Задача A. Долгая игра

Обозначим ответ для n за $A(n)$. Положим $A(0) = 0$. Для $n > 0$ выполнено $A(n) = 1 + \sum\limits_{k=0}^{n} p(n,k)A(n-k)$, где $p(n,k)$ — вероятность, что у случайной перестановки из n элементов ровно k неподвижных точек. $p(n,0) < 1$, поэтому из этого уравнения можно однозначно определить $A(n)$, если мы знаем $A(m)$ для всех $0 \leqslant m < n$.

Если мы научимся считать $p(n,k)$, то это даст нам решение за $O(n^2)$ с помощью динамического программирования. Но вместо этого давайте попробуем угадать ответ.

Утверждается, что $A(n) = n$. Если мы проверим, что это удовлетворяет уравнению для всех n, то это и есть решение, так как решение единственно. Нам нужно проверить, что $n = 1 + \sum\limits_{k=0}^{n} p(n,k)(n-k)$. Преобразуем правую часть: $1 + \sum\limits_{k=0}^{n} p(n,k)(n-k) = 1 + \sum\limits_{k=0}^{n} p(n,k)n - \sum\limits_{k=0}^{n} p(n,k)k$.

$\sum\limits_{k=0}^{n} p(n,k) = 1$, так как это вероятность полного события. Таким образом, нам надо проверить, что $\sum\limits_{k=0}^{n} p(n,k)k = 1$. Слева записано математическое ожидание количества неподвижных точек у случайной перестановки. По линейности матожидания его можно вычислить по другому: это сумма вероятностей, что данная позиция будет неподвижной точкой. Но для каждой позиции такая вероятность равна $\frac{1}{n}$, так как каждый элемент имеет одинаковую вероятность оказаться на этой позиции.

Таким образом, уравнение выполнено для всех n, а значит $A(n) = n$.

Задача B. Липецкое метро

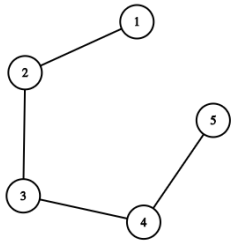
В задаче просят найти гамильтонов путь. Вообще говоря, задача поиска (и даже проверки существования) гамильтонова пути является NP-трудной, и на данный момент неизвестно быстрых алгоритмов решения этой задачи. Но в нашей задаче граф имеет специальный вид, поэтому задачу можно решить за линейное время.

Очевидно, чтобы в графе был гамильтонов путь, он должен быть связным. Проверим граф на связность. В нашем графе не более n рёбер, поэтому это либо дерево, либо дерево + ещё одно ребро. Второй тип графов называется унициклическим, потому что в нём один цикл. Такой граф можно себе представить как цикл, с вершин которого свисают деревья.

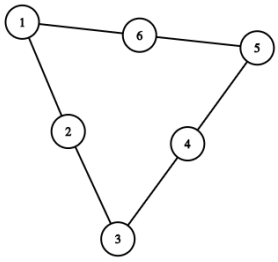
Пусть в нашем графе есть гамильтонов путь. Пронумеруем вершины в порядке гамильтонова пути. Граф обязан иметь все рёбра между соседними вершинами в гамильтоновом пути, это уже $(n-1)$ ребро. Возможно, есть ещё одно ребро. Таким образом, все возможные графы, которые имеют гамильтонов путь, принадлежат одному из четырёх классов:

1. бамбук;
2. цикл;
3. цикл, из одной из вершин которого выходит путь;
4. цикл, из двух **соседних** вершин которого выходит путь.

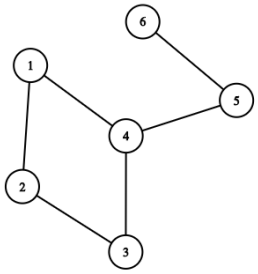
По одному представителю каждого класса вы можете видеть на картинке:



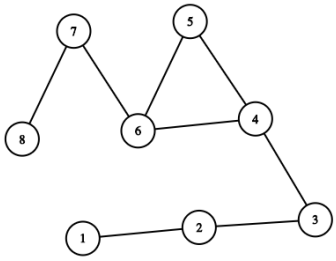
Класс 1



Класс 2



Класс 3



Класс 4

Одним из решений является проверить связность, а затем аккуратно проверить принадлежность данного графа к одному из этих четырёх классов.

Но есть и более изящные решения, не требующие разбора случаев (по крайней мере в коде) и, таким образом, более устойчивые к ошибкам при написании. Все такие решения опираются на какие-то упрощения, которые работают в случае, если в графе есть гамильтонов путь, а если его нету, то мы его и не найдём. Вот некоторые из таких решений:

1. Проверим связность, затем попытаемся правильно удалить лишнее ребро, если у нас есть цикл, затем проверим, что граф является бамбуком. Можно заметить, что лишнее ребро всегда имеет максимальную сумму степеней концов.

2. Встанем в вершину минимальной степени, затем будем жадно идти в ранее не посещённую вершину минимальной степени.

3. Напишем полный перебор (DFS, который стирает пометку при выходе из вершины), причём начнём перебирать стартовые вершины в порядке неубывания степени. Если в графе есть гамильтонов путь, то мы найдём его за линейное количество операций. Если алгоритм слишком долго ничего не находит, выведем `NO`.

Задача C. Начинающий маг

В задаче дано корневое дерево, нужно выбрать две вершины с одинаковой высотой так, чтобы в объединении путей из них до корня было не более K рёбер, при этом их высота должна быть как можно больше.

Решение 1.

Зафиксируем w — LCA двух вершин u v, которые мы хотим выбрать. Какие условия должны быть выполнены? $h[v] = h[u]$, $2h[v] - h[w] \leqslant K$, u и v находятся в разных поддеревьях w. Заметим, что если мы можем взять две вершины в разных поддеревьях w на одинаковой высоте $H > h[w]+1$, то мы можем взять и две вершины в разных поддеревьях w на одинаковой высоте $(H-1)$ — это будут

родители старых вершин. Таким образом, достаточно найти пару вершин в разных поддеревьях w с самой большой высотой, — после этого выбрать пару вершин в разных поддеревьях w с самой большой высотой так, чтобы выполнялось $2h[v] - h[w] \leqslant K$, не составит большого труда.

Чтобы найти пару вершин в разных поддеревьях w с самой большой высотой будем, эхем, обрабатывать w в порядке выхода из DFS и поддерживать две самых глубоких вершины в разных поддеревьях w. Фактически, это поддержание двух максимумов.

Нужно быть аккуратным при обновлении ответа: либо нужно насчитать двоичные подъёмы и вычислять предка на заданной высоте за $O(\log N)$, либо вычислить вершины для вывода только один раз, перед самим выводом. При аккуратной реализации двоичные подъёмы не потребуются, и всё решение можно реализовать со сложностью $O(N)$.

Это решение не требует никаких алгоритмов работы с деревьями и просто пишется, но сложнее придумать.

Решение 2.

Сделаем DFS и выпишем для каждой высоты список вершин на данной высоте в порядке обхода DFS. Можно показать, что всегда существует решение, которое выбирает две вершины на одной высоте, которые являются соседними в этом порядке. Переберём все пары соседних вершин для каждой высоты — таких пар $O(N)$, — для каждой вычислим LCA, посчитаем количество использованных рёбер и обновим ответ. Сложность $O(N \log N)$ (может быть лучше при использовании LCA в оффлайне).

Это решение проще, если вы хорошо знакомы с алгоритмами LCA и свойствами эйлерова обхода (обхода в порядке DFS).

Задача D. Юные следопыты

Отсортируем всех следопытов по неубыванию неопытности. Пусть мы сформировали какую-то группу, как проверить, что эта группа валидная? Неопытности всех следопытов в группе должны быть не больше размера группы. Но мы отсортировали всех следопытов, поэтому самая большая неопытность у последнего следопыта из группы. Следовательно, чтобы проверить группу на валидность необходимо и достаточно проверить, что неопытность последнего следопыта в группе не превышает размер группы.

Можно заметить, что мы вообще не обращаем внимание на всех следопытов, кроме последнего, нам важно только их количество. Фактически, распределение на группы можно организовать следующим образом: сначала выберем следопытов, которые будут последними в своих группах, а потом выдадим им нужное количество других следопытов. Никогда нет смысла давать больше следопытов, чем нужно, если что, лишних можно оставить в лагере.

Как же оптимально выбрать последних следопытов? Мы хотим сформировать побольше групп, поэтому сами группы должны быть поменьше... Возникает идея следующего жадного алгоритма: будем каждый раз жадно брать самого левого (а значит с самым маленьким требуемым размером группы) следопыта, слева от которого достаточно следопытов, чтобы выдать ему валидную группу. Идея в том, что мы тратим минимальное количество следопытов, и оставляем максимальное количество потенциальных последних следопытов. Докажем эту жадность строго:

Решение задаётся позициями последних следопытов $1 \leqslant p_1 < p_2 < \ldots < p_k \leqslant n$. Заметим, что решение валидно тогда и только тогда, когда $e_{p_1} + e_{p_2} + \ldots + e_{p_i} \leqslant p_i$ для всех $1 \leqslant i \leqslant k$ (нам всегда хватает следопытов, чтобы сформировать первые i групп).

Пусть $1 \leqslant p_1 < p_2 < \ldots < p_k \leqslant n$ — жадное решение, $1 \leqslant q_1 < q_2 < \ldots < q_m \leqslant n$ — оптимальное решение с максимальным общим префиксом с жадным решением. Пусть t — позиция первого отличия. $t \leqslant k$, так как иначе жадное решение не смогло добавить $(k+1)$-ю группу, хотя это было возможно. $p_t < q_t$, так как иначе жадное решение взяло бы следопыта q_t вместо следопыта p_t. Так как следопыты отсортированы, из этого следует $e_{p_t} \leqslant e_{q_t}$. Но тогда легко понять, что $1 \leqslant q_1 < q_2 < \ldots < q_{t-1} < p_t < q_{t+1} < \ldots < q_m \leqslant n$ — это валидное оптимальное решение, и у него общий префикс с жадным строго больше, что противоречит выбору оптимального решения.

Чтобы реализовать это решение, достаточно отсортировать следопытов по неубыванию неопытности, потом идти слева направо и поддерживать количество неиспользованных следопытов. Как только мы видимо возможность создать новую группу, мы её создаём.

Задача E. M — многомерность

Пересечение двух параллелепипедов — это либо тоже параллелепипед, либо пустое множество. Чтобы его найти, нужно независимо пересечь отрезки по каждой из M координат. Таким образом, мы можем пересечь два параллелепипеда за $O(M)$. Чтобы пересечь K параллелепипедов, нам потребуется $O(KM)$ времени.

Чтобы вычислить количество целочисленных точек внутри параллелепипеда, нужно посчитать количество таких точек на отрезке, соответствующем каждой из координат, и перемножить. Это можно сделать за $O(M)$.

Зафиксируем один параллелепипед A, которые мы не хотим включать в пересечение. Остальные параллелепипеды пересечём, назовём результат I. Как теперь посчитать количество точек в I, которые при этом не лежат в A? Очень просто: это количество точек в I минус количество точек в $I \cap A$.

Если мы будем перебирать A, то общее время работы составит $O(N^2M)$, что плохо. Но мы можем воспользоваться тем, что мы постоянно пересекаем одни и те же параллелепипеды. Посчитаем пересечения для всех префиксов и всех суффиксов параллелепипедов за $O(NM)$. Тогда найти пересечение всех, кроме одного, мы сможем за $O(M)$: достаточно пересечь префикс до исключённого, и суффикс до него же.

Общее время работы составит $O(NM)$.

Задача F. Игра с массивом

При $S \geqslant 2N$ Петя выигрывает: возьмём массив $[2, 2, \ldots, 2, S - 2(N-1)]$ и $K = 1$. Все элементы строго больше 1, поэтому отрезка с суммой 1 или $S - 1$ не существует.

Докажем, что при $S < 2N$ Петя проигрывает. Пусть это не так, и существует соответствующий массив и число $K > 0$ ($K = 0$, очевидно, не подходит). Заметим, что наличие отрезка с суммой K или $S - K$ равносильно наличию отрезка с суммой K в циклическом массиве. Посчитаем префиксные суммы в нашем массиве, и если есть префикс с суммой M, пометим все числа вида $M + TS$ для целых $T \geqslant 0$. Легко понять, что числа X и $X + K$ не могут быть одновременно помечены: это бы значило, что в циклическом массиве есть отрезок с суммой K. Рассмотрим полуинтервал $[0; 2KS)$. Очевидно, на нём помечено ровно $2KN$ чисел. С другой стороны, все числа этого полуинтервала можно разбить на KS пар с разностью K: $(0, K), (1, K+1), \ldots, (K-1, 2K-1), (2K, 3K), (2K+1, 3K+1), \ldots (2KS-K-1, 2KS-1)$. В каждой паре помечено не более одного числа, следовательно, всего помечено не более KS чисел. Отсюда $2KN \leqslant KS$, то есть $2N \leqslant S$. Противоречие.

Задача G. Последовательность с цифрами

Попробуем вычислить последовательность при фиксированном $a_1 = 1$: $1, 2, 6, 42, 50, 50, 50, \ldots$

Нам повезло, и минимальная цифра стала равной 0, после чего элемент перестал изменяться, так как мы прибавляем к нему 0.

На самом деле это не везение, и такое будет происходить всегда. Заметим, что мы каждый раз прибавляем не более $9 \cdot 9 = 81$, поэтому разность между двумя соседними числами в последовательности не превышает 81. Пусть мы никогда не получим минимальную цифру, равную 0. Тогда последовательность будет бесконечно возрастать. Рассмотрим $X = 1000(\lfloor \frac{a_1}{1000} \rfloor + 1)$. На отрезке $[X; X + 99]$ все числа имеют 0 в разряде сотен, поэтому никакой элемент этого отрезка не может быть членом нашей последовательности. Но в нашей последовательности есть числа больше X. Возьмем минимальное из них, оно не меньше $X + 100$. Тогда предыдущее число в последовательности не меньше $(X + 100) - 81 = X + 19$. Оно больше X, но меньше минимального из таких чисел. Противоречие.

На самом деле мы показали, что в последовательности нет чисел больше $X + 100$, а значит мы встретим число с нулевой цифрой среди первых 1001 элементов.

Это значит, что мы можем просто строить последовательность, пока не встретим элемент с нулевой цифрой, после чего этот элемент будет повторяться бесконечно.

В реальности максимальный номер первого элемента с нулевой цифрой — 54, минимальное a_1 для такой последовательности равно 28217.

Задача H. Карантин

D не важно, в конце нужно будет просто домножить ответ на D.

Если N нечётно, то расстояние между любыми двумя домами не более $\frac{N-1}{2}$. Можно заметить, что мы можем всегда проходить ровно столько: если $(N-1)$ раз проходить $\frac{N-1}{2}$ по часовой стрелке, то мы не посетим никакой дом дважды. Таким образом, ответ равен $\frac{(N-1)^2}{2}$.

Если N чётно, то расстояние между любыми двумя домами не более $\frac{N}{2}$, но есть только $\frac{N}{2}$ пар домов, между которым расстояние ровно такое. Следовательно, общая длина маршрута не может превышать $\frac{N}{2} \cdot \frac{N}{2} + (N - 1 - \frac{N}{2}) \cdot (\frac{N}{2} - 1)$. Опять же, эта оценка достижима: если чередовать шаги на $\frac{N}{2}$ и $(\frac{N}{2} - 1)$, постоянно двигаясь по часовой стрелке, то мы не посетим никакой дом дважды. Таким образом, ответ равен $\frac{N^2+(N-2)^2}{4}$.

Если вы ненавидите ифы всей душой, то ответ равен $\frac{(N-1)^2+((N+1) \bmod 2)}{2}$.

Задача I. Сосчитайте треугольники

Так как $x \leqslant y \leqslant z$, чтобы данная тройка образовывала невырожденный треугольник, необходимо и достаточно $z < x + y$. Посчитаем для каждого $s = x + y$, сколько есть способов выбрать (x, y). Для этого переберём x и прибавим 1 на отрезке $[x + B; x + C]$ в оффлайне с помощью префиксных сумм. Ещё раз посчитаем префиксные суммы, теперь мы можем за $O(1)$ посчитать сколько есть способов выбрать (x, y) так, что их сумма больше z. Переберём z, вычислим ответ. Сложность — $O(C)$.

Задача J. Реставрационное расстояние

Первым делом сделаем $M = \min(M, A + R)$ — это верно, так как мы можем эмулировать перекладывание как добавление+убирание. После этого никогда не выгодно и добавлять, и убирать в одном решении, так как всегда можно вместо этого перекладывать.

Пусть мы зафиксировали H — итоговую высоту всех столбиков. Как посчитать минимальную стоимость сделать высоту всех столбиков равной H? В каких-то столбиках не больше H кирпичей, пусть суммарное количество недостающих кирпичей в таких столбиках равно P. В других столбиках не меньше H кирпичей, пусть суммарное количество лишних кирпичей в таких столбиках равно Q. Если $P \geqslant Q$, то нам не хватает $(P - Q)$ камней, поэтому мы должны сделать $(P - Q)$ операций добавления. Больше операций добавления и удаления не будет, и нам нужно будет сделать не менее Q операций перекладывания, так как мы не можем другими способами убрать лишние кирпичи из тех столбиков, где их изначально больше H. Легко понять, что Q операций перекладывания достаточно. Таким образом, стоимость равна $C = A(P - Q) + MQ$. Аналогично, если $Q \geqslant P$, то стоимость будет $C = R(Q - P) + MP$.

Давайте теперь предположим, что $P \geqslant Q$, есть ровно X столбиков, в которых кирпичей не больше H, и ровно $N - X$ столбиков, в которых кирпичей строго больше H. Попробуем увеличить H на 1 и посмотрим, как изменится требуемая стоимость. $P' = P + X$, $Q' = Q - (N - X) = Q - N + X$. $C' = A(P'-Q')+MQ' = A(P+X-Q+N-X)+M(Q-N+X) = A(P-Q)+MQ+AN-M(N-X)$. Видно, что стоимость изменилась на $AN - M(N - X)$. Пока X не меняется, изменение стоимости будет постоянным. В какие моменты меняется X? Когда H равно исходной высоте одного из столбиков. Следовательно, функция стоимости от H кусочно-линейная с изломами в точках, соответствующим исходным высотам.

Один нюанс — мы предполагали, что $P \geqslant Q$. Понятно, что аналогичное будет верно и при $P \leqslant Q$, но при переходе между этими двумя состояниями тоже может появиться излом. Такой переход будет только один, при $H \approx \frac{\sum h_i}{N}$ (под примерным равенством здесь подразумевается, что эта точка может быть нецелой, и нужно добавить $\lfloor \frac{\sum h_i}{N} \rfloor$ и $\lceil \frac{\sum h_i}{N} \rceil$).

Экстремумы кусочно-линейной функции находятся в изломах, поэтому достаточно посчитать стоимости в изломах (исходные высоты и $H \approx \frac{\sum h_i}{N}$) и выбрать среди них минимальную.

Чтобы быстро считать стоимость при фиксированном H, отсортируем изначальные высоты и посчитаем префиксные суммы на массиве отсортированных высот. Тогда с помощью бинарного поиска можно определить, какие столбики меньше H, а какие больше, а затем вычислить P и Q. Вместо бинарного поиска можно использовать два указателя, но это не улучшает итоговую сложность решения, которая составляет $O(N \log N)$ из-за сортировки (и бинарных поисков, если ими пользоваться).

Задача K. Угадать количество делителей

Если $X = p_1^{\alpha_1} \cdot p_2^{\alpha_2} \cdot \ldots \cdot p_k^{\alpha_k}$, то $d(X) = (\alpha_1 + 1) \cdot (\alpha_2 + 1) \cdot \ldots \cdot (\alpha_k + 1)$.

Если простое p входит в X в степени α, а в Q в степени β, то в $gcd(X, Q)$ оно входит в степени $\gamma = \min(\alpha, \beta)$. Если $\gamma < \beta$, то $\alpha = \gamma$, иначе $\gamma = \beta$ и $\alpha \geqslant \gamma$. Мы не знаем X, но мы можем выбирать Q. Если мы выберем Q с известным разложением на простые множители, то мы сможем быстро (за $O(\log Q)$) посчитать всю информацию, которую мы получили из запроса.

После всех запросов мы будем для каждого простого p либо точно знать степень вхождения в X, либо иметь ограничение снизу на степень вхождения. Ограничение сверху мы можем получить из того, что $X \leqslant 10^9$.

Понятно, что мы не можем получить информацию про все простые числа — их слишком много, а запросов слишком мало. Хотелось бы как-то воспользоваться тем, что ответ нужно находить неточно...

Пусть мы смогли понять, что $X = X_1 \cdot X_2$, причём X_1 мы знаем, а у X_2 есть не более t простых делителей (считая кратность). Тогда $d(X_1) \leqslant d(X) \leqslant d(X_1) \cdot d(X_2) \leqslant d(X_1) \cdot 2^t$. Если $t \leqslant 1$, то наш ответ будет иметь относительную погрешность не более 0.5...

Один из способов гарантировать, что у X_2 мало простых делителей, — показать, что у него не может быть маленьких простых делителей. То есть для всех маленьких простых делителей мы должны точно вычислить степень. Это даёт идею решения: для всех простых $p \leqslant B$ (B — некоторая граница) сделаем запрос $Q = p^\beta$, где β выбрано так, что $p^\beta > 10^9$. Таким образом мы узнаем степень вхождения p в X точно.

Эту базовую идею можно улучшить несколькими способами:

1. У X может быть не более 9 различных простых, то есть для большинства простых степень вхождения равна 0. Если мы сможем быстро отсеять ненужные простые, это значительно ускорит решение. И такой способ есть: мы можем сделать запрос $Q = p_1 p_2 \ldots p_s$ для s простых, таким образом мы узнаем, какие из них входят в X хотя бы в первой степени;

2. β можно выбирать так, что $p^{\beta+1} > 10^9$, так как даже если $\gamma = \beta$ и $\alpha \geqslant \gamma = \beta$, мы знаем, что $\alpha \leqslant \beta$, так как иначе $X > 10^9$;

3. Из предыдущего пункта следует, что можно проверять степень для двух простых одновременно, нужно просто сделать запрос равный произведению соответствующих чисел.

Как выбрать B? Видимо, мы хотим, чтобы $B^2 > 10^9$. Но на самом деле нас устроит $t \leqslant 2$: если мы знаем, что $L \leqslant d(X) \leqslant 4L$, то можно ответить $2L$, и относительная погрешность будет не более 0.5. А значит нас устроит $B^3 > 10^9$, то есть $B = 1001$.

Это уже близко к требуемому: есть 168 простых не более 1000, мы можем проверять по 6 простых (на присутствие в X в положительной степени) за один запрос, так как $1000^6 \leqslant 10^{18}$, то есть нужно 28 запросов.

Заметим теперь, что если мы нашли какие-то простые делители X (пусть их произведение есть X_1), то $X_2 \leqslant \frac{10^9}{X_1}$. И если мы проверили все простые, не больше p, причём $X_1 \cdot p^3 > 10^9$, то у X_2 будет не более 2 простых делителей и мы победили.

Осталось воспользоваться тем, что у нас еще есть право на ошибку по абсолютному значению: если $X_1 \leqslant 3$, то нужно просто вывести 8. Либо $X_1 \leqslant 3$ и нас устроит что у X_2 не более трёх простых делителей, либо $X_1 \geqslant 4$, и нам нужно проверить простые числа не более $\sqrt[3]{10^9/4} < 630$. Таких простых чисел 114, поэтому нам точно хватит 19 запросов.

Да, нам нужны еще запросы чтобы точно определить степень вхождения тех простых, которые нашлись в X. Либо таких простых не более двух, и мы потратим 1 запрос, либо их хотя бы 3, и нам достаточно проверять простые только до $\sqrt[3]{10^9/(2 \cdot 3 \cdot 5)} < 330$, которых всего 66, поэтому первая часть решения тратит не более 11 запросов.

Таким образом, по грубым оценкам мы уложимся в 20 запросов. На самом деле это решение тратит не более 17 запросов.

Задача L. Стековая машина

В первую очередь стоит внимательно изучить список доступных инструкций и программы, приведенные в примере. Из них, например, можно понять, как написать цикл, который посчитает сумму всех чисел на стеке:

```
WHILE HAVE2 DO
BEGIN
    ADD
END
```

Мы бы хотели посчитать сумму всех чисел, кроме последнего. Если бы нам было доступно условие `HAVE3`, то это было бы легко. К сожалению, мы не можем понять, что нужно остановиться, пока мы не прибавили n. Хотелось бы самим посчитать, сколько чисел мы сложили (считая n в конце), а потом вычесть это количество (минус 1). Инструкция `SWAP3` позволяет такое сделать, но есть более изящный метод. Количество слагаемых минус 1 — это количество операций сложения, которые мы выполним. Давайте не после всех операций вычтем количество сложений, а при каждом сложении будем уменьшать ответ на 1:

```
WHILE HAVE2 DO
BEGIN
    ADD
    DEC
END
```