Задача 1. Компьютерная сеть

Автор задачи: Сергей Копелиович
Авторы разбора: Денис Денисов, Федор Царев

Условие задачи

Для проведения олимпиады школьников по информатике требуется соединить компьютеры в сеть. Организаторы олимпиады разработали схему соединения компьютеров. В соответствии с этой схемой некоторые пары компьютеров должны быть соединены кабелем, и сигнал сможет дойти по кабелям от любого компьютера до любого другого, возможно, через другие компьютеры.

Некоторые компьютеры могут быть соединены циклически. Цикл называется *простым*, если каждый компьютер из этого цикла соединён ровно с двумя другими компьютерами этого цикла, и в этот цикл никакой кабель не входит более одного раза. Некоторые кабели могут не входить ни в какой цикл.

Известно, что в разработанной схеме никакой кабель не принадлежит двум простым циклам одновременно.

Организаторам олимпиады поручено разместить компьютеры в зале соревнований. При размещении должны выполняться следующие условия:

1. Компьютеры размещаются на плоскости в точках с целочисленными координатами.
2. Координаты компьютеров x и y лежат в диапазоне $0 \le x, y \le 10^6$.
3. Никакие два компьютера не располагаются в одной точке.
4. Кабели являются отрезками прямых.
5. Кабели не пересекаются между собой и не проходят через точки размещения компьютеров, к которым они не подключены.

Требуется написать программу, выполняющую размещение компьютеров по заданному описанию схемы.

Формат входных данных

В первой строке входного файла содержатся числа N и M — количество компьютеров и количество кабелей в схеме ($1 \le N \le 100\,000$, $0 \le M \le 200\,000$). В последующих M строках содержатся пары чисел, разделенных пробелами. Каждая такая пара описывает один кабель, числа представляют собой номера соединенных компьютеров. Компьютеры пронумерованы от 1 до N. Никакая пара не встречается дважды, и никакой кабель не соединяет компьютер с самим собой.

Формат выходных данных

Выходной файл должен содержать N строк. Строка с номером i должна содержать координаты i-го компьютера, разделенные пробелом. Сначала выводится координата x, затем y. Разрешается вывести любой вариант размещения компьютеров, при котором выполняются условия 1–5.

Примечание

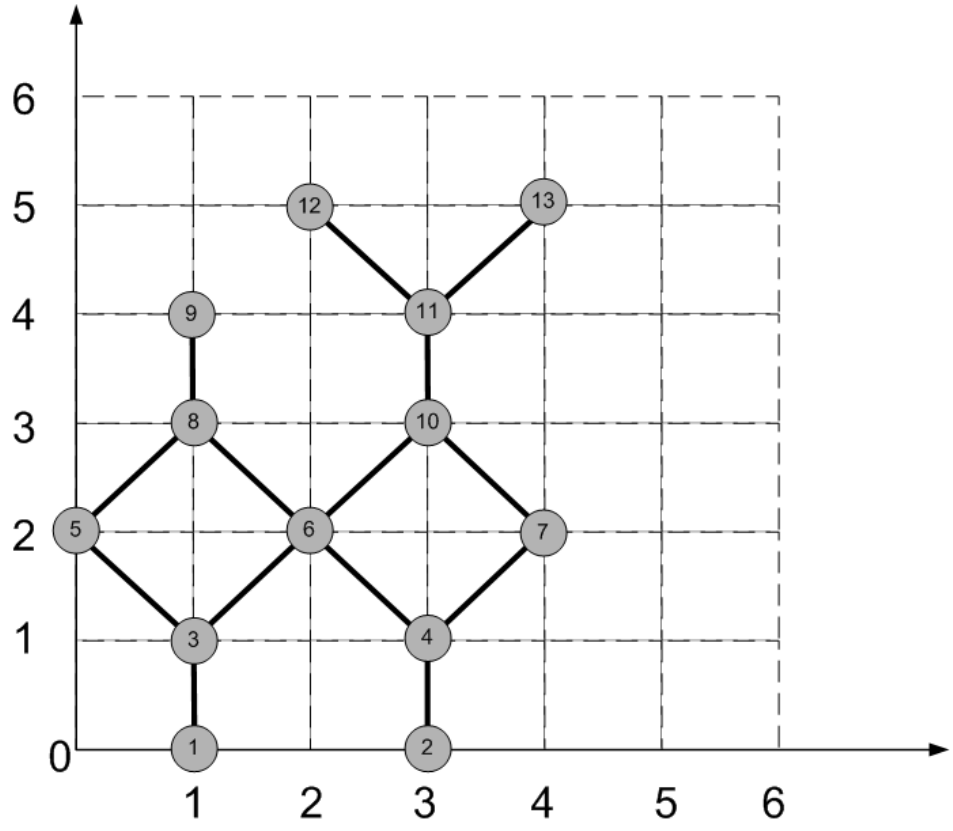
Решения, корректно работающие при отсутствии циклов, будут оцениваться исходя из 40 баллов.

Решения, корректно работающие при наличии только одного цикла, будут оцениваться исходя из 60 баллов.

Пример

network.in	network.out
13 14 11 12 11 13 1 3 3 5 5 8 8 9 8 6 6 3 4 6 4 2 6 10 10 11 10 7 7 4	1 0 3 0 1 1 3 1 0 2 2 2 4 2 1 3 1 4 3 3 3 4 2 5 4 5

Рисунок к примеру



Разбор задачи

Если перевести условие задачи на язык теории графов (вершины соответствуют компьютерам, ребра соответствуют соединениям), то схема представляет собой граф, в котором каждое ребро лежит не более чем на одном простом цикле. Такой граф принято называть *реберным кактусом*.

Рассмотрим для начала решение для случая, когда схема не содержит циклов (в таком случае граф является деревом). Для начала подвесим дерево за какую-нибудь из вершин. После этого будем строить расположение вершин следующим образом. Корень дерева расположим в начале координат. Далее будем рассматривать всех потомков корня. Первого потомка поместим на единицу выше, после чего рекурсивно расположим на плоскости все поддерево первого потомка. При этом запомним полосу x-координат, которую мы использовали для размещения этого поддерева. После этого второго потомка корня расположим на 1 выше корня и на 1 правее всех вершин первого поддерева. Если действовать и дальше таким образом (рис. 1), то все дерево без пересечений ребер будет уложено на плоскости.

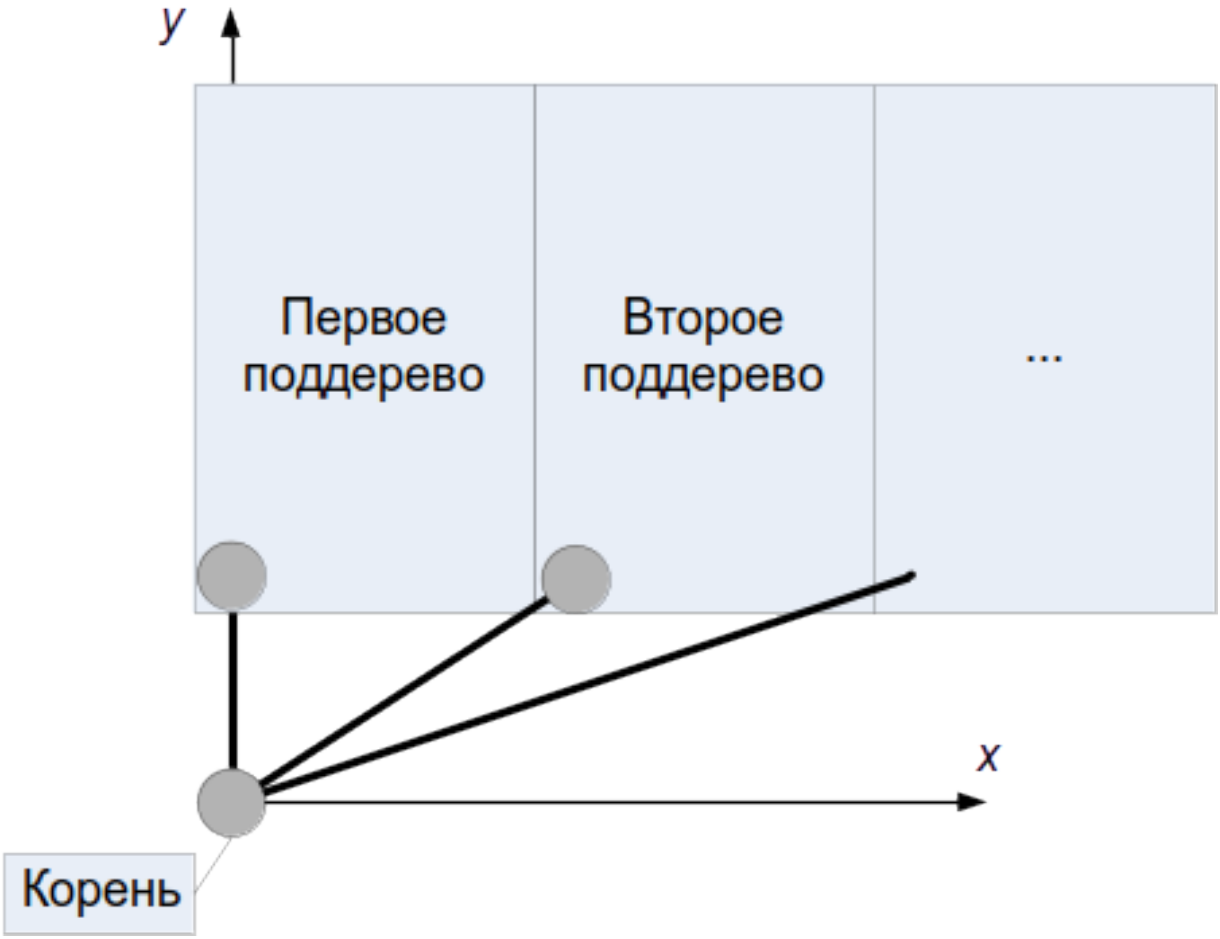


Рис. 1. Укладка дерева на плоскость

Теперь рассмотрим случай, когда в графе существует один цикл. Выделим этот цикл в графе обходом в глубину. В качестве первой вершины выберем вершину, лежащую на цикле и расположим ее в начале координат. Далее, расположим вершины цикла на единицу выше по координате y, пользуясь для укладки поддеревьев, соединенных с вершинами цикла, алгоритмом, использованным в предыдущем случае. После этого все поддеревья, соединенные с первой вершиной аналогичным образом уложим правее вершин цикла и их поддеревьев (рис. 2).

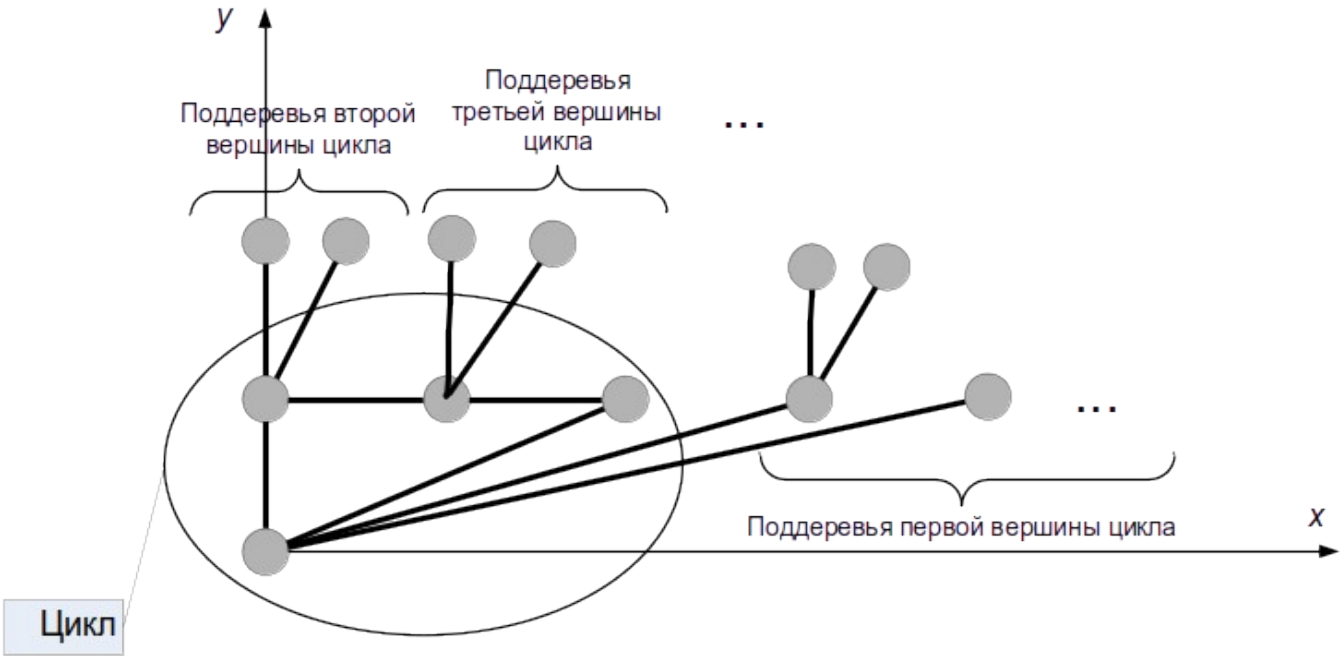


Рис. 2. Укладка графа, содержащего один цикл

Теперь пусть в графе существует несколько циклов, но никакое ребро не лежит более чем на одном цикле. Тогда можно в каждой вершине построить список циклов, на которых она лежит (это можно сделать обходом в глубину). После чего при укладке вершины поочередно уложим все циклы, которым она принадлежит, способом, описанным выше, а потом уложим все подкактусы описанным выше способом (рис. 3).

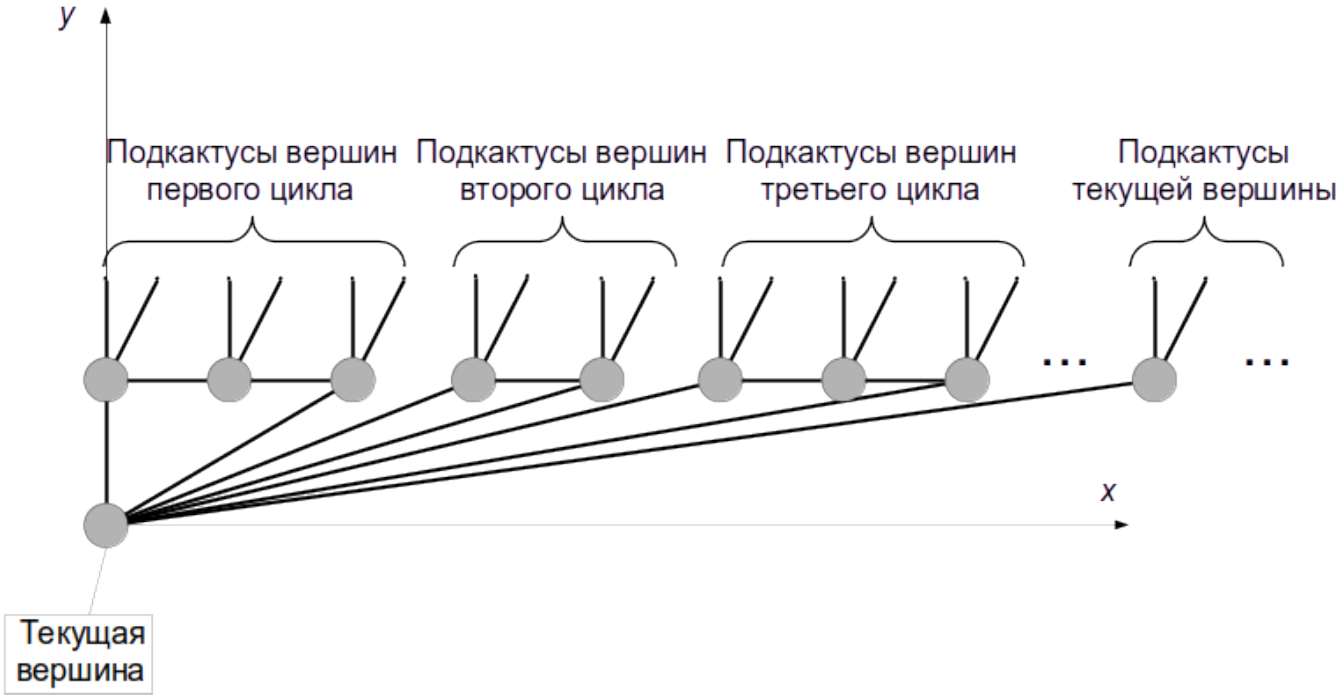


Рис. 3. Укладка нескольких циклов, имеющих общую вершину

Замечания

Описанный способ укладки кактуса на плоскость может быть реализован за время O(V+E), где V – количество вершин в кактусе, а E – количество ребер в нем. Также нетрудно видеть, что в

любом кактусе E <= 3V / 2, посему ограничение в условии задачи было слегка завышено. Также стоит отметить, что случай, в котором каждая вершина лежит не более чем на одном цикле, технически проще в реализации (не требуется хранить списки циклов для каждой вершины), поэтому этот случай в системе тестов был выделен отдельно.

Система оценки

Решения участников тестировались на 50 тестах, каждый из которых оценивался в 2 балла. Тесты для этой задачи классифицировались по двум параметрам – числу вершин графа (компьютеров) и структуре графа. В таблице указано суммарное количество баллов, которым оценивались тесты, в зависимости от значений этих параметров.

Время работы:	$N \le 10$	$N \le 10^3$	$N \le 10^5$	**Всего баллов**
Граф без циклов	10	20	10	**40**
Граф с одним циклом	4	8	8	**20**
Граф, в котором каждая вершина лежит не более чем на одном цикле	4	8	8	**20**
Граф, в котором каждое ребро лежит не более чем на одном цикле	2	4	14	**20**
Всего баллов:	**20**	**40**	**40**	**100**

Задача 2. НГУ-стройка

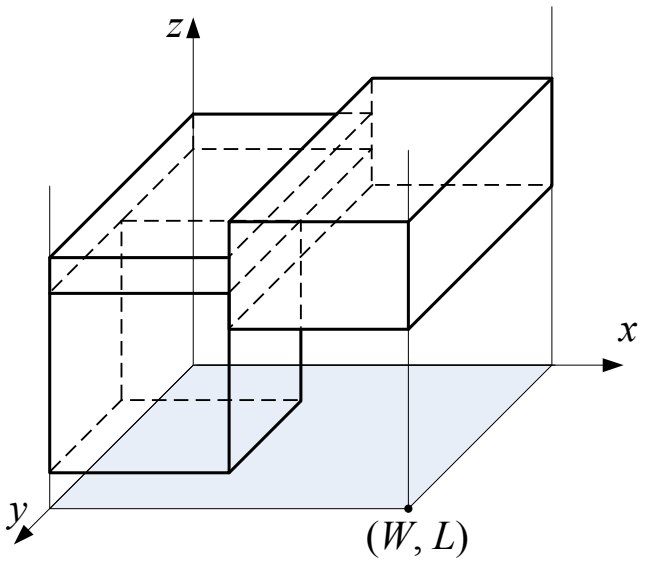
Автор задачи: Сергей Копелиович
Авторы разбора: Сергей Назаров

Условие задачи

Над ареной огромного спортивного комплекса Независимого Главного Университета (НГУ) решили построить перекрытие. Перекрытие будет построено по клеевой технологии и состоять из склеенных друг с другом блоков. Блок представляет собой легкий прямоугольный параллелепипед. Два блока можно склеить, если они соприкасаются перекрывающимися частями боковых граней ненулевой площади.

НГУ представил план комплекса, имеющий вид прямоугольника размером W на L. При этом один из углов прямоугольника находится в начале системы координат, а другой имеет координаты (W, L). Стены комплекса параллельны осям координат.

Подрядчики известили НГУ, что они готовы к определенному сроку изготовить блоки и установить их. Для каждого блока фиксировано место его возможного монтажа, совпадающее по размерам с этим блоком. Места выбраны так, что ребра блоков параллельны осям координат. Места монтажа блоков не пересекаются.



По техническим условиям перекрытие должно состоять из такого набора склеенных блоков, который содержит сплошной горизонтальный слой ненулевой толщины. Торопясь ввести комплекс в эксплуатацию, НГУ решил построить перекрытие из минимально возможного числа блоков.

Требуется написать программу, которая позволяет выбрать минимальное число блоков, которые, будучи установленными на указанных подрядчиками местах, образуют перекрытие, либо определить, что этого сделать невозможно. Высота, на которой образуется перекрытие, не имеет значения.

Формат входных данных

В первой строке входного файла указаны три целых числа: N — количество возможных блоков ($1 \le N \le 10^5$) и размеры комплекса W и L ($1 \le W, L \le 10^4$). Каждая из последующих N строк описывает место монтажа одного блока, определяемое координатами противоположных углов: (x_1, y_1, z_1) и (x_2, y_2, z_2), при этом $0 \le x_1 < x_2 \le W$, $0 \le y_1 < y_2 \le L$, $0 \le z_1 < z_2 \le 10^9$. Все числа во входном файле целые и разделяются пробелами или переводами строк.

Гарантируется, что места установки блоков не пересекаются друг с другом.

Формат выходных данных

Первая строка выходного файла должна содержать либо слово «YES», если перекрытие возможно построить, иначе — слово «NO». В первом случае вторая строка выходного файла должна содержать минимальное число блоков, образующих перекрытие, а последующие строки — номера этих блоков, в соответствии с порядком, в котором они перечислены во входном файле.

Если возможно несколько минимальных наборов блоков, выведите любой из них.

Примечания

Решения, корректно работающие в случае, когда все числа во входном файле не превышают 100, будут оцениваться из 40 баллов.

Рисунок в тексте задачи соответствует первому примеру входного файла.

Примеры входных и выходных данных

ceiling.in	ceiling.out
3 10 10 0 0 6 5 10 7 0 5 1 5 10 6 5 0 5 10 10 8	YES 2 1 3
2 10 10 0 0 6 5 10 7 5 0 5 9 10 8	NO

Разбор задачи

В условии задачи сказано, что решение, верно работающее на всех наборах входных данных, в которых *N, W, L* не превосходят 100, набирает 40 баллов. Для того чтобы набрать 40 баллов, можно поступить следующим образом. Заведем трехмерный массив, каждая клетка которого будет соответствовать единичному кубу пространства. В таком массиве можно «нарисовать» все блоки. Для того чтобы определить ответ, надо проверить для всех координат z от 0 до 100, не образуется ли на этой высоте перекрытие. Проверить это, а также найти количество блоков в перекрытии, можно с помощью массива, в котором «нарисованы» блоки. Так как блоки не пересекаются, то это решение имеет асимптотику $O(V)$, где V — суммарный объем блоков.

Тот факт, что блоки не пересекаются, дает возможность проще проверять, образовалось ли перекрытие из блоков на одной конкретной высоте или нет. Допустим, мы хотим проверить, образовалось ли перекрытие на отрезке $[z, z + 1]$. Рассмотрим все блоки, проекции которых на ось *Oz* содержат этот отрезок. Это именно те блоки, которые могут образовать перекрытие на данном отрезке. Подсчитаем суммарную площадь проекций блоков на плоскость *Oxy*. Если эта площадь равна *WL*, то так как блоки не пересекаются, то они образуют перекрытие.

Можно сделать вывод, что нас не интересуют сами блоки, а только лишь отрезки – проекции блоков на ось *Oz*. Для каждого отрезка запомним некоторое число — площадь проекции соответствующего блока на плоскость *Oxy*. Поэтому задача сводится к следующей. Задан набор помеченных числами отрезков на прямой с концами в точках с целыми координатами. Необходимо найти отрезок длиной 1, который покрыт наименьшим числом заданных отрезков, причем таких, что сумма чисел на них равна *WL*.

На рис. 1 показан пример получения набора отрезков по блокам.

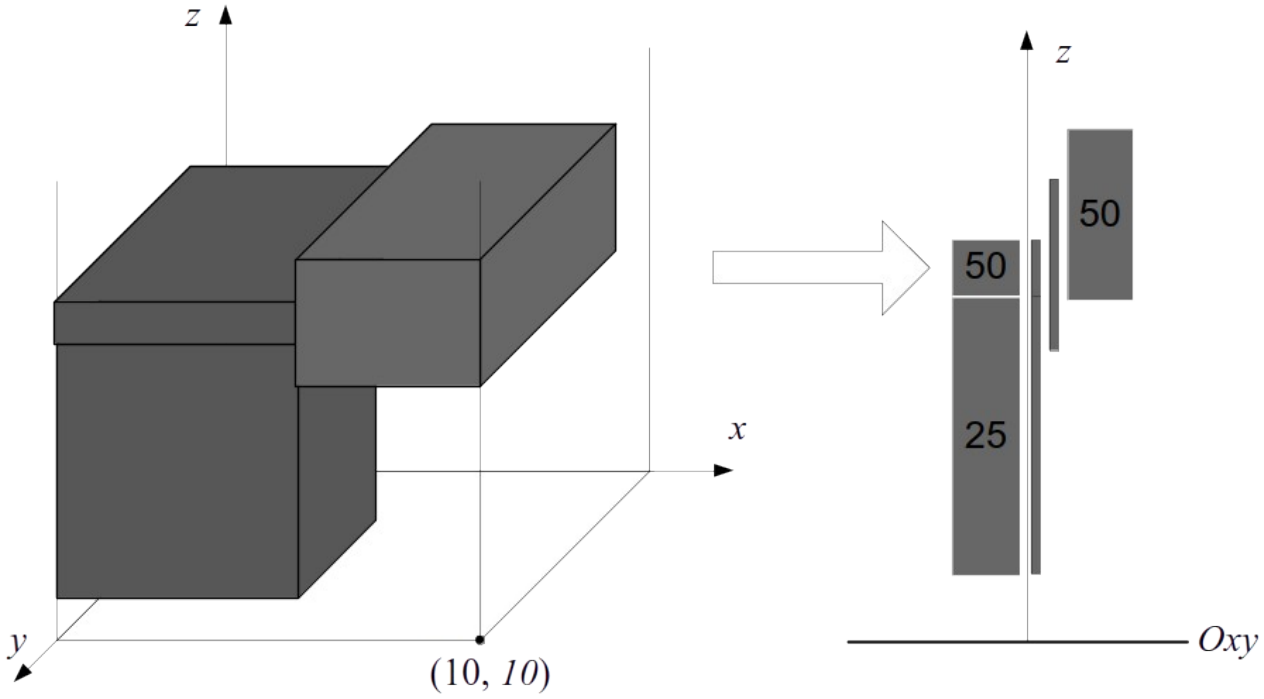


Рис. 1. Получение набора отрезков по блокам

Рассмотрим следующее решение полученной задачи. Будем перебирать все возможные высоты *z*, на которых может образоваться перекрытие (от минимальной координаты *z* во входном файле до максимальной), и проверять, покрывается ли отрезок [*z, z + 1*] отрезками с суммой чисел на них, равной *WL*. Ответ образует минимальное число блоков, полностью покрывающих некоторый отрезок [*z, z + 1*] и имеющих сумму чисел *WL*. Пусть мы проверили *H* координат *z*, для каждой координаты *z* мы будем просматривать все отрезки, поэтому асимптотика такого решения есть *O(NH)*. Такое решение набирает 50 баллов.

Улучшить это решение можно при помощи следующего наблюдения. Не нужно проверять все координаты, достаточно лишь проверять те координаты *z*, в которых начинаются какие либо отрезки. То есть существует только *N* координат *z* для которых достаточно проверить образование перекрытия. Так как для каждого *z* проверка того, что образовалось перекрытие, занимает *O(N)* времени, то мы получили решение с асимптотикой $O(N^2)$. Такое решение набирает 60 баллов.

Из предыдущего решения можно сделать вывод, что нам необходимо знать количество отрезков *K*, покрывающих отрезок [*z, z + 1*], и сумму написанных на них чисел *S* для всех координат *z,* в которых начинается какой либо отрезок. В предыдущем решении для каждого такого *z* мы каждый раз заново вычисляли числа *K* и *S,* однако можно заметить, что если перебирать интересующие нас координаты *z* от меньших к большим, то числа *K* и *S* будут меняться, когда мы будем встречать конец или начало некоторого отрезка. Если мы встречаем начало отрезка, то *K* и *S* увеличиваются, если конец отрезка, то уменьшаются.

Таким образом, у нас будет два вида событий: отрезок начался и отрезок закончился. Каждое событие будет характеризоваться парой чисел: координатой *z* и типом (начался здесь отрезок или закончился). Отсортируем события по координате *z*, а при равенстве координаты *z* по типу: при равных *z* событие конца отрезка должно быть в упорядоченном наборе раньше события начало отрезка. Затем пройдем по событиям и каждый раз, встречая определенное событие, будем изменять значение *K* и *S.* Если встретилось начало отрезка, то мы увеличиваем *K* и *S,* если конец — уменьшаем. При этом если после некоторого события *S = WL,* то, значит, образовалось перекрытие из *K* блоков. Осталось сравнить число *K* с текущим найденным ответом (чтобы выбирать минимальное число блоков, составляющих перекрытие) и запомнить *z* координату этого

события, если K теперь является новым ответом. По этой координате z мы сможем восстановить ответ. На рис. 2 показан пример работы алгоритма.

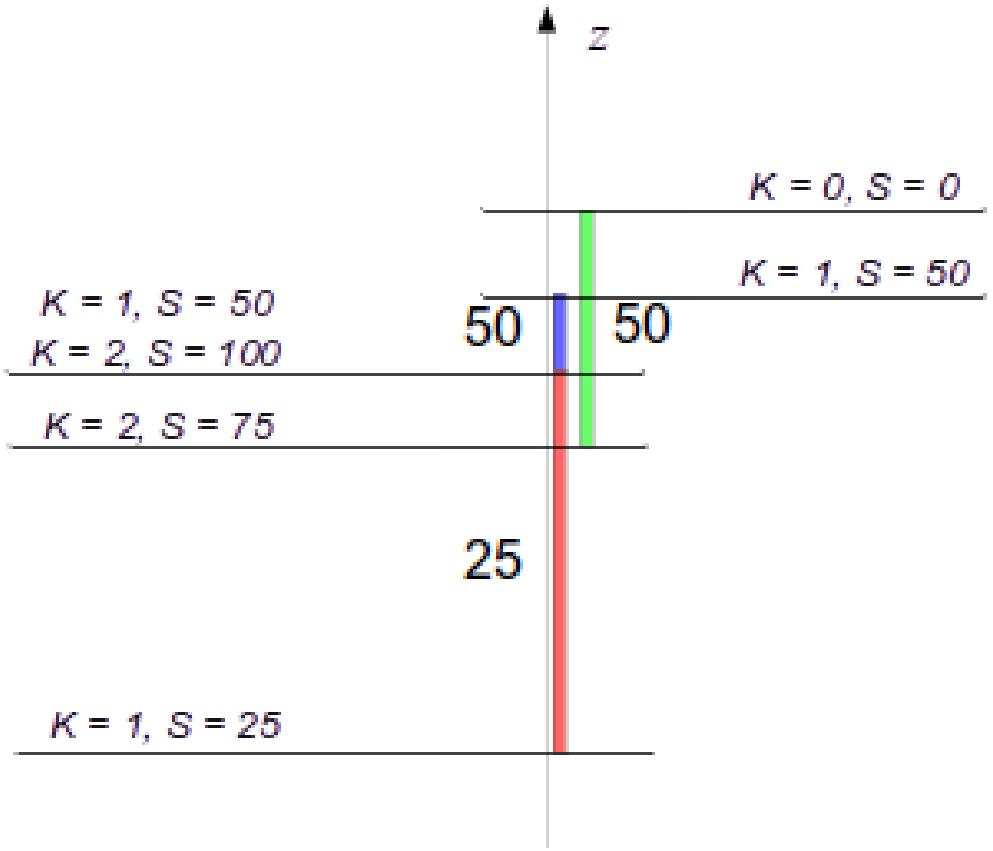


Рис. 2. Пример работы алгоритма

Так как после сортировки решение работает за линейное время, а сортировку можно реализовать за *O(NLogN)*, то полученное решение работает за *O(NLogN)*. Это решение укладывается в ограничение по времени и набирает 100 баллов.

Задача 3. Цифры и числа

Автор задачи: Владимир Алексеевич Кузнецов
Имя выходного файла: Илья Разенштейн

Условие задачи

Как много открытий можно сделать, исследуя числа и составляющие их цифры!

Петя очень любит арифметику, и кроме домашних заданий он постоянно придумывает дополнительные задачи. Однажды он стал прибавлять к натуральным числам сумму составляющих их цифр. Петя обнаружил, что некоторые числа, например 20, не могут быть получены из других чисел в результате такого действия. Эти числа ему не понравились, и он назвал их некрасивыми.

Позже, когда Петя начал изучать информатику, те же исследования он стал проводить с натуральными числами в двоичной системе счисления. Например, двоичное число 1110_2 (в десятичной системе — 14) можно получить из числа 1100_2 (в десятичной системе — 12), прибавив к последнему сумму его цифр:

$$1100_2 + 10_2 = 1110_2.$$

Петя решил исследовать множество двоичных некрасивых чисел. Первые пять некрасивых чисел он нашел без труда: $1 = 1_2$, $4 = 100_2$, $6 = 110_2$, $13 = 1101_2$, $15 = 1111_2$. Продолжить работу он собирается с помощью компьютера.

Требуется написать программу, которая определяет количество двоичных некрасивых чисел, не превосходящих заданного числа n.

Формат входных данных

В первой строке входного файла содержится число n, записанное в десятичной системе счисления ($1 \le n \le 10^{18}$).

Формат выходных данных

В единственной строке выходного файла должно содержаться единственное число — количество двоичных некрасивых чисел, не превосходящих n.

Примечание

Решения, корректно работающие при $n \le 10^6$, будут оцениваться из 40 баллов.

Примеры входных и выходных данных

digits.in	digits.out
1	1
13	4
14	4

Разбор

Вначале опишем простое решение, которое работает за время $O(n \log n)$ и использует память порядка $O(n)$. Будем перебирать числа от 1 до n и для числа i вычеркивать число $i + S(i)$, где $S(i)$ – количество единиц в двоичном разложении i. Те числа, которые мы не вычеркнули, как раз окажутся некрасивыми. Информация о том, вычеркнуто ли число, хранится в массиве. Это решение набирает около 40 баллов.

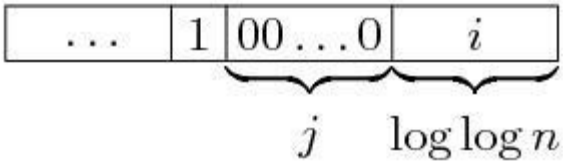
Существуют различные оптимизации данного подхода. Можно хранить массив «побитово», что позволит уместить его в кэш-память процессора при больших значениях n. Можно вычислить

значения функции $S(i)$ для 16-битовых значений i, а затем разбивать число на кусочки по 16 бит и суммировать значения функции от них. Но ключевая оптимизация, которая позволяет получить около 50 баллов, следующая. Так как $S(i) = O(\log i)$, то нужно хранить не весь массив, а только «окно» логарифмической ширины. Данный алгоритм успешно решает задачу за отведенное время для n порядка 10^8. Время работы алгоритма – $O(n \log n)$, использование памяти – $O(\log n)$.

Этот подход можно развить далее, используя массивы констант. Заранее вычислим ответы для чисел вида $10^8 * k$, где k – натуральное число. Когда нам придется решить задачу для произвольного n, найдем наибольшее k такое, что $10^8 * k \le n$ и с помощью предыдущего алгоритма решим задачу для чисел от $10^8 * k + 1$ до n. Вычисление констант для $k \le 1000$ занимает около 30 минут. Данный алгоритм успешно решает задачу для $n \le 10^{11}$ и набирает около 60 баллов.

Теперь опишем два алгоритма, которые при аккуратной реализации набирают полный балл.

Время работы первого алгоритма - $O(\log^5 n)$. Он использует тот факт, что для проверки «некрасивости» числа k достаточно проверить, что $l + S(l) \ne k$ для $k - \log k \le l \le k$.



Рассмотрим какое-либо число от 1 до n. Разобьем его двоичную запись, как изображено на рисунке. Несложно заметить, что при вычитании из него чисел от 0 до $\log n$ первый фрагмент (обозначенный на рисунке многоточием) измениться не может. Отсюда следует, что для проверки «некрасивости» нашего числа достаточно знать только количество единиц в первом фрагменте (конкретное их расположение не играет роли). Обозначим это количество k. Наш алгоритм состоит в следующем:

- Переберем i от 0 до $2^{\log\log n} = \log n$, j от 0 до $\log n - \log\log n - 1$, k от 0 до $\log n - \log\log n - j - 1$.
- Для данных i, j и k проверяем, являются ли такие числа «некрасивыми». Это делается за время $O(\log^2 n)$ наивным методом.
- Если окажется, что числа такого вида действительно являются «некрасивыми», то необходимо подсчитать их количество. Небольшая трудность состоит в том, что числа не должны превосходить n. Чтобы правильно подсчитать искомое количество, придется применить некоторые комбинаторные соображения. Время работы этой части алгоритма – $O(\log n)$.

Возможных троек i, j и k $O(\log^3 n)$, что и дает требуемую оценку времени работы. Отметим, что существует отдельный случай, когда все биты, кроме, возможно, последних $\log\log n$, нулевые. Этот случай разбирается аналогично.

Время работы второго алгоритма, который основывается на технике «разделяй и властвуй», – $O(\log^3 n)$. Сначала научимся считать количество «некрасивых» чисел на интервале $[0,2^k)$ за время $O(k^3)$. Представим наш интервал как $[0,2^{k-1}) \cup [2^{k-1},2^k)$. Количество «некрасивых» чисел на первом интервале посчитаем рекурсивно. Теперь покажем, как получить ответ для второго интервала. Заметим, что влияние первого интервала на второй не слишком велико – для всех чисел l из второго интервала, кроме первых $O(k)$, выполняется такое утверждение: l – «некрасивое» тогда и только тогда, когда «некрасивым» является число $l - 2^{k-1} - 1$. Понять, почему так происходит, нетрудно. Действительно, числа из второй половины отличаются от соответствующих чисел из первой половины ровно в одном (старшем) бите, который увеличивает сумму цифр на единицу. Первые же $O(k)$ чисел являются особыми в том смысле, что на них влияют числа из первого интервала. Будем пересчитывать их «некрасивость» явно за время $O(k^2)$ самым простым методом.

Теперь научимся решать исходную задачу. Это делается практически так же. Требуется всего лишь определить, в какой из двух интервалов попадает наше число n. Если оно попало в первый интервал – вызовем нашу функцию подсчета ответа рекурсивно. Если же во второй - то вызовем функцию подсчета ответа для первой половины и числа $n - 2^{k-1} - 1$. В этом случае некоторые проблемы опять создадут первые $O(k)$ чисел. Опять же, будем их явно обрабатывать.

Система оценки

Решения участников тестировались на 50 тестах, каждый из которых оценивался в 2 балла:

- 1 – 20 тесты: $1 \le n \le 10^6$
- 21 – 30 тесты: $10^8 \le n \le 10^{11}$
- 31 – 40 тесты: $10^{12} \le n \le 10^{14}$
- 41 – 50 тесты: $10^{14} \le n \le 10^{18}$

Задача 4. Легкоатлетический манеж НГУ

Автор задачи: Владимир Алексеевич Кузнецов
Автор разбора: Владислав Кузькоков

Условие задачи

Рядом со спортивным комплексом НГУ решили построить легкоатлетический манеж с M одинаковыми прямолинейными беговыми дорожками. Они будут покрыты полосами из синтетического материала тартана. На складе имеются N полос тартана, длины которых равны 1, 2, …, N метров соответственно (i-я полоса имеет длину i метров).

Было решено использовать все полосы со склада, что определило длину дорожек манежа. Полосы тартана должны быть уложены без перекрытий и промежутков. Разрезать полосы на части нельзя. Полосы укладываются вдоль дорожек, ширина полосы тартана совпадает с шириной беговой дорожки.

Требуется написать программу, которая определяет, можно ли покрыть всем имеющимся материалом M дорожек, и если это возможно, то распределяет полосы тартана по дорожкам.

Формат входных данных

Во входном файле содержатся два целых числа, разделенных пробелом: M — количество дорожек и N — количество полос тартана ($1 \le M \le 1000$, $1 \le N \le 30000$).

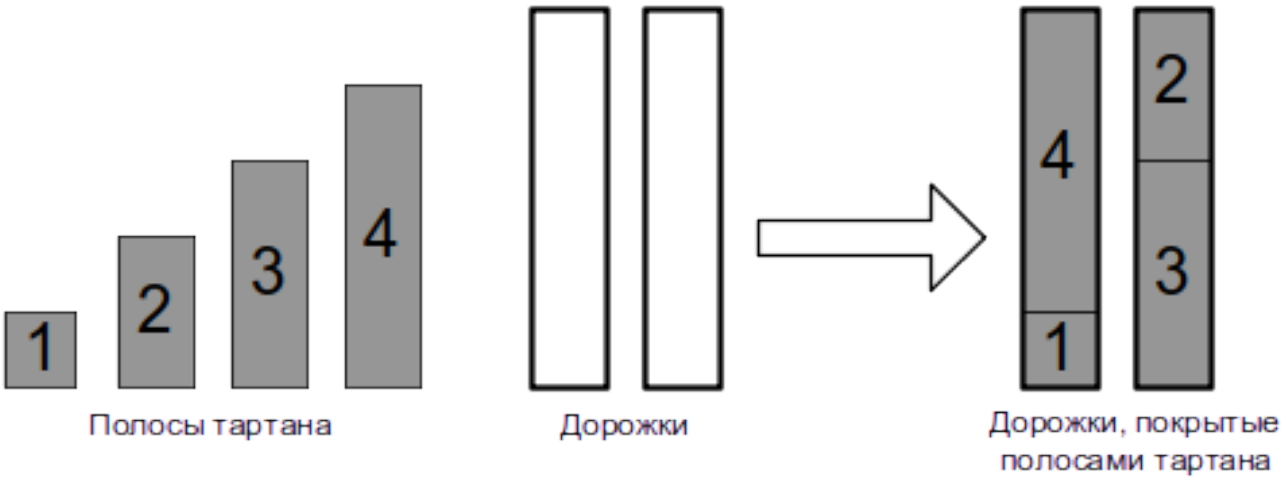
Формат выходных данных

В случае, если распределить имеющиеся полосы тартана на M дорожек одинаковой длины невозможно, то в выходной файл выведите слово «NO».

В противном случае, в первую строку выведите слово «YES». В последующих M строках дайте описание использованных полос для каждой дорожки в следующем формате: сначала целое число t — количество полос на дорожке, затем t целых чисел — длины полос, которые составят эту дорожку. Если решений несколько, можно вывести любое из них.

Примеры входных и выходных данных

manege.in	manege.out
2 4	YES 2 1 4 2 3 2
3 4	NO



Разбор задачи

Обозначим S = N(N+1) / 2M. Поскольку суммарная длина всех полос равна N(N+1)/2, и надо разделить поровну на M дорожек, то S – длина каждой дорожки. Из ограничений на S легко можно вывести два необходимых условия существования решения:

1. S – целое число. То есть N(N+1) должно делиться на 2M нацело.
2. S ≥ N. Если S меньше N, то полоса длиной N не поместится ни в одну дорожку.

Жадное решение 1

Можно реализовать жадное решение: будем выкладывать дорожки по очереди еще не использованными полосами, причем полосы будем выбирать в порядке убывания. Например:

N=15, M=5

15+5 = 14+6 = 13+7 = 12+8 = 11+9 = 10+4+3+2+1

Такое решение набирает 30 баллов. Минимальный тест, на котором это решение не работает: N=23, M=6.

Переборное решение

Из этого жадного решения можно получить перебор с возвратом следующим образом: если после выкладывания полосы решение на следующем шаге не было получено, берем полосу меньшего размера, если она есть. Если нет – возвращаемся на предыдущий шаг. При этом следует делать проверку необходимых условий, чтобы отсекать случаи с ответом NO. Правильная реализация перебора набирает 90 баллов.

Важное наблюдение заключается в том, что из решения для <M, N> всегда можно построить решение для <M, N+2M> по простому алгоритму: разделим N полос на M дорожек, а затем добавим оставшиеся полосы парами (N+1, N+2M), (N+2, N+2M-1), (N+3, N+2M-2), …, (N+M-1, N+M+2), (N+M, N+M+1).

Если в этом решении применить оптимизацию (N-2M, M), то оно тоже наберет 90 баллов.

Первым полным решением является более полное рассмотрение последнего варианта перебора. При данных ограничениях существует 6832 «интересных» теста и перебор не проходят только тесты с N=3740, M=957 и N=3827, M=979, а также полученные из них с помощью прибавления 2M к N. Для этих тестов можно получить решение любым другим способом. Перебор, не использующий наблюдение, также наберет 90 баллов, так как, перебор для больших N в данном случае работает быстрее.

Также в данной задаче можно применить следующее отсечение: если в данной ветке перебора было слишком много откатов, то считаем, что шансов найти решение в этой ветке немного и сразу выполняем возврат. При должном подборе количества рассматриваемых веток решение может набирать до 100 баллов.

Жадное решение 2

Существует другой вариант жадного решения: брать полосы в порядке убывания размера. Если существует дорожка, которую мы можем закончить этой полосой, делаем это, иначе кладем полосу на дорожку, где осталось больше всего свободного места. Такое решение набирает 88 баллов. Если после того как не удалось завершить дорожку этой полосой попытаться закончить дорожку этой полосой и какой-либо другой неиспользованной, то решение наберет 100 баллов.

Конструктивное решение

Оказывается, что указанные в начале разбора необходимые условия являются также и достаточными. Докажем это утверждение, предъявив явную конструкцию решения.

Если S ≥ 2N, то можно распределить N-2M полос по M дорожкам рекурсивно и воспользоваться этим наблюдением. Если S = N, то решение можно построить явно: (N), (N-1,1), (N-2,2)… ((N+1)/2, (N-1)/2).

Рассмотрим оставшиеся содержательные случаи: $N < S < 2N$. Будем брать полосы парами: самую большую из оставшихся и такую, что сумма составит S: (N, S-N), (N-1, S-N+1), … . Далее возможно 2 случая:

1. Если S – нечетно, то всего пар будет N-(S-1)/2. Последней парой будет ((S+1)/2, (S-1)/2). Нераспределёнными останутся полосы с длинами 1, ..., S-N-1, которыми надо будет выложить оставшиеся M+(S-1)/2-N дорожек. Распределяем рекурсивно.
2. Если S – четно, то всего пар будет N-S/2. Последней парой будет (S/2+1, S/2-1). Неиспользованными останутся полосы с длинами 1, ..., S-N-1 и S/2. Нужно этими полосами выложить M+S/2-N дорожек. Разделим каждую дорожку на 2 равных части (длиной по S/2 каждая). Тогда можно 2M+S-2N-1 дорожку длиной S/2 выложить полосами с длинами 1, ..., S-N-1 рекурсивно и добавить одну дорожку, состоящую из одной полосы длины S/2.

В обоих случаях N будет уменьшаться, поэтому рекурсия завершается (на самом деле она работает достаточно быстро).

Система тестов

Решения участников тестировались на 50 тестах, каждый из которых оценивался в 2 балла. Набор тестов содержал 35 тестов против жадного алгоритма, 10 тестов против неоптимальных вариантов перебора, по 5 тестов против оптимальных реализаций перебора.

Задача 5. Граффити на заборе

Автор задачи: Денис Денисов
Автор разбора: Денис Денисов

Условие задачи

Около прямолинейного забора, состоящего из N одинаковых бетонных плит, проводится конкурс граффити, в котором участвуют M граффити-художников. Художники должны разрисовать все плиты своими произведениями за наименьшее возможное время.

Плиты пронумерованы числами от 1 до N, граффити-художники имеют номера от 1 до M. Первоначально i-й граффити-художник находится около плиты с заданным номером p_i. Каждому художнику требуется b минут на разрисовывание любой плиты. Каждую плиту должен разрисовать ровно один граффити-художник.

В начале работы, а также после разрисовывания любой плиты граффити-художник может перейти к любой неразрисованной плите. Время перемещения граффити-художника от любой плиты к соседней с ней одинаково и равно a минут. Таким образом, чтобы перейти от плиты с номером i к плите с номером j художнику требуется $a \cdot |i - j|$ минут.

Требуется написать программу, которая поможет участникам конкурса разрисовать все плиты за минимальное возможное время.

Формат входных данных

В первой строке входного файла указаны числа N — количество плит в заборе и M — количество граффити-художников ($1 \le N, M \le 100000$). Во второй строке заданы два целых числа: a — количество минут, которое требуется для перехода от любой плиты к соседней, и b — количество минут, которое требуется граффити-художнику на разрисовывание одной плиты ($1 \le a, b \le 10^6$). В третьей строке заданы M чисел $p_1, p_2, \ldots, p_M$ — начальные положения граффити-художников ($1 \le p_i \le N$).

Формат выходных данных

В первую строку выходного файла выведите минимальное количество минут, требуемых художникам для выполнения работы.

В последующих M строках выведите описание действий художников. В i-й из этих строк должно содержаться описание действий i-го художника: количество плит, которые должен разрисовать этот художник, и номера этих плит в очередности их разрисовывания. Если оптимальных решений несколько, можно вывести любое из них.

Примечание

Решения, корректно работающие при $M \le 2$, будут оцениваться из 40 баллов.

Пример входных и выходных данных

fence.in	fence.out
3 4 2 3 3 1 3 3	5 1 2 1 1 1 3 0
2 1 1 1 1	3 2 1 2

Разбор задачи

Во-первых, разберемся, как выглядят участки забора, которые красит тот или иной художник. Несложно видеть, что они представляют собой отрезки. Действительно, если какой-то художник красит два участка забора, не соединенных друг с другом, можно так перенаправить их действия, чтобы каждый художник красил связный отрезок, причем время покраски не увеличится. Это наблюдение основано на том факте, что все художники одинаковы, и если два художника проходят навстречу друг другу, можно считать, что они не прошли навстречу, а встретились друг с другом и развернулись в обратные стороны (рис. 1).

Рис. 1.

Также заметим, что если художник номер i изначально стоял левее художника номер j, то и отрезок, который красит художник i (обозначим его $[l_i,r_i]$), находится левее отрезка, который красит художник номер j. Это тоже следует из наблюдения о развороте художников, идущих навстречу друг другу.

Ну и, наконец, заметим еще один факт: если художники могут раскрасить весь забор за T минут, то они могут его раскрасить и за T+1 минуту (действительно, можно ведь в последнюю минуту просто ничего не делать). Аналогично, если художники не могут раскрасить весь забор за T минут, то они не смогут этого сделать и за T-1 минуту.

Эти наблюдения позволяют организовать двоичный поиск по ответу (времени покраски): завести две оценки ответа (оценку снизу L, для которой художники гарантированно не успеют разрисовать забор и оценку сверху R, для которой они гарантированно успеют). После чего полагать M = (L+R)/2 и проверять, могут ли художники за M минут раскрасить весь забор. Если ответ существует, полагать R=M, иначе полагать L=M.

Осталось указать начальные значения L и R. Ясно, что в качестве начального L подойдет ноль (так как за ноль единиц времени ничего покрасить нельзя), а в качестве начального R подойдет (2a+b)N, так как за это время один художник может раскрасить весь забор, вне зависимости от своего начального положения.

Осталось решить более простую задачу: проверить, могут ли художники раскрасить весь забор за T минут для некоторого T. Отсортируем художников по возрастанию их изначального положения. В силу наблюдения о покрашиваемых отрезках, первый художник красит какой-то отрезок, второй художник красит отрезок, примыкающий справа к первому и т.д. Если обозначить отрезок i-го художника за $[l_i,r_i]$, то получим, что $l_1 = 1$, $r_1 = l_2 - 1$, $r_2 = l_3 - 1$, … , $r_m = n$.

Пусть художник номер i красит отрезок $[l_i,r_i]$. Определим, какое время ему для этого потребуется. Если он находится вне отрезка ($p_i < l_i$ или $r_i > p_i$), то художнику необходимо добраться до ближайшей к нему границы этого отрезка, а затем покрасить весь отрезок. Если он находится внутри отрезка, то на самом деле ему требуется сделать то же самое (поскольку в любом случае придется возвращаться к исходной позиции, можно покрасить забор в процессе возвращения). Тогда время, необходимое художнику определяется следующей формулой:

$t_i = (\min(|p_i - l_i|, |p_i - r_i|) + (r_i - l_i)) \cdot a + (r_i - l_i + 1) \cdot b$

Пусть теперь нам известно l_i. Каким образом можно определить r_i? Это можно сделать исходя из следующих соображений: во-первых, если художник за время T может покрасить отрезок $[l_i;r_i]$, то он может покрасить отрезок $[l_i;r_i-1]$, а во-вторых, нам выгодно, чтобы он покрасил как можно больше (сколько успеет за время T). Тогда можно определить с помощью линейного прохода, добавляя художнику покрашиваемые плиты до тех пор, пока он успевает их покрасить (время покраски определяя по формуле, приведенной выше). Теперь, определив правую границу для первого художника, мы автоматически получаем левую границу для второго художника и

можем тем же способом определить его правую границу. Как только правая граница для очередного художника стала больше, либо равна N, мы можем сделать вывод, что за время T художники могут покрасить весь забор. Если же такого не происходит, художники не могут покрасить весь забор.

Оценим время работы приведенного решения. Изначальная сортировка художников по неубыванию начальной позиции занимает время O(MlogM). Учитывая, что позиции не превосходят N, можно применить сортировку «подсчетом» и получить оценку O(N+M). Далее, двоичный поиск по ответу добавит в оценку времени работы O(log((2a+b)N)). Теперь оценим алгоритм определения возможности покраски забора за время T. Заметим, что при рассмотрении каждого художника просматривается как минимум одна плита, а если она добавляется к отрезку данного художника, то добавленная плита более не рассматривается. Таким образом, выполняется не более M+N просмотров плит, таким образом время работы внутреннего алгоритма есть O(M+N), а итоговая оценка времени работы есть O(N+M+(N+M)log((2a+b)N)).

В приведенном решении все хорошо, однако в нем есть более чем линейная зависимость от N. Можно ли от нее как-то избавиться? Да, можно. Для этого сортировать художников надо любым быстрым алгоритмом за O(MlogM), а для определения правой границы каждого художника воспользоваться бинарным поиском, вместо линейного (бинарный поиск корректен в силу соображений, приведенных в решении). В таком варианте время работы оценивается как O(MlogM+MlogNlog((2a+b)N)). Это позволяет решать задачу для больших значений N.

Однако и это время работы можно еще немного улучшить. Для этого надо более внимательно рассмотреть возможные действия художника, для которого известны его начальная позиция p_i и левая граница отрезка, который он красит l_i. Здесь возникает два случая:

1) $l_i \le p_i$. В таком случае действия художника весьма просты: он подходит к левой границе отрезка, а затем двигаясь вправо красит плиты до тех пор, пока ему хватает времени (рис. 2).

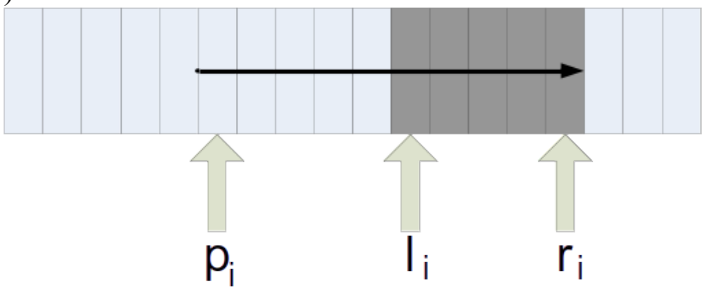


Рис. 2.

2) $l_i \ge p_i$. В таком случае у художника есть три варианта:
   1. Художник идет влево, по пути начиная в какой-то момент красить плиты, мимо которых проходит (рис. 3).

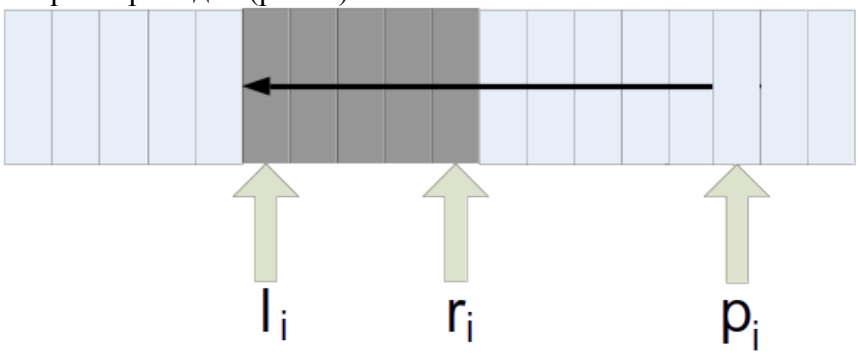


Рис. 3.

   2. Художник идет влево, доходит до левой границы своего отрезка, и затем красит его, двигаясь вправо (рис. 4).

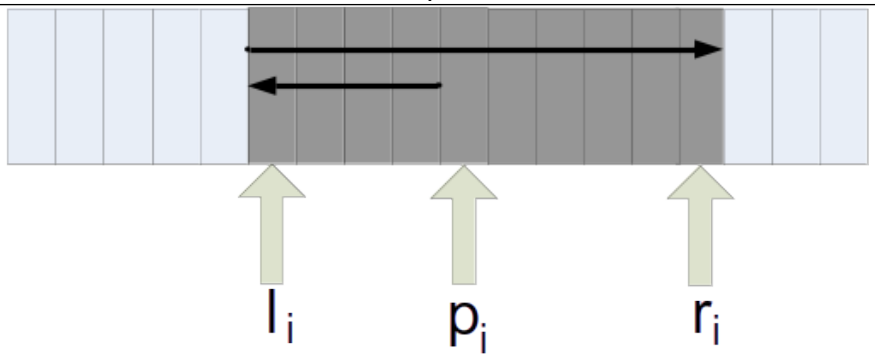


Рис. 4.

3. Художник идет вправо, доходит до правой границы своего отрезка, а затем красит его, двигаясь влево (рис. 5).

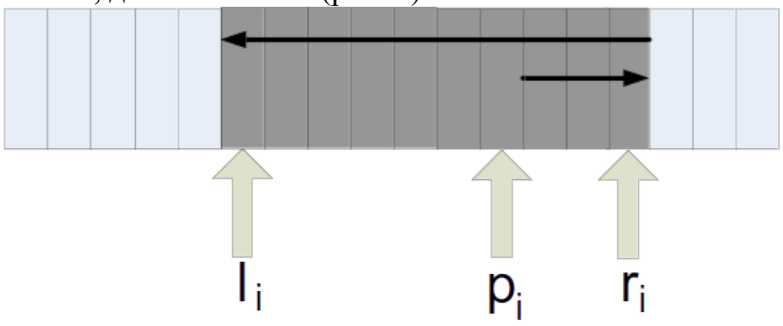


Рис. 5.

Во всех этих случаях правая граница отрезка может быть определена за время O(1). Таким образом итоговое время работы всего алгоритма будет O(MlogM+Mlog((2a+b)N)).

Все вышеприведенные решения набирают 100 баллов.

Частичные решения

Для решения задачи при небольших N и M можно применить динамическое программирование. Обозначим за F[i, j] минимальное время, которое потребуется первым I художникам для покраски первых j плит. Тогда пересчет можно выполнить по следующей формуле: F[i,j] = min(max(F[i-1,k], Time(i, k + 1, j))) по всем k от 0 до j. Здесь за Time(i, l_i, r_i) обозначено время, требуемое i-му художнику для покраски отрезка [l_i;r_i]. Такое решение работает за время пропорциональное $O(N^2M)$ и набирает 40 баллов.

Также можно рассмотреть случай M=2 следующим образом: перебрать границу отрезка, который будет красить художник номер 1, а общее время покраски определять по формуле приведенной выше. Такое решение тоже набирает 40 баллов.

Возможные ошибки при реализации

1) Оптимальное время покраски может быть не представимым в 32-битном типе данных. Решения, использующие 32-битный тип данных, набирает не более 80 баллов.

2) Если производить сортировку художников медленным алгоритмом за $O(M^2)$, то такое решение также наберет не более 80 баллов.

3) Если не производить сортировку художников, то такое решение наберет не более 30 баллов.

Система тестов

Решения участников тестировались на 50 тестах, каждый из которых оценивался в 2 балла. Тесты для этой задачи классифицировались по двум параметрам – длине забора n и числу

художников m. В таблице указано суммарное количество баллов, которым оценивались тесты, в зависимости от значений этих параметров.

Время работы:	$M \le 2$	$M \le 100$	$M \le 10^5$	**Всего баллов**
$N \le 100$	30	10	0	**40**
$N \le 10^5$	10	4	46	**60**
Всего баллов:	**40**	**14**	**46**	**100**

Задача 6. Стековый калькулятор

Автор задачи: Денис Денисов
Авторы разбора: Виталий Вальтман,
Сергей Копелиович

Условие задачи

В кабинете информатики есть старый *стековый калькулятор*. Он содержит K ячеек памяти, организованных в виде *стека*. Первая ячейка называется *вершиной* стека. На индикаторе калькулятора отображается содержимое вершины стека, если стек непуст.

Над стеком может выполняться операция *проталкивания*. Применение этой операции приводит к записи числа на вершину стека. Перед этим, если в стеке уже были числа, то каждое из них перемещается в ячейку с номером на единицу больше. Если в стеке уже находится K чисел, то выполнение операции проталкивания невозможно.

Калькулятор позволяет выполнять арифметические операции. Они применяются к числам, хранящимся во второй и первой ячейках стека. Результат операции записывается в первую ячейку стека, а число из второй ячейки удаляется. После этого, если третья ячейка непуста, то число из неё переписывается во вторую, если есть число в четвертой ячейке — оно переписывается в третью и так далее до последней занятой ячейки, которая становится пустой. Для выполнения арифметической операции в стеке должно быть хотя бы два числа. Например, при выполнении операций сложения или умножения, значения соответственно суммы или произведения чисел из первой и второй ячеек помещаются на вершину стека. Операция вычитания выполняется так: из содержимого второй ячейки стека вычитается содержимое первой ячейки.

Известно, что калькулятор неисправен. Из цифровых клавиш работает только клавиша «`1`». Нажатие этой клавиши приводит к проталкиванию в стек числа 1. Например, попытка набрать число 11, два раза нажав клавишу «`1`», приводит к тому, что в стек два раза проталкивается число 1. Из операций работают только сложение (клавиша «+»), умножение (клавиша «`*`») и вычитание (клавиша «−»). Если в результате вычитания получено отрицательное число, то калькулятор зависает.

Легко заметить, что на индикаторе возможно получить произвольное натуральное число. Например, чтобы получить число 3, необходимо дважды нажать клавишу «`1`», затем клавишу «+» (на индикаторе после этого появится число 2), затем один раз нажать клавишу «`1`» и один раз — клавишу «+». При $K > 2$ того же результата можно достичь иначе, трижды нажав клавишу «`1`», а затем два раза клавишу «+». Дополнительно используя операции умножения и вычитания, в некоторых случаях число N можно получить быстрее, чем сложив N единиц.

Требуется написать программу, которая определяет, каким образом можно получить на индикаторе калькулятора заданное число N, выполнив минимальное количество нажатий клавиш. Стек изначально пуст.

Формат входных данных

В единственной строке входного файла записаны два натуральных числа — N и K ($1 \le N \le 10^9$, $2 \le K \le 100$).

Формат выходных данных

В первой строке выходного файла выведите минимальное количество нажатий клавиш, необходимых для получения числа N. Если число нажатий не превосходит 200, то дополнительно выведите во второй строке оптимальную последовательность нажатий, приводящих к появлению данного числа на индикаторе.

Последовательность необходимо выводить без пробелов. Клавиши обозначаются символами «`1`» — протолкнуть число 1 в стек, «+» — выполнить операцию сложения, «`*`» —

выполнить операцию умножения, «−» — вычесть из значения второй ячейки стека значение первой ячейки.

В результате выполнения выведенной последовательности нажатий на индикаторе должно отображаться число N. Если оптимальных последовательностей нажатий несколько, разрешается выводить любую из них.

Примечания

Решения, корректно работающие при $N \le 100$ и $K \le 10$, будут оцениваться из 40 баллов.

Решения, корректно работающие при $N \le 10^6$ и $K \le 100$, будут оцениваться из 80 баллов.

Примеры входных и выходных данных

stack.in	stack.out
6 3	9 111++11+*
11 4	15 11+111++*11+*1-

Разбор задачи

Решение задачи основывается на методе динамического программирования и алгоритмах поиска кратчайших путей в графе. Вопрос в том, что выбрать в качестве состояний (вершин графа) и как этот граф обрабатывать. Различные подходы к этому вопросу давали различные (частичные и полное) решения.

В качестве вершины графа можно рассматривать набор чисел, содержащихся в стеке, а ребра графа в этом случае будут вычисляться на основе операций из условия задачи. Тогда на получившемся графе достаточно запустить поиск в ширину, так как все ребра имеют единичный вес. Для N ≤ 100 при аккуратной реализации это решение успешно работало. Таким образом можно было получить 40 баллов.

В качестве состояния можно рассматривать пару (n, k), где n – число, которое мы хотим получить, а k – число ячеек стека, которое для этого можно использовать. В итоге на стеке должно оказаться только одно число — n. Число n мы могли получить одним из трех способов: n = a + b, n = a – b, n = a * b. Значит на предыдущем шаге на стеке было два числа — a и b. Число a мы получили используя все k ячеек стека, а когда получали число b на стеке уже лежало число a (итого пользоваться можно было только k-1 ячейкой). Получаем переход

$f(n, k) = \min(f(a, k) + f(b, k-1) + 1)$ (*), где $f(n, k)$ — это ответ для задачи, а числа a и b такие, что одно из чисел $a+b$, $a-b$, $a*b$ равно n.

Если догадаться до того, что числа больше чем 2N использовать не выгодно, то получается, что число состояний есть O(NK), а из каждого можно сделать O(N) переходов (то есть суммарно O(N) ребер). Функцию f хочется посчитать с помощью динамического программирования, используя формулу (*).

Проблема заключается в том, что есть циклы (например f(70, k) ← f(75, k) + f(5, k-1) + 1 и f(75, k) ← f(70, k) + f(5, k - 1) + 1). Для решения этой проблемы можно применить итерационный алгоритм – изначально f(1, k) = 1 для всех k > 0, а остальные f (n, k) = +∞.

Теперь перебираем все возможные a, b и k и, если это можно сделать, улучшаем значения f(a+b,k), f(a*b,k) и f(a-b,k). Повторяем этот процесс, пока происходят улучшения. Получается алгоритм (похожий на алгоритм Форд-Белмана) работающий за время $O(TE)=O(KN^2 \log N)$, где E — число переходов (KN^2), а T — число итераций процесса. T не больше чем f(n, k), которое в свою очередь есть $O(\log_2 N)$ при k > 2 (эту оценку можно получить из жадного алгоритма, который умеет делать только 2 операции: +1 и *2).

Случай k = 2 рекомендуется разобрать отдельно. Это решение проходило все тесты при N ≤ 1000 и получало 50 баллов. Аналогично можно получить «алгоритм Дейкстры» работающий за время $O(V^2)=O((NK)^2)$, где V — число вершин (KN).

Для получения лучших решений предполагалось провести исследование ответов для маленьких N и K. Первое наблюдение — при K > 5 ответы не отличаются от ответов для K=5. Отсюда вывод: можно решать задачу для K ≤ 5.

Теперь будем сокращать число переходов. Заметим, что число переходов типа «умножение» имеет порядок $N\log N$, так как оно в точности равно сумме количеств делителей у всех чисел от 1 до N. А число переходов типа «сложение» и «вычитание» $O(N^2)$. Второе наблюдение — поскольку $f(n,5)=O(\log_2 N)$, при сложении и вычитании можно перебирать только те a и b, для которых одно из чисел маленькое. На самом деле «маленькие» — это числа, не превосходящие трёх.

При помощи этих наблюдений мы сокращаем число переходов до O(NlogN), а число состояний до O(N). И получаем решение, работающее за время $O(N\log^2 N)$. Это решение укладывается в ограничение по времени для всех $N\le 10^6$ и получает 80 баллов.

Чтобы решить задачу полностью осталось сделать два улучшение. Посмотрим, какие формулы переходов у нас получились после сделаных допущений. Переходы типа «умножение» не изменились, а переходы типа «сложение» и «вычитание», если учесть, что для $|a|\le 3$ верно, что $f(a,k)=2a-1$, будут выглядеть так:

$f(n,k) \leftarrow f(n+a,k)+2|a|$ (**), где $|a|\le 3$.

Далее можно заметить, что невыгодно делать два сложения или вычитания подряд. Поэтому переход (**) можно заменить на такой:

$f(n,k) \leftarrow f(x,k)+f(y,k-1)+2|a|+1$, где $x*y=n+a$ и $|a|\le 3$.

Заметим теперь, что тогда n>x, n>y при n>5, а для n=1..5 f(n,k)=2n-1. Таким образом наш граф стал ацикличным, поэтому можно применить метод динамического программирования и получить решение за O(числа переходов). Теперь заметим, что конечное состояние (N,K) достижимо далеко не из всех состояний, а число состояний, из которых конечное достижимо, растет примерно как $\sqrt{N}$. Итого: если мы будем рассматривать в процессе работы алгоритма только «осмысленные» состояния, мы получим, решение укладывающееся в ограничение по времени и по памяти для $N\le 10^9$.

Система тестов

Решения участников тестировались на 50 тестах, каждый из которых оценивался в 2 балла. В таблице указано суммарное количество баллов, которым оценивались тесты, в зависимости от диапазона, в котором находятся числа N и K.

	N≤100	**N≤1000**	**N≤10^5**	**N≤10^6**	**N≤10^9**	**Всего**
K=2	4	0	0	0	2	**6**
K=3	16	4	6	6	2	**34**
K=4	8	0	2	6	6	**22**
K=5	2	0	0	0	6	**8**
K=6	2	0	0	0	0	**2**
K=7..10	8	2	0	0	0	**10**
K=11..100	0	4	2	8	4	**18**
Всего	**40**	**10**	**10**	**20**	**20**	**100**