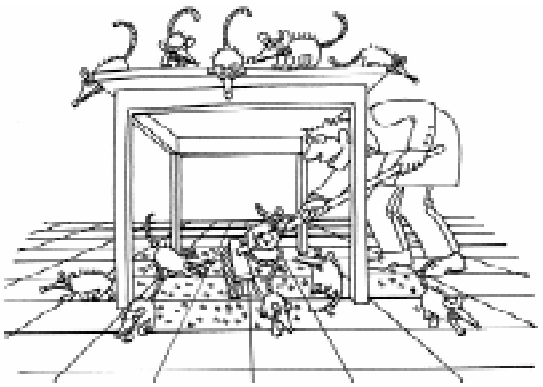
Школа современного программирования

Станкевич Андрей Сергеевич

РАЗБОР ЗАДАЧ ТРЕТЬЕЙ ВСЕРОССИЙСКОЙ КОМАНДНОЙ ОЛИМПИАДЫ ШКОЛЬНИКОВ ПО ПРОГРАММИРОВАНИЮ

Полные тексты программ и тесты жюри можно найти на сайте школьных олимпиад http://neerc.ifmo.ru/school.

ЗАДАЧА A. МОНСТРЫ



В одной секретной лаборатории вывели новый вид маленьких монстров размером чуть больше суслика. В ходе исследований ученые решили поставить следующий эксперимент. В центре комнаты устанавливается прямоугольный стол, поверхность которого разбита на $N \times M$ клеток размера 1×1. В начальный момент времени на некоторых его клетках располагаются монстры, смотрящие параллельно сторонам стола. По команде экспериментатора монстры начинают двигаться по прямой в ту сторону, в которую они смотрят, доходят до края стола и спрыгивают на пол. Там их собирает лаборант Петя и относит в клетку.

Поскольку у монстров очень грязные лапки, они оставляют следы на тех клетках, на которых они побывали. Так как отмывать стол придется лаборанту Пете, его заинтересовал вопрос в каком количестве клеток побывают монстры. Помогите ему решить эту сложную задачу.

Формат входных данных

Первая строка входного файла содержит числа M и N – размеры лабораторного стола ($1 \le M, N \le 10^6$). Следующая строка содержит число K – количество монстров ($0 \le K \le 10^3$). Следующие K строк содержат описания монстров – два целых числа и один символ из множества {N, E, S, W} – начальные координаты и направление соответствующего монстра (соответствие направлений и координат приведено на рисунке 1). Символ отделен от чисел ровно одним пробелом.

Формат выходных данных

В выходной файл выведите единственное число – количество клеток стола, на которых побывают монстры.

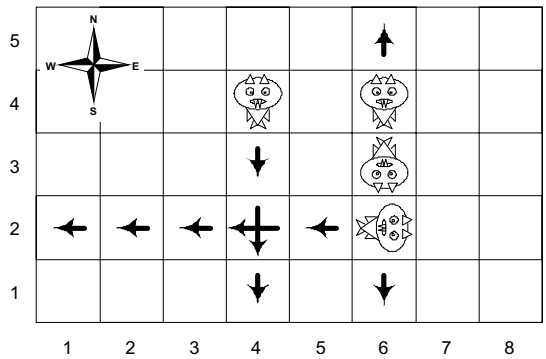


Рисунок 1.

Пример

Пример соответствует расположению монстров, приведенному на рисунке 1.

monsters.in	monsters.out
8 5 4 4 4 S 6 2 W 6 3 N 6 4 S	13

Решение

Подсчитаем отдельно количество клеток, на которых побывают монстры, идущие по горизонтали (обозначим количество таких клеток как H), количество клеток, на которых побывают монстры, идущие по вертикали (обозначим их количество как V), и количество клеток, на которых побывают оба типа монстров (обозначим их количество как B). Тогда ответом на задачу будет число $H + V - B$.

Предположим, что на столе имеются только монстры, идущие по горизонтали (то есть на запад или восток). Поставим себе целью заменить этих монстров на других таким образом, чтобы никакие два монстра не проходили по одной и той же клетке, а клетки, на которых они побывают, не изменились. Сделаем это независимо для всех горизонтальных рядов. Рассмотрим какой-либо горизонтальный ряд. Если на нем нет монстров, то, очевидно, на нем нет клеток, по которым проходят монстры, идущие по горизонтали.

Пусть на рассматриваемом ряду изначально стоят несколько монстров (напомним, что мы рассматриваем только монстров, которые идут по горизонтали). Выберем самого западного монстра, который идет на восток (если такие есть) и самого восточного монстра, который идет на запад. Всех остальных монстров, находящихся на этой горизонтали, можно удалить со стола: по всем клеткам, по которым они проходят, пройдут и выбранные крайние монстры. Если монстров, идущих в одну из сторон, нет, то выбранный один монстр проходит по тем же клеткам, что и все монстры на этом горизонтальном ряду, таким образом, для данного ряда наша цель выполнена. Если же имелись монстры, идущие в разные стороны, то выбранные два монстра могут либо не проходить по одной и той же клетке (см. рисунок 2), либо проходить (см. рисунок 3).

В первом случае оставим этих двух монстров на поле. Во втором случае на всех клетках выбранного ряда побывают монстры. Заменим их всех на одного монстра, который идет, к примеру, на восток из самой западной клетки (см. рисунок 4).

Аналогичную операцию проведем для всех вертикальных рядов, только теперь будем выбирать самого северного монстра, идущего на юг, и самого южного монстра, идущего на север, соответственно.

Теперь по каждой клетке проходит не более одного монстра, идущего по горизонтали, и не более одного монстра, идущего во вертикали. Найти числа H и V теперь не составляет труда, найдем число B. Для этого переберем все пары горизонтальных и вертикальных монстров, и, если есть клетка, по которой проходят оба монстра пары, увеличим числа B на один.

Зная H, V и B, мы можем по формуле, указанной в начале разбора, найти ответ.

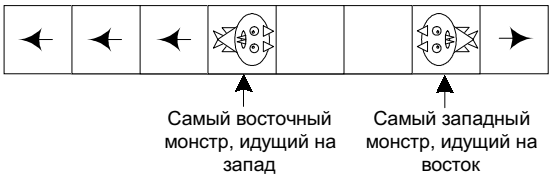


Рисунок 2.

Рисунок 3.

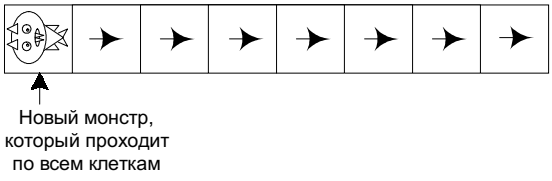


Рисунок 4.

ЗАДАЧА B. ФОРУМ

Клуб Юных Хакеров организовал на своем сайте форум. Форум имеет следующую структуру: каждое сообщение либо начинает новую тему, либо является ответом на какое-либо предыдущее сообщение и принадлежит той же теме.

После нескольких месяцев использования своего форума юных хакеров заинтересовал вопрос, какая тема на их форуме наиболее популярна. Помогите им выяснить это.

Формат входных данных

Первая строка входного файла содержит целое число N – количество сообщений в форуме ($1 \le N \le 1000$). Следующие строки содержат описание сообщений в хронологическом порядке.

Описание сообщения, которое представляет собой начало новой темы, состоит из трех строк. Первая строка содержит число 0. Вторая строка содержит название темы. Длина названия не превышает 30 символов. Третья строка содержит текст сообщения.

Описание сообщения, которое является ответом на другое сообщение, состоит из двух строк. Первая строка содержит целое число – номер сообщения, ответом на которое оно является. Сообщения нумеруются, начиная с единицы. Ответ всегда появляется позже, чем сообщение, ответом на которое он является. Вторая строка содержит текст сообщения.

Длина всех сообщений не превышает 100 символов.

Формат выходных данных

Выведите в выходной файл название темы, к которой относится наибольшее количество сообщений. Если таких тем несколько, то выведите первую в хронологическом порядке.

Пример

forum.in	forum.out
7 0 Олимпиада по информатике Скоро третья командная олимпиада? 0 Новая компьютерная игра Вышла новая крутая игра! 1 Она пройдет 24 ноября 1 В Санкт-Петербурге и Барнауле 2 Где найти? 4 Примет участие более 50 команд 6 Интересно, какие будут задачи	Олимпиада по информатике

Решение

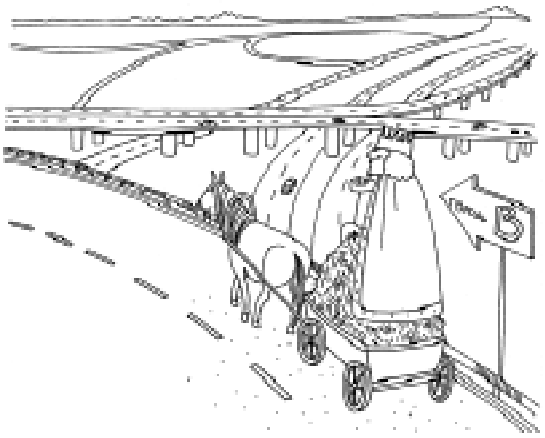
Это, пожалуй, самая простая задача на олимпиаде. Будем решать ее следующим образом. Для каждого сообщения будем хранить его тему. Для каждой темы будем хранить количество сообщений в ней и ее название. Разумеется, текст самих сообщений можно просто игнорировать.

При чтении очередного сообщения сначала читаем номер сообщения, ответом на которое оно является. Если такого нет (первое число в описании сообщения равно нулю), то увеличиваем на один количество тем и устанавливаем новому сообщений очередной номер темы. Количество сообщений в этой теме полагаем равным 1. В противном случае (если сообщение является ответом на другое сообщение) устанав-

ливаем значение темы равным значению темы сообщения, ответом на которое оно является, и увеличиваем на один количество сообщений в этой теме.

Затем стандартным способом находим максимум по количеству сообщений среди всех тем.

ЗАДАЧА C. ТРИ ГОРОДА



В одном государстве имеется N городов. Некоторые города соединены дорогами, причем для любых двух городов A и B выполняется следующее свойство: существует ровно один способ попасть из города A в город B, если можно перемещаться только по дорогам и не разрешается проезжать по одной и той же дороге более одного раза.

Недавно президента этой страны заинтересовал вопрос: какие три города являются наиболее удаленными друг от друга. А именно, назовем взаимной удаленностью друг от друга трех городов A, B и C минимальное количество дорог, которое необходимо использовать, чтобы доехать от A до B, затем от B до C и затем от C до A (при этом разрешается использовать одну и ту же дорогу в различных путешествиях).

Требуется найти три города, для которых взаимная удаленность друг от друга будет максимальной.

Например, для пяти городов, соединенных дорогами так, как это показано на рисунке 5, три наиболее удаленных друг от друга города – это города 1, 2 и 5 (взаимная удаленность равна 2 + 3 + 3 = 8), а для городов на рисунке 6 – это любые три города,

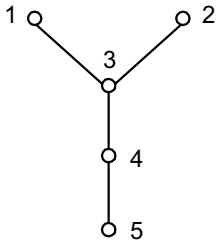
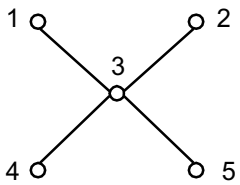


Рисунок 5.
Пример городов, соединенных дорогами, решение 1, 2, 5.

Рисунок 6.
Пример городов, соединенных дорогами, решение неоднозначно.



выбранные из множества {1, 2, 4, 5} (удаленность 2 + 2 + 2 = 6).

Формат входных данных

Первая строка входного файла содержит число N – количество городов ($3 \le N \le 1000$). Следующие N строк содержат описания городов. Описание i-го города сначала содержит K_i – количество городов, с которыми он соединен дорогами ($1 \le K_i < N$), а затем K_i чисел – номера городов, с которыми он соединен.

Гарантируется, что входные данные корректны, то есть если есть дорога из города A в город B, то есть и дорога из города B в город A, причем для всех пар городов выполняется свойство, указанное в условии задачи.

Формат выходных данных

Выведите в выходной файл три различных числа – номера трех наиболее удаленных друг от друга городов в произвольном порядке. Если решений несколько, выведите любое из них.

Примеры

three.in	three.out
5 1 3 1 3 3 1 2 4 2 3 5 1 4	1 2 5

three.in	three.out
5 1 3 1 3 4 1 2 4 5 1 3 1 3	1 2 4

Решение

В противоположность предыдущей, эта задача одна из самых сложных на соревновании (хотя отметим, что на этой олимпиаде не было предложено ни одной «неподъемной» задачи, благодаря чему каждую задачу решила хотя бы одна команда).

Для решения этой задачи удобно применять терминологию теории графов. Напомним, что графом называется множество вершин V с введенным на нем отношением $E \subset V \times V$, которое задает множество ребер графа. В нашем случае вершинами графа будут города, а ребрами – дороги. Отметим, что свойство, сформулированное в условии задачи о том, что существует ровно один простой путь из города A в город B для любых двух городов A и B, означает, что наш граф является деревом, то есть не имеет циклов. Дальнейшее изложение решения этой задачи подразумевает базовое знакомство читателя с терминологией, применяемой в теории графов.

Переформулируем нашу задачу в терминах теории графов следующим образом: найти в дереве три вершины, сумма попарных расстояний между которыми максимальна для данного дерева.

Прежде чем перейти к решению задачи, докажем один вспомогательный результат. Любой замкнутый путь в дереве (то есть путь, начало и конец которого совпадают) проходит по каждому ребру дерева четное число раз (возможно, ни разу). Действительно, рассмотрим произвольное ребро e. Если мы удалим его из дерева, оно распадется на две компоненты связности. Обозначим множество вершин одной компоненты как U, а множество вершин другой компоненты как W. Пусть наш замкнутый путь начинается в вершине из множества U. Тогда, поскольку он должен в конце также оказаться в компоненте U, он проходит по ребру, соединяющему компоненты четное число раз, что и требовалось. В силу произвольности ребра e, это доказывает наше утверждение.

Перейдем теперь к решению нашей задачи. Для решения применим следующий прием, помогающий часто эффективно организовать поиск решения. Предположим, что решение найдено и три вершины, наиболее удаленные друг от друга, – это A, B и C. Рассмотрим простые пути от A к B, от B к C и от C к A. Покажем, что у них есть ровно одна общая вершина. Будем действовать от противного.

Пусть у всех трех путей более одной общей вершины. Выберем среди них две, обозначим их как D и E. По условию, существует ровно один простой путь от D до E. Выберем на нем какое-либо ребро e. Поскольку каждый из трех рассматриваемых путей проходит и через D и через E, то каждый из них проходит по e. Но поскольку весь циклический путь $A \to B \to C \to A$ должен проходить по этому ребру четное число раз, то какой-либо из простых путей $A \to B$, $B \to C$ и $C \to A$ должен проходить по нему хотя бы два раза, что противоречит тому, что этот путь простой. Таким образом, у этих трех путей не более одной общей вершины.

Пусть теперь у них нет общих вершин. Но тогда рассмотрим наиболее удаленную от вершины A общую вершину путей $A \to B$ и $A \to C$. Поскольку она не лежит на пути $B \to C$, ни одно из следующих за ней ребер на путях к B и к C не принадлежит этому пути. Но тогда по каждому из этих ребер при проходе по циклу $A \to B \to C \to A$ мы пройдем ровно один раз, что противоречит доказанному нами выше результату о четности количества проходов по ребру на циклическом пути (рисунок 7).

Итак, у трех указанных путей имеется ровно одна общая вершина (см. рисунок 8).

Будем перебирать все вершины графа, полагая их общими для трех путей между наиболее удаленными вершинами. Пусть мы выбрали центральную вершину, обозначим

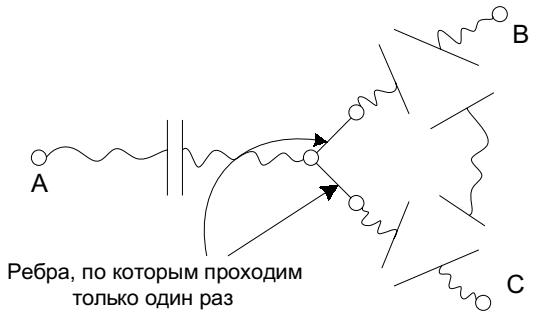


Рисунок 7.

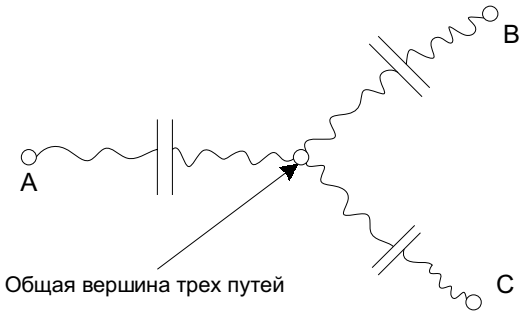


Рисунок 8.

ее как *K*. Подвесим наше дерево за эту вершину, то есть ориентируем все ребра графа по направлению от этой вершины. Тогда сумма путей между тремя вершинами *A*, *B* и *C*, находящимися в разных поддеревьях, равна $2(AK + BK + CK)$, где за *XY* обозначено расстояние между вершинами *X* и *Y*. А значит, получить три наиболее удаленные вершины в дереве в предположении, что общей вершиной является вершина *K*, можно, взяв в каждом поддереве наиболее удаленную вершину и выбрав среди них три наиболее удаленных (рисунок 9).

Если при подвешивании дерева за вершину у нее оказывается одно поддерево, то это лист, и ее можно проигнорировать (она не может быть центральной точкой трех путей), а если только два, то можно в качестве третьей вершины взять саму вершину *K* (рисунок 10).

Для каждого поддерева найти в нем наиболее удаленную вершину можно за линейное относительно размера этого дерева время. Для этого достаточно организовать поиск в глубину и устанавливать расстояние до вершины равным глубине рекурсии.

После этого найти три максимальных элемента можно за линейное время. А можно сделать проще, отсортировав массив наиболее удаленных вершин, причем можно использовать даже квадратичную сортировку. При этом если обозначить степень *i*-ой вершины как d_i, а количество ребер в графе как *E*, то суммарное время работы этой части алгоритма по всем вершинам, за которые осуществляется подвешивание, будет равно $O\left(\sum_i d_i^2\right) = O\left(E^2 \sum_i \frac{d_i^2}{E^2}\right)$.

Поскольку $d_i \le E$, то $\frac{d_i^2}{E^2} \le \frac{d_i}{E}$, значит, время работы можно оценить как

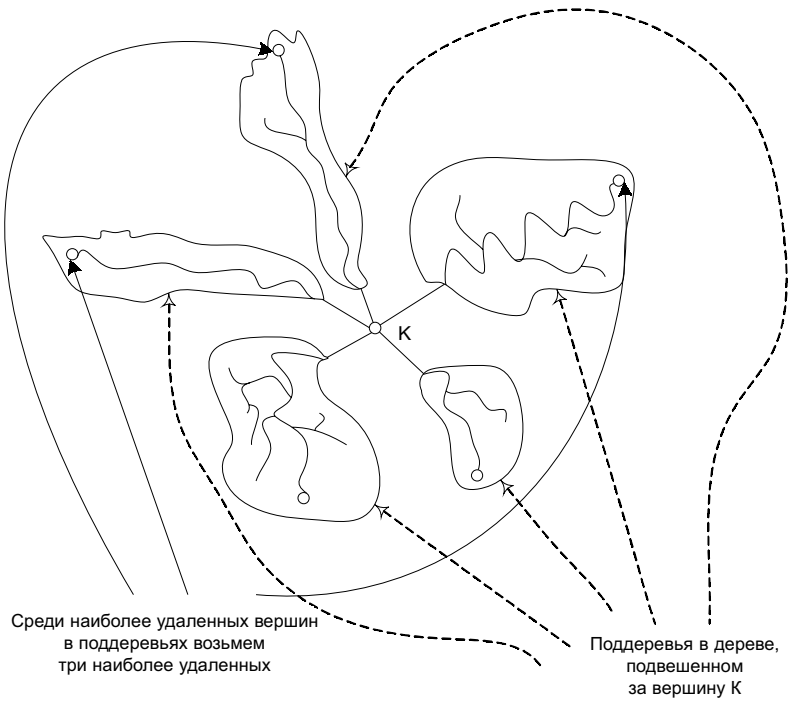
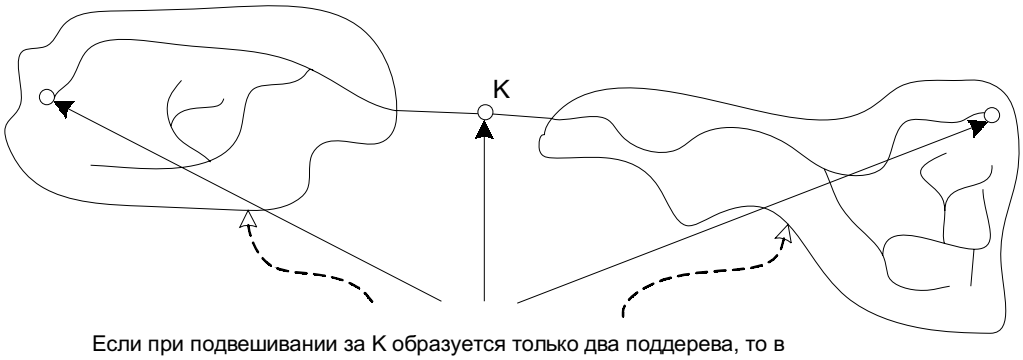


Рисунок 9.



Если при подвешивании за K образуется только два поддерева, то в качестве третьей вершины следует взять саму вершину K

Рисунок 10.

$O\left(E^2\sum_i \frac{d_i^2}{E^2}\right)=O\left(E^2\sum_i \frac{d_i}{E}\right)=O\left(E^2\right)$, а так как $E=N-1=O(N)$, то время работы этой части будет $O\left(N^2\right)$. В свою очередь, время работы поиска наиболее удаленной вершины в поддереве работает в сумме за время, равное $O(E)=O(N)$ для каждой вершины, за которую осуществляется подвешивание, и, следовательно, общее время работы алгоритма есть $O\left(N^2\right)$.

ЗАДАЧА D. ПОЛОВИННОЕ ДЕЛЕНИЕ

Рассмотрим выпуклый многоугольник, вершины которого лежат в точках плоскости с целыми координатами. Требуется разбить его на треугольники с вершинами в точках с целыми координатами, каждый из которых имел бы площадь 1/2 (рисунок 11), либо выяснить, что это сделать невозможно.

Формат входных данных

Первая строка входного файла содержит число N – количество вершин многоугольника ($1 \le N \le 10$). Следующие N строк содержат координаты вершин многоугольника в порядке обхода их по часовой стрелке. Все координаты – целые неотрицательные числа, не превышающие 10. Никакие три последовательные вершины многоугольника не лежат на одной прямой.

Формат выходных данных

Если выполнить разбиение указанным образом невозможно, выведите в выходной файл единственное число 0.

В противном случае выведите несколько строк, содержащих по 6 чисел каждая. Количество строк должно быть равно количеству треугольников в найденном разбиении. Числа в каждой строке должны представлять собой координаты вершин соответствующего треугольника $x_1, y_1, x_2, y_2, x_3, y_3$. Площадь каждого треугольника должна быть 1/2. Порядок перечисления треугольников и вершин в каждом из треугольников может быть произвольным. Если допустимых разбиений несколько, выведите любое.

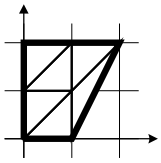


Рисунок 11. Разбиение многоугольника на треугольники с площадью 1/2.

Пример

half.in	half.out
4 0 0 0 2 2 2 1 0	0 0 1 1 1 0 0 0 1 1 0 1 0 1 1 1 1 2 0 1 1 2 0 2 1 0 2 2 1 1 1 1 2 2 1 2

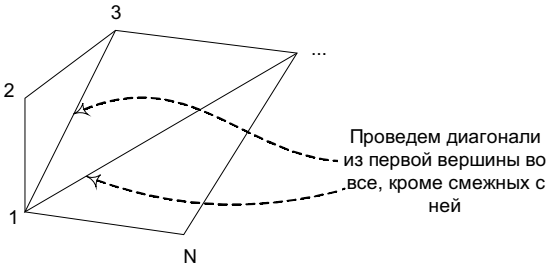


Рисунок 12.

Решение

Будем для краткости называть точки с целыми координатами *целыми точками*. Для решения этой задачи полезно вспомнить формулу Пика для многоугольника с вершинами в целых точках, связывающую количество целых точек внутри многоугольника (I), на его границе (B) и его площадь (S). В соответствии с этой формулой $I + B/2 = S + 1$.

Применим эту формулу для треугольника с вершинами в целых точках. Предположим, что его площадь равна 1/2. Тогда из этого следует, что $I + B/2 = 3/2$, а поскольку $B \geq 3$ (так как вершины треугольника являются целыми точками и лежат на его границе), то $I = 0$ и $B = 3$. С другой стороны, если $I = 0$ и $B = 3$, то площадь треугольника по формуле Пика равна 1/2. Итак, мы доказали следующее утверждение: чтобы площадь треугольника с вершинами в целых точках была равна 1/2, необходимо и достаточно, чтобы внутри него не было целых точек и на его границе не было целых точек, отличных от его вершин.

Таким образом, нам требуется разбить многоугольник на такие треугольники, что внутри и на границе каждого треугольника нет целых точек, отличных от его вершин. Сделаем это постепенно. Начнем с того, что триангулируем наш многоугольник. Триангулировать выпуклый многоугольник несложно: достаточно провести диагонали из одной вершины во все остальные, кроме смежных с ней (рисунок 12).

Ясно, что если мы разделим каждый из получившихся треугольников требуемым образом и затем объединим разбиения, то получится как раз разбиение исходного многоугольника.

Рассмотрим теперь процесс разбиения треугольника. Если внутри треугольника нет точек с целыми координатами и таких точек, отличных от его вершин, нет на его границе, то этот треугольник уже имеет площадь 1/2, его можно вывести в выходной файл (рисунок 13).

В противном случае возьмем какую-либо целую точку внутри или на границе и разобъем треугольник на три или на два новых треугольника (рисунок 14).

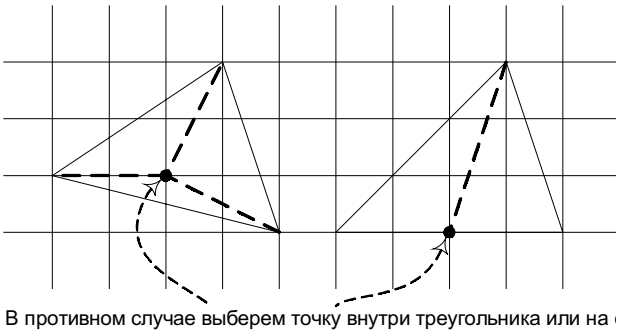


Рисунок 13.

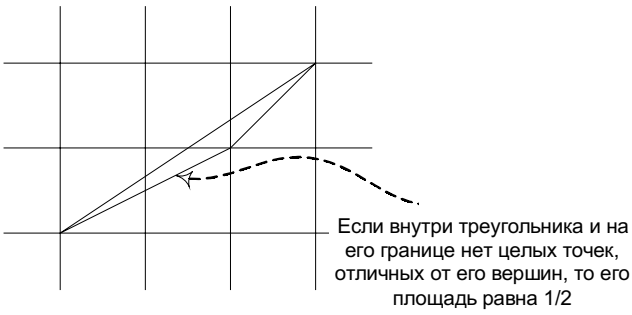
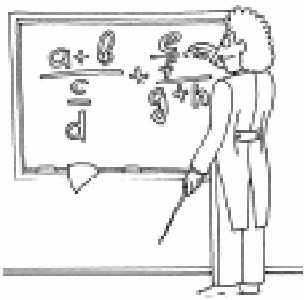


Рисунок 14.

При этом удобно всегда разбивать треугольник на три части и затем откидывать треугольники, площадь которых равна 0. Поскольку для каждого треугольника с вершинами в целых точках выполняется одно из двух: либо он имеет площадь 1/2, либо его можно разбить на треугольники меньшей площади с вершинами в целых точках, то задача всегда имеет решение.

ЗАДАЧА Е. ДРОБЬ В LATEX-Е



Издательская система $LaTeX$ предназначена для верстки сложных научно-технических текстов с большим количеством формул. Исходный файл для системы $LaTeX$ пишется на языке TeX и представляет собой текст документа, в который включены специальные символы и команды. Специальные символы и команды описывают размещение текста, в частности, в математических формулах. Команда представляет собой последовательность латинских букв (регистр важен), перед которой стоит символ \. Так, команда **\frac** предназначена для описания дроби, в которой числитель расположен над знаменателем. Рассмотрим простейшую структуру команды **\frac**.

Комадна **\frac** имеет два параметра: числитель и знаменатель. Перед самой командой не обязательно ставить пробел. Следом за ключевым словом **\frac** записываются числитель и знаменатель. Если числитель и знаменатель имеют длину более одного символа, они заключаются в фигурные скобки. Если числитель или знаменатель записываются одной буквой или цифрой, их можно не брать в фигурные скобки. Если числитель записывается одним символом, то он отделяется от **\frac** хотя бы одним пробелом. Если знаменатель записывается одним символом, то он не отделяется пробелом от числителя. Произвольное ненулевое количество пробелов считается синтаксически эквивалентным одному пробелу. Нельзя разделять пробелами на части ключевое слово **\frac**.

Дадим также формальное определение выражения для нашей задачи (см. рисунок 15).

Здесь вертикальная черта | означает «или», заключенная в квадратные скобки часть может отсутствовать, а символы, записаные в кавычках, обозначают самих себя. Печатный символ – любой символ с ASCII кодом от 32 (пробел) до 127.

Например, выражение

$$\frac{a+b}{d+1}+\frac{a}{x}-\frac{2}{2+\frac{3}{y}}$$

<выражение> ::= <элемент> | <элемент><выражение>
<элемент> ::= <дробь> | { <выражение> } | <другой математический элемент>
<дробь> ::= “**\frac**” <тело дроби>
<тело дроби> ::= <числитель><знаменатель>
<числитель> ::= <пробелы><непробельный символ> | [<пробелы>] “{” <выражение> “}”
<знаменатель> ::= <непробельный символ> | [<пробелы>] “{” <выражение> “}”
<другой математический элемент> ::= *произвольная последовательность печатных символов, не содержащая фигурных скобок и подстроки* \frac
<пробелы> ::= “ ” | “ ” <пробелы>
<непробельный символ> ::= *произвольный печатный символ, за исключением* “ ”, “\”, “{” и “}”

Рисунок 15.

записывается на языке TEX как

```
\frac{a+b}{d+1}+\frac ax -\frac 2 {2+\frac{3}{y}}
```

Чтобы в печатаемом документе вывести формулу, необходимо вычислить ее высоту для используемого при печати шрифта. Шрифт определяет размеры S – высоту отдельного символа и D – высоту горизонтальной дробной черты. Значения S и D задаются целыми числами. Ваша задача – для заданного выражения на языке T_EX вычислить высоту формулы.

Отметим, что если две дроби принадлежат одному выражению, то их дробные черты записываются на одном уровне, а если нет (например, относятся к числителям или знаменателям различных дробей), – это свойство может и не выполняться. Чтобы проиллюстрировать применение этого правила, приведем два примера:

```
\frac{a+b}{\frac cd}+\frac{\frac ef}{g+h}
```

$$\frac{a+b}{\frac{c}{d}}+\frac{\frac{e}{f}}{g+h}$$

```
\frac{a+b+c}{\frac{\frac de}{g+h}}+\frac{i+j+k}{\frac{l+m}{\frac no}}
```

$$\frac{a+b+c}{\frac{\frac{d}{e}}{g+h}}+\frac{i+j+k}{\frac{l+m}{\frac{n}{o}}}$$

Формат входных данных

В первой строке находятся целые положительные числа S и D ($1 \le S, D \le 10000$). Следующая строка содержит описание формулы на T_EX-е, длина строки не более 200 символов. Гарантируется, что формула синтаксически корректна, то есть фигурные скобки образуют правильную скобочную последовательность и строка содержит только печатные символы. Все символы `\`, встречающиеся в строке, относятся к некоторой командной последовательности (не обязательно **`\frac`**). Можете считать, что все прочие командные последовательности задают символы, высота которых равна S. Числитель и знаменатель каждой дроби содержат хотя бы по одному символу, вся формула содержит хотя бы один символ.

Формат выходных данных

Выведите в выходной файл единственное число – высоту формулы.

Примеры (см. рисунок 16)

Решение

Прежде всего отметим, что на самом деле указанная схема не применяется напрямую в системе T_EX. Во первых, в T_EX-е числитель и знаменатель дроби имеют меньший размер, чем основной текст, числитель и знаменатель дроби, находящейся в числителе или знаменателе другой дроби, имеют еще меньший размер и т. д. Кроме того, в T_EX-е используется специальная система определения размеров промежутков между

frac.in	frac.out
10 2 \frac{a+b}{d+1}+\frac ax -\frac 2 {2+\frac{3}{y}}	34
10 2 no fractions here	10
10 2 \frac {\alpha} {\beta+\sin{2+x}}	22
10 2 \cos{\frac{\alpha}b}	22
10 2 \frac a {sin{a}}	22
10 2 \frac{a+b}{\frac cd}+\frac{\frac ef}{g+h}	46
10 2 \frac{a+b+c}{\frac{\frac de}{g+h}}+\frac{i+j+k}{\frac{l+m}{\frac no}}	46

Рисунок 16.

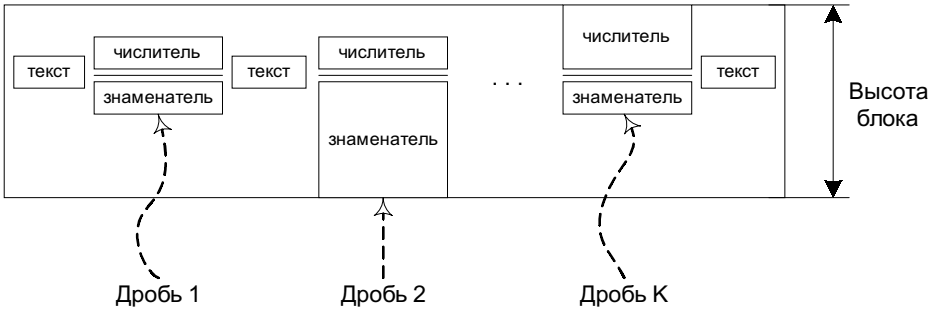


Рисунок 17.

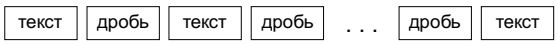
элементами в математической формуле, учитывающая в том числе высоту отдельных символов. Таким образом, указанная схема не соответствует применяемой в T_EX-е на самом деле. Система, похожая на изложенную, используется в системах визуального набора формул *Microsoft Equation* и *Math Type*.

Перейдем теперь к решению нашей задачи.

При решении основную проблему представляет чтение ввода и выделение дробей. Отметим, что поскольку все прочие символы имеют одинаковую высоту, смысл прочих кодовых последовательностей не важен, а числитель и знаменатель не могут быть пустыми, то нет необходимости писать лексический анализ текста. Можно решить задачу проще.

Построим функцию, которая будет возвращать высоту математического выражения, записанного в строке с i-го по j-ый символ, при этом будем считать, что этот фрагмент выражения не пересекает границ описания никакой дроби.

Найдем высоту блока, который соответствует указанному выражению. Выделим все дроби в этом выражении. Тем самым мы представим наш блок в виде последовательности блоков:



Если в блоке нет дробей, то он состоит из единственного блока, его высота равна S. В противном случае для каждой дроби найдем высоту ее числителя и знаменателя. Высотой блока будет $D + \max H(num) + \max H(den)$, где $H(num)$ – высота числителя дроби, $H(den)$ – высота знаменателя дроби и максимум в обоих случаях берется по всем дробям в выражении. Высота же числителей и знаменателей вычисляется рекурсивно тем же способом (см. рисунок 17).

Таким образом, процесс вычислений выглядит следующим образом (см. рисунок 18).

Находим в исходной строке первое вхождение подстроки **`\frac`**. Аккуратно выделяем числитель и знаменатель. Рекурсивно вызываемся от них, обновляем при необходимости значение максимума для высот числителей и знаменателей, соответственно. В конце, если была найдена дробь,

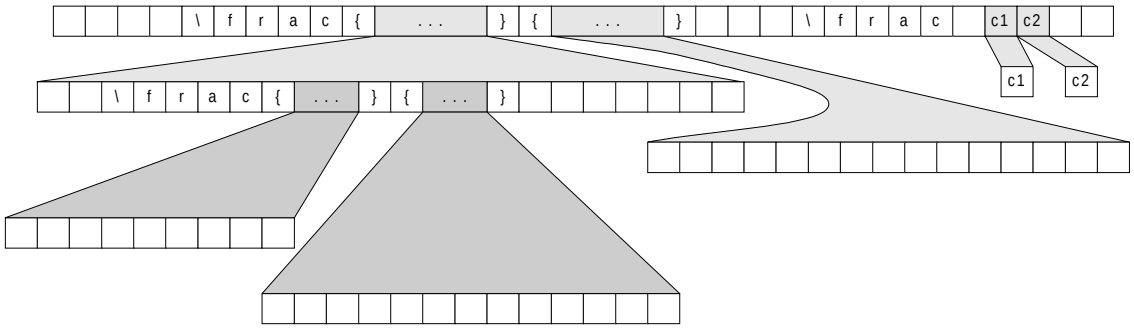
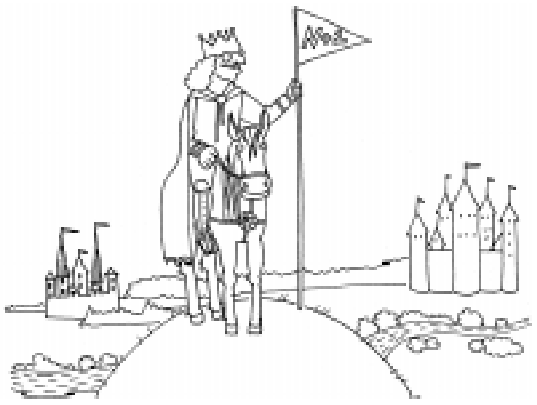


Рисунок 18.

возвращаем сумму найденных максимумов плюс *D*, иначе возвращаем *S*.

ЗАДАЧА F. СТОЛИЦА



В некотором царстве, в некотором государстве было *N* городов, и все они, судя по главной карте императора, имели целые координаты. В те годы леса были дремучие, дороги же строить умели только параллельно осям координат, так что расстояние между двумя городами определялось как $|x_1 - x_2| + |y_1 - y_2|$.

Император решил построить *N* + 1-ый город и сделать его столицей своего государства, при этом координаты столицы также должны быть целыми. Место для столицы следует выбрать так, чтобы среднее арифметическое расстояний между столицей и остальными городами было как можно меньше. Однако, разумеется, столицу нельзя строить на месте существующего города.

Нелегкая задача выбрать место для столицы поручена Вам.

Формат входных данных

Первая строка входного файла содержит число *N* количество городов ($1 \le N \le 100$). Следующие *N* строк содержат координаты городов – пары целых чисел, не превышающих 1000 по абсолютной величине.

Формат выходных данных

Выведите в выходной файл два целых числа – координаты точки, где следует построить столицу. Если решений несколько, выведите любое.

Примеры

capital.in	capital.out
8 0 0 1 0 2 0 0 1 2 1 0 2 1 2 2 2	1 1
4 0 0 1 1 0 1 1 0	0 2

Решение

Для начала предположим, что у нас нет ограничения на строительство столицы в точке, в которой уже имеется другой город.

Для минимизации среднего арифметического достаточно минимизировать сумму. Таким образом, нам требуется минимизировать функцию $d(x_c, y_c) = \sum_{i=1}^{N} \left(|x_c - x_i| + |y_c - y_i| \right)$, где (x_c, y_c) – координаты столицы. Заметим, что эту функцию можно представить в виде суммы двух независимых функций от одной переменной: $d_x(x_c) = \sum_{i=1}^{N} |x_c - x_i|$ и $d_y(y_c) = \sum_{i=1}^{N} |y_c - y_i|$. Таким образом, можно независимо минимизировать обе эти функции, и тогда их сумма также будет минимальна.

Посмотрим, как минимизировать функцию $\sum_{i=1}^{N} |x_c - x_i|$. Отсортируем координаты x_i по возрастанию и рассмотрим пары $x_1 \leftrightarrow x_N$, $x_2 \leftrightarrow x_{N-1}$, ..., $x_{\lfloor (N+1)/2 \rfloor} \leftrightarrow x_{\lceil (N+1)/2 \rceil}$. В случае, если *N* нечетно, элементы в последней паре совпадают (см. рисунок 19).

Разобьем теперь слагаемые в нашей сумме на пары в соответствии с выбранным нами разбиением на пары координат. В случае нечетного *N* одно из слагаемых останется без пары.

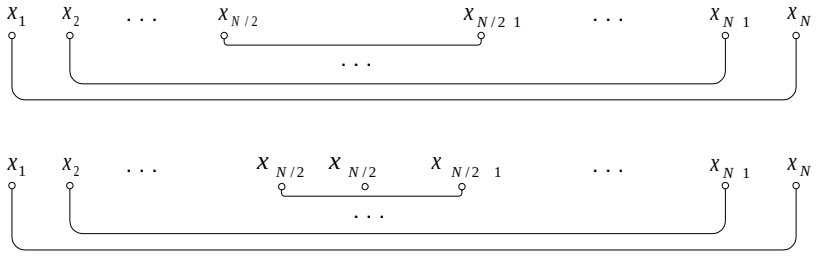


Рисунок 19.

Пусть N четно.

Тогда $\sum_{i=1}^{N}|x_c - x_i| = \sum_{i=1}^{N/2}\left(|x_c - x_i| + |x_c - x_{N+1-i}|\right)$. Каждое слагаемое в сумме представляет собой сумму расстояний от некоторой точки на прямой до двух заданных точек. Заметим, что если точка x_c лежит между точками x_i и x_{N+1-i}, то это слагаемое равно $|x_i - x_{N+1-i}|$. Если же это не так, то $|x_c - x_i| + |x_c - x_{N+1-i}| > |x_i - x_{N+1-i}|$. Заметим также, что поскольку мы отсортировали координаты, то для рассматриваемых отрезков между парными точками выполняется включение $[x_1, x_N] \supset [x_2, x_{N-1}] \supset \ldots \supset [x_{N/2}, x_{N/2+1}]$, и, следовательно, все точки отрезка $[x_{N/2}, x_{N/2+1}]$ принадлежат также всем остальным отрезкам, и для этих точек искомая сумма имеет одно и то же значение. В то же время в случае, если точка не принадлежит хотя бы одному отрезку, сумма окажется больше. Следовательно, сумма имеет минимальное значение в случае, когда координата x_c находится между $N/2$ и $N/2 + 1$ координатой городов.

В случае нечетного N приведенное рассуждение остается в силе, за исключением того, что необходимо независимо рассматривать среднюю координату $x_{\lceil N/2 \rceil}$. Слагаемое с этой координатой не имеет пары. Оно равно нулю в случае, если $x_c = x_{\lceil N/2 \rceil}$. Эта точка также лежит на всех отрезках $[x_1, x_N] \supset [x_2, x_{N-1}] \supset \ldots \supset \left[x_{(N-1)/2}, x_{(N+1)/2+1}\right]$, а значит, в этом случае достигается минимум нашей суммы.

Объединяя результаты для обоих случаев, получаем следующее утверждение: $d_x(x_c)$ минимально, если $x_{\lceil N/2 \rceil} \le x_c \le x_{\lfloor N/2 \rfloor + 1}$ (отметим, что в случае нечетного N отрезок решений этого неравенства вырождается в точку, он может выродиться в точку в случае, если некоторые координаты городов по оси x совпадают).

Аналогично получаем, что оптимальная координата по оси y должна удовлетворять неравенству $y_{\lceil N/2 \rceil} \le y_c \le y_{\lfloor N/2 \rfloor + 1}$.

Таким образом, оптимальные решения задачи без дополнительного ограничения на строительство столицы в точке, уже занятой городом, лежат в некотором прямоугольнике (рисунок 20). Назовем его *оптимальным прямоугольником*.

Вернемся теперь к исходной задаче. Имеется следующее препятствие к непосредственному применению полученного результата к нашей задаче: выбрав произвольную точку, координаты которой удовлетворяют указанным неравенствам, мы можем выбрать уже существующий город. Мы могли бы перебрать все точки оптимального прямоугольника, но это не поможет: во-первых, он может иметь большую площадь (хотя в этой задаче координаты ограничены небольшой величиной и такое решение могло бы уложиться в отведенный период времени), но более важно то, что *все* точки прямоугольника могут уже быть заняты городами.

Интуиция подсказывает, что следует перемещать место для столицы, пока оно не окажется в точке, где нет города. Однако непонятно, в какую сторону следует осуществлять перемещение и почему найденное таким образом место будет оптимально. Можно было бы проверить все точки в окрестно-

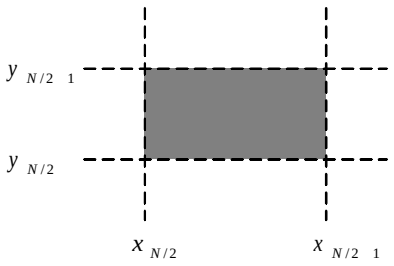


Рисунок 20.

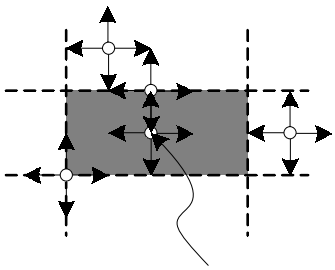


Рисунок 21.

сти оптимального прямоугольника и выбрать среди них наилучшую. Однако непонятно, сколько точек придется проверить.

Оказывается, достаточно проверить всего $O(N)$ точек. Докажем следующее утверждение: для того чтобы выбрать оптимальное место для столицы, достаточно проверить одну произвольную точку оптимального прямоугольника и четыре соседних точки рядом с каждым городом (см. рисунок 21). При этом если точка, которую мы проверяем, занята городом, то можно ее просто пропустить.

Действительно, рассмотрим три случая.

Случай 1: выбранная нами точка оптимального прямоугольника не занята городом. В этом случае, как уже было показано, она оптимальна.

Случай 2: выбранная нами точка оптимального прямоугольника занята городом, но в оптимальном прямоугольнике имеется хотя бы одна не занятая точка. В таком случае, очевидно, хотя бы одна такая точка лежит рядом с каким-либо городом.

Случай 3: в оптимальном прямоугольнике нет свободных точек. Заметим, что если точка не лежит в оптимальном прямоугольнике, то соседняя с ней точка, у которой одна из координат ближе к оптимальному прямоугольнику, имеет меньшую искомую сумму (поскольку по этой координате она приближается к большему количеству городов, чем удаляется). Рассмотрим оптимальную точку (точку, имеющую минимум искомой суммы из незанятых городами). Поскольку она не лежит в оптимальном прямоугольнике, то, по крайней мере, в одной из четырех соседних с ней точек искомая сумма меньше. Но это и означает, что та точка занята городом и оптимальная точка является соседней с ним.

ЗАДАЧА G. ПЕРЕСТАНОВКИ

Ваня и Петя играют в следующую игру. Ваня пишет на бумаге какую-либо перестановку чисел от 1 до N (то есть выписывает все числа от 1 до N в некотором порядке) и расставляет на столе в ряд N предметов. После этого Петя переставляет предметы в соответствии с ваниной перестановкой. А именно, Петя выполняет следующие действия: если i-ое число в ваниной перестановке равно a_i, то Петя ставит предмет, который стоит на i-ом месте, на место с номером a_i.

Обозначим предметы числами от 1 до N. Тогда начальное расположение предметов можно обозначить последовательностью чисел (1, 2, ..., N). К примеру, если N = 5, то начальное расположение предметов есть (1, 2, 3, 4, 5). Пусть Ваня написал перестановку <2, 5, 4, 3, 1>. Это значит, что после перемещения предметов они окажутся расставлены в следующем порядке: (5, 1, 4, 3, 2).

Однако, переставив предметы, Петя не останавливается на достигнутом и вновь переставляет их в соответствии с ваниной перестановкой. Так, если в приведенном выше примере повторно применить перестановку, предметы окажутся расположены в следующем порядке: (2, 5, 3, 4, 1).

Таким образом, Петя переставляет предметы в соответствии с ваниной перестановкой, пока их расположение не окажется таким же, как исходное. В нашем примере Пете потребуется сделать еще 4 действия, порядок предметов после каждого из них будет следующим: (1, 2, 4, 3, 5), (5, 1, 3, 4, 2), (2, 5, 4, 3, 1), (1, 2, 3, 4, 5). Всего Пете потребовалось применить перестановку 6 раз.

Добрый Ваня хочет, чтобы Пете пришлось выполнить как можно больше действий. Помогите ему выбрать соответствующую перестановку.

Формат входных данных

Входной файл содержит единственное целое число N – количество предметов ($1 \le N \le 100$).

Формат выходных данных

Выведите в выходной файл перестановку чисел от 1 до N, при которой количество действий, которое придется сделать Пете, максимально. Если таких перестановок несколько, можно вывести любую.

Пример

perm.in	perm.out
5	2 5 4 3 1

Решение

Выясним сначала, как по перестановке узнать, сколько действий потребуется совершить Пете, прежде чем он снова получит исходное расположение предметов. Это число называется *порядком* перестановки. Для этого построим ориентированный граф с N вершинами и N дугами. Вершины будут обозначать позиции предметов. Проведем из i-ой вершины в j-ую дугу, если $a_i = j$.

Таким образом, мы может считать, что изначально i-ый предмет размещен в i-ой вершине графа, и каждый раз, когда Петя производит перемещение предметов, перемещается по дуге, выходящей из вершины, в которой он находится (ясно, что в графе могут быть петли, и предмет может оставаться на месте, это соответствует случаю когда $a_i = i$).

Заметим, что в нашем графе из каждой вершины выходит и входит ровно одна дуга, а значит, граф представляет собой набор циклов. Если длина некоторого цикла есть L, то через каждые L операций по перемещению предметы в этом цикле оказываются на своем месте. Значит, если мы найдем циклы нашего графа (они также называются циклами данной перестановки), то мы сможем найти минимальное число действий, которое потребуется совершить Пете. Это будет наименьшее число, которое делится на длины всех циклов, то есть наименьшее общее кратное длин всех циклов.

Итак, порядок перестановки равен наименьшему общему кратному длин циклов, на которые распадается эта перестановка.

В приведенном примере перестановка распадается на два цикла: (2, 5, 1) и (4, 3), их длины равны 3 и 2 соответственно, значит, ее порядок равен 6.

Заметим, что сумма длин всех циклов перестановки равна количеству элементов в ней, и если заданы числа $L_1, L_2, ..., L_k$, такие что $L_1 + L_2 + ... + L_k = N$, то можно построить перестановку N чисел такую, что в ней будет k циклов с длинами $L_1, L_2, ..., L_k$, соответственно. Это перестановка $(2, 3, ..., L_1, 1)(L_1 + 2, L_1 + 3, ..., L_1 + L_2, L_1 + 1)...(L_1 + ... + L_{k-1} + 2, ..., N, L_1 + ... + + L_{k-1} + 1)$.

Таким образом, нашу задачу можно переформулировать следующим образом. Задано число N, найти такие числа, чтобы их сумма была равна N и их наименьшее общее кратное было максимальным.

Установим сначала некоторые факты относительно оптимального решения.

Пусть мы знаем решение для $M < N$. Тогда мы можем превратить его в решение для N, добавив необходимое количество единиц. Таким образом, целевая функция (оптимальное наименьшее общее кратное) не убывает.

Покажем, что существует оптимальное решение, в котором все числа попарно взаимно просты. Действительно, пусть в

решении наибольший общий делитель L_i и L_j равен $d > 1$. Тогда заменим число L_j на $L_j / d < L_j$. Наименьшее общее кратное чисел не изменится, а сумма уменьшится и, следовательно, может быть дополнена до N единицами. Продолжая этот процесс, получим решение, которое имеет такое же значение НОК, но в котором все числа попарно взаимно просты.

Заметим, что в случае попарно взаимно простых слагаемых их наименьшее общее кратное равно их произведению.

Покажем, что в качестве слагаемых достаточно брать только степени простых чисел. Действительно, пусть в оптимальном решении присутствует число $L_i = AB$, где $A > 1$, $B > 1$ и наибольший общий делитель A и B равен 1. Поскольку если $A \ge 2$ и $B \ge 2$, то $AB \ge A + B$, можно заменить число L_i на два – A и B, соответственно, при этом сумма не увеличится, а произведение останется прежним. Продолжая этот процесс, получим оптимальное решение, в котором все слагаемые – степени простых чисел.

Итак, требуется разложить число N в сумму степеней различных простых чисел и единиц, чтобы их произведение было максимально.

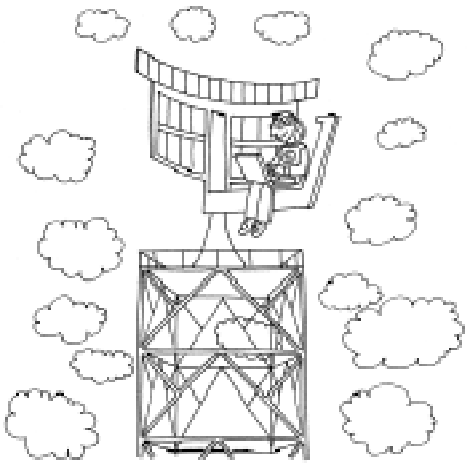
Применим для решения динамическое программирование. Обозначим через $a[i][j]$ максимальное произведение, которое может иметь набор степеней различных простых чисел, сумма которых не превышает i, и используются только первые j простых чисел. Тогда $a[i][0] = 1$ для всех i, $a[0][j] = 1$ для всех j.

Если $i > 0$, $j > 0$, то

$$a[i][j] = \max\left\{a[i][j-1], \max_k\left(a[i-k][j-1]\times k\right)\right\},$$

где k принимает все возможные значения вида $k = p_j^{\,D}$ такие, что $k \le i$, где p_j – j-ое простое число, D – целое. Искомым максимальным произведением будет $a[N][P]$, где P – количество простых чисел, не превышающих N. Чтобы восстановить необходимые слагаемые и построить саму перестановку, требуется запоминать в специальном массиве, в каком случае достигался максимум в указанной формуле.

ЗАДАЧА H. ВОЕННАЯ СОРТИРОВКА



Фирма *Macrohard* получила заказ от армии одной страны на реализацию комплекса программного обеспечения для нового суперсекретного радара. Одной из наиболее важных подпрограмм в разрабатываемом комплексе является процедура сортировки.

Однако, в отличие от обычной сортировки, эта процедура должна сортировать не произвольный массив чисел, который передается ей на вход, а специальный заранее заданный массив из N чисел, в котором записана некоторая фиксированная перестановка чисел от 1 до N, и, кроме того, ни одно число в нем изначально не находится на своем месте (то есть на позиции с номером i изначально не находится число i).

В связи с повышенными требованиями к надежности комплекса в процессе сортировки разрешается выполнять единственную операцию: менять местами два соседних элемента массива. Кроме того, в связи с необходимостью соответствия уставу, не разрешается изменять положение числа, которое уже находится на своем месте.

Например, если массив из 6 элементов в некоторый момент имеет вид <2, 1, 3, 6, 4, 5>, то можно поменять местами 1 и 2, 6 и 4 или 4 и 5, а менять местами 1 и 3 или 3 и 6 нельзя, поскольку число 3 находится на своем месте (на позиции с номером 3).

Вам дан входной массив и поставлено важное задание. Найти последовательность обменов (не обязательно кратчайшую),

сортирующую массив и удовлетворяющую приведенным условиям.

Подсказка: найти такую последовательность обменов всегда возможно.

Формат входных данных

Первая строка входного файла содержит целое число N – размер входного массива ($1 \le N \le 100$). Вторая строка содержит N целых чисел – исходную перестановку чисел от 1 до N в массиве. Изначально ни одно число не стоит на своем месте.

Формат выходных данных

Выведите в выходной файл K строк, где K – количество обменов в Вашей сортировке. На каждой строке выведите по два числа x_i и y_i, разделенных пробелом, – позиции в массиве, числа на которых следует поменять местами на i-ом обмене. Помните, что должно выполняться условие $|x_i - y_i| = 1$ и что нельзя перемещать число, которое уже стоит на своем месте.

Пример

sort.in	sort.out
6 2 3 1 6 4 5	2 3 1 2 4 5 5 6

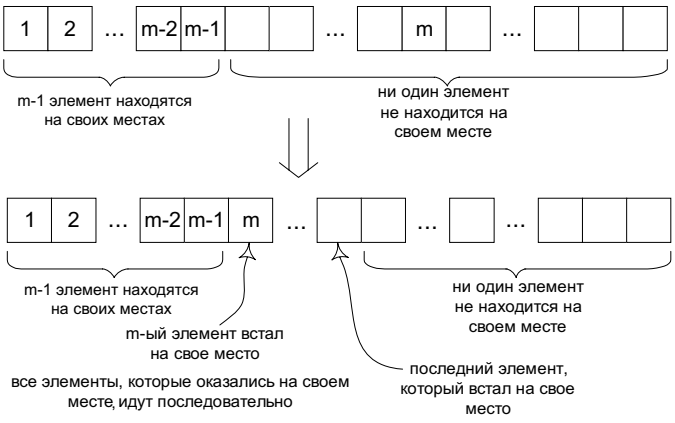


Рисунок 22.

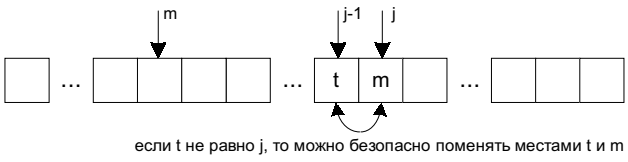


Рисунок 23.

В приведенном примере массив последовательно имеет следующий вид:

исходный вид массива	2 3 1 6 4 5
поменяли местами числа на 2 и 3 позициях	2 1 3 6 4 5
поменяли местами числа на 1 и 2 позициях	1 2 3 6 4 5
поменяли местами числа на 4 и 5 позициях	1 2 3 4 6 5
поменяли местами числа на 5 и 6 позициях	1 2 3 4 5 6

Решение

Будем устанавливать элементы массива на свои места по очереди. Пусть первые $m - 1$ числа уже находятся на своем месте. Рассмотрим оставшуюся часть массива. Предположим, что в ней ни одно число не находится на своем месте.

Рассмотрим минимальное число в оставшейся части массива. Оно равно m. Поставим себе целью поставить его на свое место таким образом, чтобы при этом не возникла ситуация, когда некоторое число оказывается на своем месте и слева от него имеются числа, которые еще не находятся на своих местах (рисунок 22).

Будем решать задачу индукцией по расстоянию от минимального числа до его места. Если минимальное число уже стоит на своей позиции в массиве, то подзадача решена.

Пусть мы умеем решать поставленную подзадачу в случае, когда ни одно число (кроме, возможно, минимального) в массиве не находится на своем месте и минимальное число находится в массиве на i-ом месте, где $m \le i < j$. Научимся решать эту задачу в случае, если минимальное число находится на j-ом месте. Если число на $j - 1$-ой позиции не равно j, то его можно поменять с минимальным числом. Поставленное нами условие не нарушится, минимальное число приблизится к своему месту, и получится задача, которую мы уже умеем решать по индукционному предположению (рисунок 23).

Пусть теперь число на позиции $j - 1$ равно j. Найдем максимальное число $k < j$ такое, что число на k-ой позиции не равно $k + 1$ (положим $k = 0$, если таких нет) (рисунок 24).

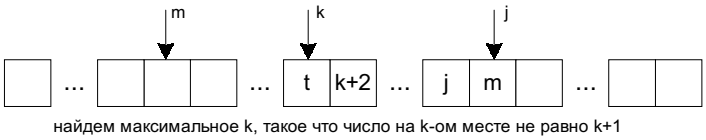


Рисунок 24.

Если $k = m - 1$ (то есть все числа, предшествующие минимальному в оставшейся части массива, находятся на одну позицию левее своего места), то просто последовательными обменами переместим число m на свое место. При этом числа с $m + 1$ по j также станут на свои места (рисунок 25).

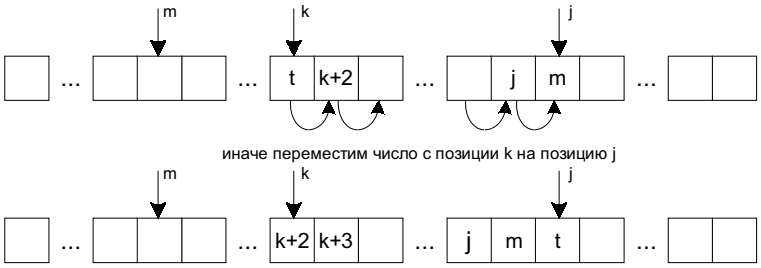


Рисунок 25.

Если же $k > m$, то переместим число, стоящее на k-ой позиции, на j-ую позицию последовательными обменами. При этом ни одно число не окажется на своем месте. После этой операции число m окажется на позиции с номером $j - 1 < j$, и, следовательно, получится задача, которую мы умеем решать по индукционному предположению (рисунок 26).

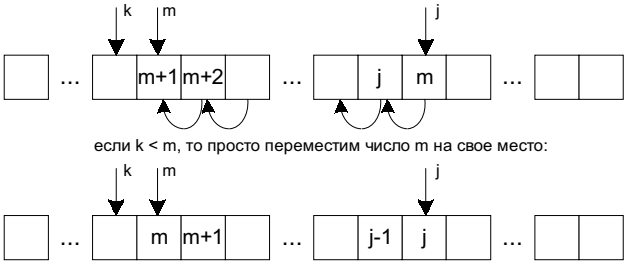


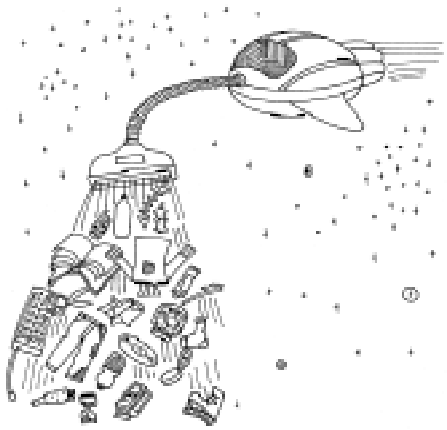
Рисунок 26.

Таким образом, мы можем последовательно поставить на свое место все числа в массиве.

Мы построили алгоритм решения задачи и параллельно доказали факт, указанный в условии в качестве подсказки – решение всегда существует.

ЗАДАЧА I. КОСМИЧЕСКИЙ МУСОРЩИК

В околоземном космическом пространстве накопилось много мусора, поэтому ученые сконструировали специальный аппарат-ловушку для космического мусора. Для того чтобы хорошо собирать мусор, этот аппарат должен двигаться по достаточно сложной траектории, сжигая собранный по пути мусор. Ловушка может передвигаться в пространстве по 6 направлениям: на север (N), на юг (S), на запад (W), на восток (E), вверх (U) и вниз (D). Движением ловушки управляет процессор. Программа движения задается шестью правилами движения, которые соответствуют каждому из указанных направлений. Каждое такое правило представляет собой строку символов из множества {N, S, W, E, U, D}.



Команда ловушки есть пара из символа направления и параметра целого по-

ложительного числа *M*. При исполнении такой команды ловушка в соответствии со своей программой выполняет следующее. Если параметр больше 1, то она перемещается на один метр в направлении, которое указано в команде, а затем последовательно выполняет команды, заданные правилом для данного направления, с параметром меньшим на 1. Если же параметр равен 1, то просто перемещается на один метр в указанном направлении.

Пусть, например, заданы следующие правила:

Направление	Правило
N	N
S	NUSDDUSE
W	UEWWD
E	
U	U
D	WED

Тогда при выполнении команды S(3) мусорщик выполнит следующие действия:

1) переместится на 1 метр в направлении S;

2) выполнит последовательно команды N(2), U(2), S(2), D(2), D(2), U(2), S(2), E(2).

Если далее проанализировать действия мусорщика при выполнении команд из пункта 2, получим, что в целом он совершит следующие перемещения: SNNUUSNUS DDUSEDWEDDWEDUUSNUSDDUSEE.

По заданной команде определите, какое общее количество перемещений на один метр совершит ловушка при выполнении заданной команды. В приведенном примере это количество равно 34.

Формат входных данных

Первые шесть строк входного файла задают правила для команд с направлением N, S, W, E, U и D соответственно. Каждая строка содержит не более 100 символов (и может быть пустой). Следующая строка содержит команду ловушки: сначала символ из множества {N, S, W, E, U, D}, затем пробел и параметр команды – целое положительное число, не превышающее 100.

Формат выходных данных

Выведите в выходной файл единственное число – количество перемещений, которое совершит ловушка. Гарантируется, что ответ не превышает 10^9.

Пример

trash.in	trash.out
N NUSDDUSE UEWWD U WED S 3	34

Замечание: Для всех задач максимальное время работы на одном тесте – 2 секунды, максимальный объем доступной памяти – 8 мегабайт.

На диске к журналу находятся фрагменты текстов программ.

Станкевич Андрей Сергеевич,
тренер сборной команды
СПб ГИТМО (ТУ)
по программированию.