем примере есть такая линия, это — столбец 4, покрашенный в цвет 5. Значит, мы нашли команду $Q-1$:

$$Q \rightarrow \texttt{hor 4 6}$$
$$Q-1 \rightarrow \texttt{ver 4 5}$$

Заметим, что клетка (4, 4) была покрашена сначала командой $Q-1$, а затем ещё раз командой Q. Следовательно, не важно, в какой цвет её покрасила команда $Q-1$, ведь команда Q её перекрасит. Снова перерисуем наш образец:



Аналогичными рассуждениями мы можем найти команду $Q-2$:

$$Q \rightarrow \texttt{hor 4 6}$$

$Q-1$ → ver 4 5
$Q-2$ → hor 2 4

1	3	1	5
4	4	4	5
2	3	2	5
6	6	6	6

Продолжим и найдём команду $Q-3$:

Q → hor 4 6
$Q-1$ → ver 4 5
$Q-2$ → hor 2 4
$Q-3$ → ver 2 3

1	3	1	5
4	4	4	5
2	3	2	5
6	6	6	6

Теперь мы получили ситуацию, что нам на роль команды $Q-4$ подходит две линии: строка 1 и строка 3. Можно доказать, что в таких ситуациях не имеет значения, какую линию мы возьмём первой. Для в нашем примере возьмём сначала строку 3:

Q → hor 4 6
$Q-1$ → ver 4 5
$Q-2$ → hor 2 4
$Q-3$ → ver 2 3
$Q-4$ → hor 3 2

1	3	1	5
4	4	4	5
2	3	2	5
6	6	6	6

Здесь тоже есть несколько вариантов команды $Q-5$: строка 1, столбец 1 и столбец 3. Не имеет значения, какую из этих линий мы выберем, но для краткости примера выберем строку 1. После этого у нас получится, что все клетки уже были закрашены какой-то командой. Добавлять ещё какие-то команды в ответ не имеет смысла, ведь все клетки, которые они покрасят, будут в последствии перекрашены командами, которые мы нашли раньше. Значит, команда $Q-5$ — это команда 1:

6 → hor 4 6
5 → ver 4 5
4 → hor 2 4
3 → ver 2 3
2 → hor 3 2
1 → hor 1 1

Таким образом, алгоритм решения следующий. Найдём линию, все клетки которой одного цвета, добавим соответствующую команду в ответ, и отметим все клетки этой линии. Затем найдём следующую линию, все клетки которой, за исключением уже отмеченных, одного цвета,

добавим соответствующую команду в ответ, и снова отметим все клетки этой линии. Так будем продолжать в цикле, пока либо не отметим все клетки, либо не не сможем найти ни одной такой линии. В первом случае построение ответа завершено, и нам остаётся только вывести команды в обратном порядке (ведь мы искали их от Q к 1, а вывести в ответ нужно от 1 к Q). Во втором случае мы обнаружили, что ответа нет.

При данных ограничениях поиск следующей линии можно реализовать перебором всех столбцов, а затем всех строк, и просмотром всех клеток этого столбца или строки.

B. Носки

Автор задачи: Виктория Рувинская
Количество сдавших команд: 31 из 31

В задаче спрашивалось, сколько носков нужно вынуть не глядя из мешка, в котором находится N разных пар носков, чтобы гарантировано вынуть одну пару. В худшем случае, доставая носки, мы каждый раз будем доставать носок из новой пары, пока не вынем по одному носку из каждой пары. После этого мы должны вынуть ещё один носок и он уже точно подойдёт к одному из тех, что мы достали раньше. Таким образом, ответ — $N+1$.

При реализации следовало учесть, что, учитывая ограничения на N, результат вычисления мог быть вне диапазона значений целых знаковых 32-битный чисел (`int`). Поэтому, для вычислений следовало использовать либо 32-битные без знака, либо 64-битные целые числа (`unsigned` или `long long`).

C. Нечетный набор

Автор задачи: Денис Остапенко
Количество сдавших команд: 29 из 31

В задаче нужно было представить заданное число N в виде суммы минимального количества нечётных чисел. Возможны следующие варианты:

- Если $N=0$, то N можно представить как 0 чисел (сумма нуля чисел равна нулю)
- Если $N \, mod \, 2=1$, то N можно представить как одно число — само себя. Многие участники в данном случае представляли N как сумму трёх чисел, что неправильно.
- Иначе N можно представить как два числа, например, 1 и $N-1$. Возможны и другие представления (3 и $N-3$, 5 и $N-5$ и так далее). Все такие представления тоже являются правильными ответами.

При реализации нужно было учесть, что согласно ограничениям число N может не помещаться в 32-битное целое. Нужно было использовать `long long`.

D. Побудова рядка

Автор задачи: Иван Фекете
Количество сдавших команд: 17 из 31

В первую очередь, важно понимать, что каждый символ в строке сам по себе является палиндромом. Например, в строке «abcd» есть четыре палиндрома — каждый из символов. На этом факте основывается очевидное но неправильное решение: выводить строку вида «abcabcabc...» (то есть, выводить символы abc по кругу) длиной N символов. В таких строках действительно будет требуемое количество подстрок-палиндромов, но поскольку согласно ограничениям $N \le 10^9$, длина построенных таким образом строк может превышать заданное в условии ограничение на 10^5 символов.

Поскольку согласно условию в ответ можно вывести любую строку, удовлетворяющую требованиям по длине и количеству подстрок-палиндромов, для каждого N существует множество правильных ответов. Следовательно, для данной задачи существует большое количество разных правильных решений. Далее рассмотрено одно из самых простых решений.

Для начала рассмотрим, как изменяется количество палиндромов в строке при последовательном добавлении к ней одного и того же символа:

- a — 1 палиндром
- aa — 3 палиндрома (первое «a», второе «a», «aa»)
- aaa — 6 палиндромов (первое «a», второе «a», третье «a», первое «aa», второе «aa», «aaa»)
- aaaa — 10 палиндромов (первое «a», второе «a», третье «a», четвёртое «a», первое «aa», второе «aa», третье «aa», первое «aaa», второе «aaa», «aaaa»)
- aaaaa — 15 палиндромов

Видно, что при каждом добавлении новой буквы в строке остаются все старые подстроки-палиндромы, и появляется такое количество новых, как новая длина строки. Действительно:

- $1=1$
- $1+2=3$
- $3+3=6$
- $6+4=10$
- $10+5=15$

Это объясняется тем, что в строку фактически добавляются подстроки-палиндромы, заканчивающиеся в добавленном символе. При этом начало этих новых подстрок-палиндромов может быть в каждом из символов новой строки. Таким образом, в строке из M одинаковых символов будет $1+2+3+\ldots+(M-1)+M$ подстрок-палиндромов.

Будем формировать строку-ответ по следующему шаблону: «aaa...aaabbb...bbbccc...cccaaa...aaabbb...bbbccc...ccc...», то есть, сначала M_1 символов «a», затем — M_2 символов «b», затем — M_3 символов «c», затем — M_4 символов «a», затем — M_5 символов «b», и так далее. Делать это будем следующим образом.

Начнём с пустой строки. Будем добавлять к ней в конец символ «a», пока не наступит момент, когда добавление ещё одного такого символа привело бы к тому, что общее количество подстрок-палиндромов превысило бы N. Тогда начнём добавлять в конец строки символ «b», и будем снова добавлять его, пока очередное добавление не привело бы к избытку общего количества подстрок-палиндромов. Затем перейдём на добавление символов «c», и снова будем делать то же самое. После этого снова перейдём на символы «a», затем на «b», и так по кругу. Остановим процесс, как только в строке в сумме наберётся ровно N подстрок-палиндромов.

E. Эмгыр и взлом сети

Автор задачи: Максим Молчанов
Количество сдавших команд: 3 из 31

Оптимальная стратегия в данной задаче следующая. Если мы дойдём так, чтобы нас не заметили, до некоторой вершины v, из которой до вершины N можно дойти не больше, чем за D времени, то дальше мы гарантированно успеем достичь нашей цели (то есть, взломать вершину N). Следовательно, нам следует идти к такой вершине v, до которой вероятность дойти незамеченными будет максимальной. Эта вероятность и будет ответом задачи.

Для начала следует для каждой вершины v найти время F_v, необходимое на завершение взлома вершины N, если двигаться от v, включая время взлома самой v. Поскольку граф ориентированный и ациклический, это можно сделать, например, рекурсивным обходом в глубину. После обхода всех дочерних вершин некоторой вершины v, следует выбрать ту дочернюю вершину u, в которой время F_u получилось минимальным, и прибавить время на взлом вершины t_v В результате получится искомое время $F_v = F_u + t_v$. Для вершины N следует принять $F_N = t_N$.

Далее есть несколько способов решения. Первый способ — для каждой вершины v посчитать максимальную вероятность p_{1v} незамеченным дойти до этой вершины и начать её взлом. Это можно сделать способом, аналогичным подсчёту F_v, только обходя граф с развёрнутыми в обратном направлении дугами начиная из вершины N. Следует принять $p_{11} = 1$, а для остальных вершин будет $p_{1v} = (1 - p_v) \cdot max(p_{1u})$ по всем u, для которых в исходном графе есть дуга $v \to u$. После этого ответом задачи будет $max(p_{1v})$ среди всех v, для которых $F_v - t_v \le D$. Заметим, что поскольку p_{1v} включает вероятность, что нас не засекут в момент начала взлома v, а F_v включает время взлома v, при сравнении нам нужно вычесть t_v, иначе мы в некотором смысле учтём взлом v дважды.

Второй способ — посчитать для каждой вершины v вероятность p_{2v} успешного достижения цели. Ответом задачи будет p_{21}. Для всех вершин, для которых $F_v \le D$ эта вероятность будет равна 1. Для остальных вершин она будет равна $p_{2v} = (1 - p_v) \cdot max(p_{2u})$ по всем u, для которых в графе есть дуга $v \to u$. Подсчитать эти вероятности можно во время того же рекурсивного обхода в глубину, который подсчитывает значения F_v.

При реализации следует обратить внимание на случаи, когда в графе есть тупиковые пути, несколько компонент связанности, либо вершина N недостижима из 1. Кроме того, при реализации первого способа в качестве небольшой оптимизации можно вместо двух обходов выполнить топологическую сортировку графа и затем обрабатывать вершины в получившемся порядке. При подсчёте F_v следует обрабатывать вершины в порядке от дочерних к родительским, а при подсчёте p_{1v} — от родительских к дочерним. Про топологическую сортировку можно подробнее почитать, например, здесь: http://e-maxx.ru/algo/topological_sort .

F. Сигнализация

Автор задачи: Вова Мшанецкий
Количество сдавших команд: 3 из 31

Для начала рассмотрим вариант задачи без датчиков. Это — классическая задача о нахождении кратчайшего пути к выходу в лабиринте. Она решается с помощью обхода в ширину, который в данном случае называют волновым алгоритмом. Описание этого алгоритма выходит за рамки данного разбора, но его можно найти в интернете. Например, есть статья в Википедии: https://ru.wikipedia.org/wiki/Алгоритм_Ли .

Вернёмся к исходной задаче. Очевидно, что в клетку с датчиком можно заходить только в том случае, если мы успеем добраться от неё до целевой клетки не больше, чем за T_i шагов (где T_i — количество времени на таймере датчика в данной клетке). В противном случае её можно считать непроходимой, как и клетки с #. Сперва может прийти идея запустить поиск в ширину из каждой клетки с датчиком и проверить получившуюся длину пути до целевой клетки. Но это будет работать слишком долго ($O\left((HW)^2\right)$ — порядка $2{,}6 \cdot 10^{10}$ операций). Тогда можно заметить, что вместо того, чтобы запускать поиск отдельно из каждой клетки с датчиком, можно запустить его один раз из целевой клетки и сразу определить расстояния до всех клеток с датчиками. Это будет работать уже за оптимальное время. Однако, оба варианта будут работать неправильно, так как длина пути между целевой клеткой и клеткой с датчиком может зависеть от того, можно ли заходить в другие клетки с датчиками, которые встретятся на пути. Для того, чтобы учесть это, нужно немного модифицировать алгоритм обхода лабиринта.

Запустим поиск в ширину из целевой клетки. Рассматривая соседа некоторой клетки и определяя, следует ли переходить в этого соседа (то есть, обновлять расстояние до него и добавлять его в очередь), следует проверять следующие условия:

- клетка-сосед проходима
- клетка-сосед ещё не посещалась
- если в клетке-соседе есть датчик, то расстояние, которое мы собираемся записать в эту клетку, не превышает T_i этого датчика

Последнее условие обеспечит то, что мы не будем входить в клетки с датчиками, от которых мы не успеем добраться до целевой клетки. Поскольку при обходе в ширину мы будем рассматривать клетки в порядке возрастания расстояний от целевой клетки до них, мы всегда сначала рассмотрим клетки с датчиками, которые могут встретиться на пути от других, более дальних, датчиков до целевой клетки, а уже потом — эти более дальние датчики. Таким образом мы найдём правильные расстояния с учётом проходимости датчиков, которые встречаются на пути от других датчиков.

В процессе такого обхода мы можем отметить клетки с датчиками, в которые нельзя заходить, и потом запустить новый обход из стартовой клетки в целевую. Или мы можем заметить, что получившееся в результате такого обхода расстояние от целевой клетки до стартовой уже будет ответом задачи.

При реализации следует обратить внимание на следующие случаи:

- в стартовой и/или целевой клетке есть датчик
- стартовая и целевая клетка совпадают
- стартовая и целевая клетка совпадают, и в этой клетке есть датчик

G. Чеканной монетой

Автор задачи: Денис Остапенко
Количество сдавших команд: 23 из 31

Очевидно, что для того, чтобы получить N копеек минимальным количеством монет, нужно взять как можно больше монет номиналом 29 копеек. Возможны разные решения, но одно из самых простых — перебрать количество монет номиналом 29 копеек A от $\lfloor N \div 29 \rfloor$ до 0. Как только для некоторого A у нас получилось $(N - 29 \cdot A) mod 10 = 0$, мы нашли ответ и $B = (N - 29 \cdot A) \div 10$. Если мы дошли до нуля (и проверили случай $A = 0$) и так и не нашли ответ, то получить сумму данными номиналами невозможно. Может показаться, что такой перебор может работать очень долго, но это не так: несложно доказать, что он завершится не более, чем за 9 итераций.

Альтернативный способ — перебрать количество монет номиналом 10 копеек от 0 до 28. Очевидно, что 29 монет по 10 копеек брать не имеет смысла, так как их можно было бы заменить 10 монетами по 29 копеек.

При реализации следовало учесть, что согласно ограничениям N, A и B могут не помещаться в 32-битное целое (`int`). Следовало использовать 64-битные числа (`long long`).

H. Илмюзия прогреста

Автор задачи: Дима Садовый
Количество сдавших команд: 4 из 31

Для начала рассмотрим эту задачу без требования вывести минимальную лексикографически строку. Эта задача решается динамическим программированием (описание принципа динамического программирования выходит за рамки данного разбора). Пусть $f(n,c)$ — минимальная стоимость (в штрафных балах), за которую можно изменить префикс строки S длиной n согласно правилам так, чтобы последний символ получился $S_n=c$ ($'a'\le c\le 'z'$). Тогда мы можем вычислить $f(n,c)$, перебрав предыдущий символ p ($S_{n-1}=p$) и выбрав вариант, в котором стоимость получится минимальной:

$$f(n,c)=cost(S_n,c)+\min_{p='a',p\neq c}^{'z'}(f(n-1,p))$$

Здесь $cost(a,b)$ — стоимость получения из символа a символа b, которая рассчитывается как:

$$cost(a,b)=\begin{cases}b-a, a\le b\\ 26-(a-b), a>b\end{cases}$$

Минимальной стоимостью изменения всей строки будет:

$$\min_{c='a'}^{'z'}(f(N,c))$$

Чтобы восстановить ответ, нужно при вычислении каждого $f(n,c)$ сохранять p, при котором был получен минимум, и потом выполнить обратный проход.

Теперь рассмотрим требование выбора лексикографически минимальной строки из строк с одинаковой минимально возможной стоимостью. Сначала может показаться, что достаточно модифицировать выбор минимума таким образом, чтобы при равной стоимости выбирался минимальный p. Но это — не совсем правильно: при такой модификации, поскольку последние символы строки обрабатываются позже первых, они будут иметь больший приоритет по минимальности символов. Таким образом, мы получим лексикографически минимальную строку, если читать её с конца. Для того, чтобы исправить это, изменим нашу динамику так, чтобы она работала не наращиванием длины префикса, а наращиванием длины суффикса.

То есть, $f_s(n,c)$ — минимальная стоимость за которую можно изменить суффикс строки S длиной n так, чтобы первый символ получился $S_n=c$. Рассчитываться она будет аналогично:

$$f_s(n,c)=cost(S_n,c)+\min_{p='a',p\neq c}^{'z'}(f_s(n+1,p))$$

При этом нужно из нескольких равных минимумов выбирать тот, который при меньшем значении p. Минимальной стоимостью изменения всей строки будет:

$$\min_{c='a'}^{'z'}(f_s(1,c))$$

Восстановить ответ также можно с помощью запоминания p и обратного прохода.

I. Состав команд, или состязание равных

Автор задачи: Леонид Беркович
Количество сдавших команд: 6 из 31

Авторское решение (meet-in-the-middle)

Пусть n — число участников турнира, sum — сумма рейтингов всех участников, m = n / 2 — число участников в каждой команде.

Если sum нечетное число, ответом будет -1.

Если sum четно, разделим массив рейтингов на 2 равные части.

Переберем в двумерном массиве dp[M][M] все возможные сочетания из m по k ($1 \le k \le m$). Здесь M и N — максимальные значения m и n.

Сначала заполним нулевую строку массива. Далее в каждом элементе массива ([j][i]) строится вектор сочетаний из i по j, полученный из вычисленных ранее значений для [j][i-1] и [j-1][i-1].

То есть, если мы строим, например, вектор сочетаний из 4 по 2, берем сочетания из 3 по 2 ({0, 1}, {0, 2}, {1, 2}) и добавляем в наш вектор, а затем добавляем сочетания из 3 по 1 вместе с новым элементом 3 ({0, 3}, {1, 3}, {2, 3).

Каждое сочетание сохраняется как bitset<M>.

В том же двумерном цикле получаем и значения сумм рейтингов для всех сочетаний, по аналогии используя ранее вычисленные суммы.

Суммы рейтингов сохраняются в векторах sums1[M][M] и sums2[M][M] (отдельно для каждой половины множества участников).

Сочетание как bitset для текущего значения суммы sums1[][] заносится в массив bitset<M> nums1[M * R + 1][M + 1] и, соответственно, bitset для последнего значения sums2[][] — в массив nums2[M * R + 1][M + 1]. Здесь R — максимально возможное значение рейтинга. То есть, если в каком-то сочетании из m по k для первой половинки мы получили сумму sum1, то присваиваем элементу nums1[sum1][k] bitset этого сочетания.

Полученные значения для второй половинки будут в массиве nums2.

Заметим, что максимальное число сочетаний в нашем случае 2^M.

Теперь пройдем в двумерном цикле по массиву nums1. Если для текущих значений sum1 и k есть ненулевой bitset в nums2[sum / 2 – sum1][m - k], то нам остается соединить два bitset-а nums1[sum1][k] и nums2[s / 2 – sum1][m - k] и таким образом составить первую команду. Состав второй команды можно получить так: command2 = command1.flip(); ну а если прошли весь цикл без выхода, то ответ -1.

Альтернативное решение (динамическое программирование)

Пусть $f(n, \Delta c, \Delta s)$ — такое разбиение первых n участников на две команды, при котором разность количества участников в командах равна Δc, а разность сумм рейтингов — Δs. Разбиение будем хранить, например, как двоичную маску, или как массив номеров участников, включённых в первую команду.

Чтобы найти разбиение $f(n, \Delta c, \Delta s)$ следует попробовать дополнить разбиения первых $n-1$ участников, определив участника n в первую или вторую команду:

- Если разбиение $f(n-1, \Delta c-1, \Delta s-R_n)$ возможно, то $f(n, \Delta c, \Delta s) = f(n-1, \Delta c-1, \Delta s-R_n)+\{n \to 1\}$
- Если разбиение $f(n-1, \Delta c+1, \Delta s+R_n)$ возможно, то $f(n, \Delta c, \Delta s) = f(n-1, \Delta c+1, \Delta s+R_n)+\{n \to 2\}$
- Иначе разбиение $f(n, \Delta c, \Delta s)$ невозможно

Здесь запись $A+\{n \to b\}$ обозначает, что мы берём разбиение A и дополняем его, определив участника n в команду b.

Начальные условия — $f(0,0,0)=\{\}$ (пустое разбиение), $f(0,\Delta c,\Delta s)=\text{impossible}$ для всех остальных Δc и Δs. Ответом задачи будет разбиение $f(N,0,0)$.

Реализовать динамику можно либо в виде рекурсивной функции с запоминанием, либо заполнением трёхмерного массива от меньших n к большим n. Следует учитывать, что значения Δc и Δs могут быть отрицательными, поэтому нужно сдвинуть индекс нуля в массиве.

Альтернативное решение (meet-in-the-middle)

Переберём все возможные разбиения первой половины участников ($1 \dots N/2$), посчитаем для каждого такого разбиения Δc и Δs и запишем в массив `arr[` Δc `][` Δs `] = mask`, где `mask` — двоичная маска текущего разбиения. Потом переберём все возможные разбиения второй половины участников ($N/2+1 \dots N$), тоже посчитаем для каждого Δc и Δs и проверим, записано ли что-то в `arr` в ячейке `arr[` $-\Delta c$ `][` $-\Delta s$ `]`. Если да, то мы нашли ответ, осталось только объединить маску `arr[` $-\Delta c$ `][` $-\Delta s$ `]` и текущую маску разбиения второй половины участников. Если мы не нашли ответа, перебрав все возможные разбиения второй половины участников, то ответа нет.

При реализации, разумеется, следует при индексации в массив `arr` сдвигать индексы нулей. Например, использовать `arr[` $N/2 \pm \Delta c$ `][` $S \pm \Delta s$ `]`, где $S=R_1+R_2+\dots+R_{N-1}+R_N$. Изначально можно заполнить массив `arr` числами -1, и проверку «записано ли что-то» реализовать как проверку на неравенство -1.

J. Гиперсемена

Автор задачи: Денис Остапенко
Количество сдавших команд: 9 из 31

Задача решается динамическим программированием (описание принципа динамического программирования выходит за рамки данного разбора). Пусть $f(n,m)$ равно `true`, если возможно распределить n семян таким образом, чтобы остаток от деления урожая на 47 был равен m. Тогда мы можем определить значение $f(n,m)$, перебрав количество семян k в последнем горшке от 1 до n. Для каждого k посчитаем $k^k \bmod 47$ и определим, какой остаток должен быть в предыдущих горшках, чтобы общий остаток получился m: $m_{prev}=(47+m-(k^k \bmod 47)) \bmod 47$. После этого проверим, что $f(n-k,m_{prev})=\text{true}$. Если это так, то $f(n,m)$ тоже равно `true`. Иначе — продолжаем поиск. Более формально:

$$\begin{gathered}
f(n,m)=f(n-1,(47+m-(1^1 \bmod 47)) \bmod 47) \\
\vee f(n-2,(47+m-(2^2 \bmod 47)) \bmod 47) \\
\vee f(n-3,(47+m-(3^3 \bmod 47)) \bmod 47) \\
\dots \\
\vee f(3,(47+m-((n-3)^{n-3} \bmod 47)) \bmod 47) \\
\vee f(2,(47+m-((n-2)^{n-2} \bmod 47)) \bmod 47) \\
\vee f(1,(47+m-((n-1)^{n-1} \bmod 47)) \bmod 47) \\
\vee f(0,(47+m-(n^n \bmod 47)) \bmod 47)
\end{gathered}$$

Начальные условия: $f(0,0)=\text{true}$, $f(0,i)=\text{false}$ для всех $1 \le i \le 46$. Если $f(N,0)=\text{true}$, то искомое распределение существует, иначе — нет. Чтобы восстановить искомое распределение, нужно при каждом вычислении $f(n,m)$ сохранять значение k, при котором было найдено $f(n-k,m_{prev})=\text{true}$. Тогда после вычисления $f(N,0)$ можно выполнить обратный проход.

Реализовать динамику можно заполнением массива от меньших n к большим n, либо рекурсивной функцией. При заполнении массива на практике удобнее реализовывать динамику в наоборот: если некоторое $f(n,m)=\text{true}$, то для всех k от 1 до $N-n$ значение $f(n+k,(m+(k^k \bmod 47)) \bmod 47)$ следует тоже установить `true`. При вычислении $x^x \bmod 47$ следует использовать свойство модуля $(a \cdot b) \bmod m=((a \bmod m)(b \bmod m)) \bmod m$. Для повышения эффективности можно заранее подсчитать $x^x \bmod 47$ для всех k от 1 до 74 и сохранить в массив.

Кроме этого, учитывая малый размер множества возможных входных данных, данную задачу можно было решить с помощью precalc. При должном упорстве, наверное, можно было даже найти ответы для всех N от 1 до 74 вручную, используя только калькулятор и листок бумаги. После этого можно было отправить программу, которая просто выводит эти ответы.

K. Кубики

Автор задачи: Иван Фекете
Количество сдавших команд: 25 из 31

Получить башенки требуемых высот ($b_1, b_2, \ldots, b_n$) можно тогда и только тогда, когда общее количество кубиков в этих башенках ($b_1+b_2+\ldots+b_n$) равно общему количеству кубиков в исходных башенках ($a_1+a_2+\ldots+a_n$).

Если суммы совпали, то чтобы определить, сколько кубиков нужно переложить, нужно для каждого i от 1 до n сравнить a_i и b_i. Если $a_i=b_i$, то ничего перекладывать не нужно. Если $a_i>b_i$, то нужно взять a_i-b_i кубиков сверху башенки и переложить их на какие-то другие башенки, где $a_j<b_j$. Если $a_i<b_i$, то ничего делать не надо, поскольку мы достроим эти башенки, обрабатывая случаи $a_i<b_i$. Поскольку мы уже проверили, что $a_1+a_2+\ldots+a_n=b_1+b_2+\ldots+b_n$, то мы точно знаем, что для каждого кубика, который мы сняли в случаях $a_i>b_i$, найдётся место, куда его положить, в одном из случаев $a_i<b_i$. Таким образом, ответ можно посчитать как:

$$\sum_{i=1}^{n} max(0, a_i-b_i)$$

При реализации следовало учесть, что при данных ограничениях на n, a_i и b_i суммы $a_1+a_2+\ldots+a_n$ и $b_1+b_2+\ldots+b_n$, а также ответ могли выйти за диапазон 32-битовых целых чисел (`int`). Поэтому следовало использовать 64-битные числа (`long long`).