Министерство образования и науки, молодежи и спорта Украины

Зимняя школа по программированию

Харьков, ХНУРЭ
2012

Оглавление

День первый (18.02.2012 г.)
Контест Akai

Об авторе...

Миланин Александр, родился 28 августа 1988 года. Окончил Таврический национальный университет им. В.И. Вернадского. Аспирант кафедры “Прикладной математики”.

Основные достижения:

В составе команды Akai занял:

- В 2009 году — 2 место на чемпионате KPI-Open в Киеве.
- В 2009 году — 6 место в полуфинале чемпионата мира региона SEERC в Бухаресте.
- В 2010 году — 3 место в финале чемпионата Украины в городе Винница.
- В 2010 году — 4 место в полуфинале чемпионата мира региона SEERC в Бухаресте.
- В 2011 году — 1 место во II этапе Всеукраинской студенческой олимпиады по программированию.
- В 2011 году — 2 место в полуфинале чемпионата мира региона SEERC в Бухаресте.
- В составе команды Akai будет представлять Украину на финале ACM ICPC в Варшаве в мае 2012 года.

Теоретический материал. Алгоритм Эдмондса о сжатии цветков.

Постановка задачи.

Пусть дан произвольный неориентированный граф без петель и кратных ребер. Требуется найти в нем максимальное по мощности подмножество ребер такое, что любая вершина графа инцидентна не более чем одному ребру из этого подмножества. Это подмножество называется паросочетанием.

Общий подход.

Если существует ребро из паросочетания, инцидентное некоторой вершине, то эта вершина называется насыщенной, иначе — свободной.

Простой путь в графе называется чередующимся, если он содержит по очереди ребра из паросочетания и не из паросочетания.

Чередующийся путь называется увеличивающим, если он начинается и заканчивается в свободных вершинах. Эти вершины различны, поскольку путь простой.

Для поиска максимального паросочетания используется основная идея, высказанная в теореме Бержа: паросочетание является максимальным тогда и только тогда, когда в графе не существует увеличивающих путей.

Краткое доказательство. В одно сторону тривиально: пусть существует увеличивающий путь, тогда можно инвертировать все его ребра, тем самым паросочетание увеличится на 1 — противоречие. В обратную сторону сложнее: допустим M — паросочетание и оно не максимальное. Тогда пусть M' — максимальное паросочетание. Множество S — симметрическая разность M и M', то есть $S = (M \cup M') \setminus (M \cap M')$. Тогда S состоит из цепочек или простых циклов четной длины, в которых ребра чередуются то из M, то из M'. Действительно, если бы был цикл нечетной длины, это бы означало, что некоторая вершина насыщена двумя ребрами одного паросочетания. Если бы была цепочка нечетной длины, это бы означало наличие увеличивающего пути, что невозможно. Тогда, поскольку для получения множества S были выброшены общие ребра из M и из M', а также тот факт, что цепочки и циклы четной длины, то S содержит одинаковое количество ребер из M и из M'. Следовательно, M также является максимальным — противоречие.

Доказательство теоремы само по себе указывает на способ поиска максимального паросочетания. Нужно находить в графе увеличивающий путь, после чего инвертировать все его ребра, увеличивая паросочетание на 1. Как только путь не найден, то паросочетание — максимальное.

Поиск увеличивающих путей.

Для поиска увеличивающего пути можно попробовать запустить поиск в глубину из свободной вершины. Обход должен идти по чередующимся ребрам из паросочетания и не из него. Если найдется свободная вершина, то это будет означать наличие увеличивающего пути. Рассмотрим подробнее. Пусть есть некоторая вершина v при обходе и мы перебираем всех ее соседей не из паросочетания. Тогда в случае, если соседняя вершина u свободная, то найден увеличивающий путь, если насыщенная, но не посещенная, то надо ее посетить и запустить обход из ее пары в паросочетании. А вот если вершина уже посещена, то есть два варианта: если из вершины u мы ранее шли по ребру из паросочетания, то все нормально, так как и в этот раз предполагается именно это. Однако, если из вершины u мы ранее запускали обход по ребрам не из паросочетания, то имеет место несоответствие. Так как сейчас мы, по идее, должны запустить обход из u по ребру из паросочетания, чего ранее сделано не было. Обход в глубину не предполагает повторный обход из вершины, которую уже посетили.

Из этого следует, что обходом в глубину можно пользоваться в тех случаях, когда описанная выше ситуация гарантировано не произойдет. Эта ситуация характеризуется наличием в графе цикла нечетной длины. То есть, если в графе заведомо нет циклов нечетной длины, то обход в глубину всегда будет находить увеличивающий путь, если такой есть. Отсутствие в графе циклов нечетной длины эквивалентно двудольности графа. Поэтому для поиска максимального паросочетания в двудольном графе может использоваться описанный выше алгоритм.

Цветки. Сжатие цветков.

Вернемся к общей схеме. Проблемная ситуация, которая у нас возникла, называется цветком из-за внешнего сходства. То есть, чередующийся путь начинается из свободной вершины и доходит до некоторой вершины b — стебель, вершина b — база, из нее начинается цикл нечетной длины с чередующимися ребрами. Вершина b может быть либо свободной, либо насыщенной последним ребром из стебля цветка.

Утверждается, что весь цикл можно стянуть в одну псевдовершину b сделав ее соседями все вершины, которые были соседними хотя бы для одной вершины из цикла. В получившемся графе можно продолжить поиск увеличивающего пути. В случае его нахождения нужно развернуть цикл, чтобы восстановить корректный увеличивающий путь.

Конкретнее. Во-первых, инвертируем все ребра стебля, очевидно, размер паросочетания не изменится. Тогда вершина b оказалась свободной. Докажем, что цикл можно стянуть в вершину b. Пусть C — цикл нечетной длины, b — свободная вершина из цикла. Граф $G' = G \setminus C$. Пусть b' — но-

вая вершину, образованная при стягивании цикла. Она является свободной вершиной в графе G'. Пусть P — увеличивающий путь в графе G. Если он не содержит вершин цикла C, то он, очевидно, будет увеличивающим путем и в графе G'. Если путь P проходит через какую-либо вершину c цикла C, то заменив ее на b', путь в графе G' тоже будет увеличивающим. Пусть теперь P' — увеличивающий путь в графе G'. Если P' не проходит через вершину b', то он будет увеличивающим и в графе G. В противном случае P' начинается в вершине b'. А вторая вершина пути P' — s обязательно смежна с некоторой вершиной из цикла C. Тогда, поскольку цикл C нечетной длины, то из b существует чередующийся путь в любою вершину цикла, в том числе и в ту, что смежна с s. Таким образом, мы получили увеличивающий путь в графе G.

Не считая трудностей реализации, мы получили метод поиска увеличивающего пути в произвольном графе.

Эффективное решение.

Эффективная реализация описанного выше алгоритма требует дополнительной хитрости. Потому что в явном виде выполнять сжатие и разворачивание цветков проблематично. Поэтому вместо обхода в глубину предлагается обход в ширину. Он не дает принципиальной выгоды кроме возможности добавлять вершины в очередь в любой момент.

Итак, в процессе обхода мы делим все вершины на четные и нечетные. Четные — те, к которым существует чередующийся путь четной длины из корня. Естественно, каждая четная вершина обязательно должна ровно один раз появиться в очереди. Для каждой доступной вершины на протяжении всего обхода будем поддерживать возможность восстанавливать чередующийся путь до корня. Это обеспечит простое инвертирование ребер вдоль увеличивающего пути, если такой найдется. Восстанавливать путь будем следующим образом: из каждой четной вершины переходим в ее пару из паросочетания, а из нечетной — в предка, которого будем определять в процессе обхода. Для поддержания требуемой информации будем для каждой вершины хранить ссылку на базу цветка, в который входит вершина, или на себя, если она не входит ни в один цветок.

Допустим, мы обрабатываем некоторую вершину v, очевидно, четную. Рассмотрим возможные варианты ребер. Пусть ребро ведет в некоторую вершину u. Если u — свободная вершина, то увеличивающий путь найден. Если u — новая, то есть не была ранее отмечена ни как четная, ни как нечетная, то она отмечается как нечетная. А вершина v назначается ее предком. После чего в очередь кладется пара вершины u из паросочетания. Если вершина u уже была отмечена как нечетная, то текущее ребро символизирует наличие цикла четной длины, при этом делать ничего не

надо. Наконец, если вершина u — четная, то это означает, что найден цикл нечетной длины.

Рассмотрим этот случай отдельно. Поскольку мы обнаружили цветок, то должны сжать его в одну вершину — его базу. Для этого ее сначала надо найти. База цветка будет находиться в наименьшем общем предке вершин u и v. Но тут есть один важный нюанс: если наименьший общий предок уже находится в каком-то цветке, то в качестве базы надо взять базу этого цветка. То есть, в этом случае нужно найти первый общий цикл на пути к корню из вершин u и v, и взять его базу (может так оказаться, что их наименьший общий предок сам по себе является базой цикла, тогда нужно брать именно его). Это важно для корректного восстановления путей. После того, как мы нашли наименьшего общего предка, надо все нечетные вершины обособленного цикла сделать четными, то есть, добавить в очередь. Еще надо вдоль цикла расставить новые ссылки на предков для возможности восстановления пути. Каждой вершине кроме базы цветка в качестве предка назначается соседняя вершина цикла, но не ее пара из паросочетания. Таким образом, из каждой вершины цикла по принципу восстановления пути можно будет дойти до базы цикла. Что нам и требовалось. Надо отметить еще одну деталь, если вершины u и v уже лежат в одном цикле, то ребро, их связывающее, рассматривать вообще не надо.

Докажем, что действия в последнем случае, описанные выше, не портят восстанавливаемость путей. Очевидно, что все вершины нового цикла получили чередующийся путь к корню. Так как они получили чередующийся путь к найденной базе, а от нее путь существовал, и мы не могли его изменить. Допустим, проставление предков в новом цикле как-то повлияло на восстанавливаемость пути из некоторой вершины w. Но тогда вершина w уже лежит в каком-то цикле, через базу которого мы провели новый цикл. Если из вершины w в базу ее цикла путь не испортился, тогда чередующийся путь проходит через эту базу и дальше идет по ребрам нового найденного цикла. Если же путь из w к ее прежней базе испортился, то тем, что на пути возникло ребро нового цикла. То есть, опять-таки, можно продолжить движение по нему. Итак, наличие чередующегося пути сохраняется при сжатии цветка.

Оценим сложность решения. Каждый цветок сжимается за $O(N)$, где N — количество вершин. Сжатий цветков может быть не более N. Количество итераций поиска увеличивающего пути не более $\frac{N}{2}$. Итого, общая сложность алгоритма — $O(N^3)$.

Задачи и разборы

Задача A. Ааа

Имя входного файла:	`a.in`
Имя выходного файла:	`a.out`
Ограничение по времени:	2 с
Ограничение по памяти:	256 Мб

Эта и все последующие задачи посвящены истории о девушке, которая сыграла немаловажную роль в жизни авторов.

А началось все так:

Староста группы математиков на очередной абсолютно неважной паре занималась тем, что равномерно распределяла в области ближайшего окружения листочки с судоку. Саша не оказался обделенным вниманием, что послужило поводом заговорить.

— А, надеюсь, это самый сложный уровень?

— Как раз для тебя.

— Ну-ну, — видя глубокую иронию в ее глазах и то, что первые три строки судоку уже заполнены.

— Решишь — еще дам.

— Я тебе не просто решу, а скажу еще, сколько решений существует, — лихо заметил Саша, не подозревая, какие проблемы себе создал.

— Ну-ну.

Следующие два дня Саша занимался только тем, что искал количество решений судоку. Кстати, судоку — это головоломка, в которой предлагается заполнить таблицу 9×9 числами от 1 до 9 так, чтобы в каждой строке, столбце и каждом из 9 квадратов 3×3 все числа были различны. Изначально некоторые клетки уже заполнены и остается вписать числа в пустые клетки.

— А как девушку-то зовут?

— Ааа...

Формат входного файла

В трех строках дано по 9 чисел от 1 до 9 в каждой — первые три строки судоку.

Формат выходного файла

Единственное число — количество вариантов решения. Гарантируется, что хотя бы один вариант существует.

Пример

a.in	a.out
1 4 7 5 6 3 8 2 9 2 5 8 4 7 9 1 3 6 3 6 9 1 2 8 5 4 7	7013953152

Разбор задачи A. Aaa

Решением является перебор с мемоизацией и комбинаторными отсечениями.

Рассмотрим некоторое заполнение первых трех строк. Очевидно, что для дальнейшего заполнения не важен порядок чисел в каждом из столбцов первых трех строк. Поэтому состоянием будет являться набор чисел в каждом из столбцов. На первый взгляд их количество $(C_9^3 \cdot C_6^3 \cdot C_3^3)^3$. Но на самом деле можно хранить гораздо меньше, так как в пределах одного маленького квадрата порядок столбцов не важен. А также можно считать, что первый квадрат — это $\begin{matrix}1 & 4 & 7\\ 2 & 5 & 8\\ 3 & 6 & 9\end{matrix}$ Считаем, сколько способов получить каждое состояние S. Назовем это $D(S)$.

Теперь зафиксируем первую тройку строк из условия и будем перебирать вторую, чтобы она не противоречила первой. Но тоже в терминах состояния, то есть каждый столбец вторых трех строк — это упорядоченный набор чисел. В итоге получится некоторое состояние S_2, и однозначно восстанавливается состояние третьей тройки строк S_3. Тогда к ответу можно прибавить $D(S_1)D(S_2)$.

При переборе состояний второй тройки строк придется их приводить к виду, описанному вначале, то же необходимо делать с третьей тройкой. А именно, нужно переобозначать цифры, чтобы первый квадрат получался тривиальный, и сортировать столбцы.

Учитывая то, что порядок строк в тройке неважен, можно сократить перебор в 6 раз, домножив потом ответ на 36.

Задача B. Bachelor pursuing

Имя входного файла:	`b.in`
Имя выходного файла:	`b.out`
Ограничение по времени:	1 с
Ограничение по памяти:	256 Мб

Не узнав имени девушки в предыдущей задаче, Саша решил навести

справки, благо база данных университета и ВКонтакте всегда под рукой. Оказалось, что Ира — уже четыре года как отличница — твердо хотела получить степень бакалавра, и, как выяснилось впоследствии, степень магистра и степень кандидата наук. И мало ли что дальше?..

— Сколько же это будет в сумме? — с восторгом думал Саша, при этом подсознательно вспоминая задачу с недавнего контеста, которую он так и не решил: даны два числа N и K, вычислить

$$\sum_{i=1}^{N} i^K$$

Формат входного файла

В первой строке дано число Q — количество запросов. В каждой из Q следующих строк даны два числа N и K.

Формат выходного файла

Для каждого запроса в отдельной строке вывести одно число — сумму K-ых степеней натуральных чисел от 1 до N по модулю $1,000,000,007$.

Ограничения

$1 \leq Q \leq 41000$
$1 \leq N \leq 10^9$
$1 \leq K \leq 1000$

Пример

b.in	b.out
4 2 1 4 2 3 3 11 11	3 30 36 675987247

Разбор задачи B. Bachelor pursuing

В этой задаче ограничения были подобраны специально так, чтобы возможные решения с помощью интерполяции не проходили по времени. Так что для решения необходимо было сгенерировать k полиномов, зависящих от n и удовлетворявших условиям

$$\sum_{i=1}^{n} i^k = \sum_{i=0}^{k+1} a_{ki} n^i$$

Для этого обратимся к формуле, использующей числа Бернулли:

$$\sum_{i=1}^{n} i^k = \sum_{i=1}^{k+1} (-1)^{k-i+1} B_{k-i+1} \frac{k!}{i!(k-i+1)!}$$

Заметим, что если мы увеличим k и i на единицу оба, то значение $k-i+1$ не изменится. Таким образом, получаем формулу пересчёта коэффициентов полиномов:

$$a_{ki} = a_{k-1,i-1} \cdot \frac{k}{i}$$

Это даёт нам все значения a, кроме значений a_{k1}. Их мы можем посчитать исходя из того, что любой полином в точке 1 принимает значение 1.

$$a_{k1} = 1 - \sum_{i=2}^{k+1} a_{ki}$$

Таким образом, мы получили все необходимые полиномы за время $O(k^2)$. Далее каждый запрос можно выполнить за $O(k)$ простых операций, просто подставив n в k-тый полином.

Итоговое время выполнения — $O(k^2 + q \cdot k)$

Задача C. Concatenation of credits

Имя входного файла:	`c.in`
Имя выходного файла:	`c.out`
Ограничение по времени:	1 с
Ограничение по памяти:	256 Мб

Ире за ее выдающиеся достижения доверили вести пары у младшекурсников. Имея неплохой опыт занятия на парах посторонними вещами (задача A), Ира нашла себе другое развлечение. Пользуясь тем, что студенты писали зачет и сдали ей все зачетки, она стала искать закономерности в их оценках.

Преподаватели в университете, надо сказать, суровые. Во-первых, 100 баллов принципиально не ставят. А во-вторых, они никогда не поставят в зачётку оценку, которая там уже есть.

Таким образом, Ира видела в каждой зачётке шесть разных оценок от 10 до 99. И вместо того, чтобы поставить туда седьмую, она конкатенировала эти шесть оценок и делила на своё любимое число.

Саша, прогуливающий свою пару ради того, чтобы поприсутствовать на Ириной, следил за происходящим.

— Знаешь, комбинаций оценок, делящихся на это твое любимое число без остатка, довольно много.

— Аж три? — спросила Ира с плохо скрываемой издёвкой.

— Не совсем — у нас в университете студентов не хватит.

— Ну-ну.

Формат входного файла

Дано единственное любимое число I.

Формат выходного файла

Вывести количество способов выбрать упорядоченную шестерку различных двузначных чисел такую, что их конкатенация делится на I.

Ограничения

$1 \leq I \leq 100$

Пример

c.in	c.out
10	44828253360

Разбор задачи C. Concatenation of credits

Динамика $d(m, i, j)$ — количество способов выбрать маску m из 6 чисел таким образом, что все они не больше i, и остаток от деления, который вкладывают эти числа при последующей конкатенации, равен j.

Пересчитывается перебором разряда, который получил это максимальное число i.

$d(m, i, j) = d(m, i-1, j) + \sum\limits_{k \in m} d(m \setminus k, i-1, j - i(100^k) \bmod I)$

Задача D. DeviantArt

Имя входного файла:	d.in
Имя выходного файла:	d.out
Ограничение по времени:	2 с
Ограничение по памяти:	256 Мб

Фотография — еще одно увлечение Иры. Ее произведения занимают достойное место на страницах DeviantArt. Каждый свободный пользователь интернета может зайти в ее галерею и плюсануть понравившиеся изображения.

— Ммм! Неплохая работа для бота.

Суть бота заключалась в следующем: специально созданный пользователь плюсует K фотографий, начиная с I-ой, с шагом A. Второй бот был создан для контроля результата. Он проверяет сумму всех плюсов L фотографий, начиная с J-ой, с шагом B. Фотографии нумеруются с 0.

Боты работают независимо до тех пор, пока администрация DeviantArt не заметит накрутки вклада и не забанит анонимусов.

Формат входного файла

В первой строке даны четыре числа N, A, B и Q — количество фотографий, величины шагов для ботов, которые не меняются на протяжении всей их жизни, и суммарное количество итераций. Далее в каждой из Q строк записана команда для бота:

`s` I K — первый бот плюсует K фотографий, начиная с I-ой.

`g` J L — второй бот суммирует плюсы L фотографий, начиная с J-ой.

До действия ботов фотографии имели рейтинг 0.

Формат выходного файла

Для каждой команды второго бота вывести ее результат в отдельной строке.

Ограничения

$1 \le N \le 10^5$
$1 \le A \le N$
$1 \le B \le N$
$0 \le Q \le 10^5$
$0 \le I \le N - 1$
$1 \le K \le N, I + (K - 1)A \le N - 1$
$0 \le J \le N - 1$
$1 \le L \le N, J + (L - 1)B \le N - 1$

Пример

d.in	d.out
10 2 5 7 s 0 5 g 0 2 g 1 2 g 2 2 s 1 3 g 3 1 g 4 2	1 1 1 1 1

Разбор задачи D. DeviantArt

Используем корневую эвристику по запросам. Ответ на запрос второго типа будет состоять из двух частей: общего пересечения с теми запросами на инкрементирование, которые были после последнего слияния и до него. Слияние происходит каждый $\sqrt{Q}$ запрос. При этом мы объединяем все запросы на инкрементирование с помощью отложенных сумм, считаем количество плюсов каждого элемента, а потом для каждого элемента получаем частичную сумму всех плюсов с шагом B до этого элемента.

Вторую часть суммы для запроса второго типа можно получить, перебрав все запросы первого типа, которые были после последнего слияния. Для этого надо найти, сколько чисел удовлетворяют двум условиям: $i + A \cdot x, 0 \leq x < k$ и $j + B \cdot y, 0 \leq y < l$. Эти числа будут идти с периодом $LCM(A, B)$, надо только найти первое совпадение, а потом можно определить, сколько попадают в заданный промежуток.

Задача E. Exam

Имя входного файла:	`e.in`
Имя выходного файла:	`e.out`
Ограничение по времени:	1 с
Ограничение по памяти:	256 Мб

Неприязнь к философии сближает, необходимость сдавать экзамен — тем более. Но главное — четкое ощущение нежелания к нему готовиться.

— Смотри, Ира, подходишь к преподавателю и предлагаешь ему такую штуку: у нас есть 18 билетов, они лежат лицом вниз. Давайте K из них перевернем, потом Вы их хорошенько перетасуете и положите в одну стопку. А дальше я, в смысле ты, с закрытыми глазами разделишь их на две стопки так, что в них будет одинаковое количество открытых билетов. При этом тебе разрешается, скажем, поменять местами два билета в первой стопке, перевернуть какой-то билет в первой стопке ну и безвозвратно переложить билет из первой стопки во вторую. Ты все это делаешь с закрытыми глазами и о состоянии билетов не имеешь никакой информации кроме той, что ты знаешь число K и помнишь все свои действия.

— Ладно, и что дальше?

— Так вот, если у тебя это благополучно получается, то преподаватель дает тебе возможность отвечать любой среди открытых билетов.

— А если нет?

— Будем верить в чудо.

— Почему бы тебе самому не попробовать?
— Не то что бы у меня хорошая репутация.
— Поэтому хочешь испортить мою?
— Аж три раза.

Формат входного файла

Дано единственное четное число K — количество перевернутых лицом вверх билетов (изначально все 18 билетов находятся в первой стопке).

Формат выходного файла

Если нет возможности разделить билеты на две непустые стопки требуемым образом, вывести -1. Иначе в первой строке вывести количество действий Q. Далее в Q строках описать порядок действий. Каждая строка должна содержать команду одного из трех типов:

`swap i j` — поменять местами билеты на позициях i и j первой стопки.

`rev i` — перевернуть билет на позиции i первой стопки.

`out i` — переместить билет на позиции i из первой стопки во вторую. После этой операции все билеты первой стопки, начиная с $(i+1)$-го, занимают позицию на единицу меньше.

Стопки должны оказаться непустые и содержать одинаковое количество открытых билетов.

Ограничения

$0 \le K \le 18$, K — четное
$0 \le Q \le 2^9 + 36$
$1 \le i \ne j \le 18$, не должны превышать текущий размер первой стопки

Пример

e.in	e.out
18	9 out 1 out 1 out 1 out 1 out 1 out 1 out 1 out 1 out 1

Разбор задачи E. Exam

Решение: взять любые K билетов, перевернуть и отправить во вторую стопку, если $K = 0$, то проделать то же с 9 билетамии. Пусть из K взятых билетов L оказались открытыми, тогда $K - L$ открытых остались в первой стопке. Из перенесенных K билетов после переворачивания стало $K - L$ открытых. Следовательно, во второй тоже $K - L$ открытых.

Задача F. Fate to hate

Имя входного файла:	`f.in`
Имя выходного файла:	`f.out`
Ограничение по времени:	1 с
Ограничение по памяти:	256 Мб

Близился день рожденья Иры. В ночь с 27-го на 28-е Саша долго не мог уснуть. Математически логичный бред дискретно лез в голову. Обыкновенное гадание на ромашке превратилось в жестокую игру чисел и битовых операций.

Снилось бесконечное поле абсолютно одинаковых ромашек. У каждой ромашки было N лепестков, а на каждом лепестке написано число. Время от времени страх перед тем, что Ира может сменить номер своего ICQ заставлял просыпаться в холодном поту. И каждый раз, когда новый номер приходил в голову, его непременно нужно было получить из лепестков, применяя к написанным на них числам операции AND и OR. Если номер удавалось получить, это было хорошим знаком, иначе же сон пропитывался ужасом от ненависти и зла.

Все ромашки одинаковые и их можно безвозмездно срывать для того, чтобы использовать нужные лепестки. То есть для получения одного номера можно сорвать несколько ромашек и взять непустое подмножество их лепестков. В своих снах Саша способен полностью контролировать порядок выполнения операций.

Формат входного файла

В первой строке дано число N — количество лепестков у ромашки. В следующей строке перечислены целые числа на лепестках a_i. Далее дано число Q — количество ICQ-номеров, которые необходимо получить. В каждой из следующих Q строк записано по одному числу b_i.

Формат выходного файла

Для каждого номера в отдельной строке выведите “Yes”, если это число

можно получить из чисел на лепестках, применяя к ним операции побитового AND и OR, и “No”, если нельзя.

Ограничения

$1 \leq N \leq 10^5$
$0 \leq a_i \leq 10^9$
$1 \leq Q \leq 10^5$
$0 \leq b_i \leq 10^9$

Пример

f.in	f.out
3 1 4 5 6 1 2 3 4 5 6	Yes No No Yes Yes No

Разбор задачи F. Fate to hate

Если число можно получить, то можно получить как результат операции OR какого-то множества чисел, каждое из которых является результатом применения операции AND других множеств.

Тогда рассмотрим произвольный единичный разряд интересующего числа. Поскольку его точно нужно получить, то имеет смысл при этом минимально повлиять на остальные разряды. Найдем для каждого разряда i число S_i, которое можно получить из исходного множества, что в нем этот разряд равен 1 и количество других разрядов, равных 1, минимально. Для этого применим операцию AND ко всем числам множества, которые имеют 1 в этом разряде. Теперь, если после применения OR ко всем S_i таким, что i-ый разряд числа из запроса равен 1, получается число из запроса, то его получить можно, иначе — нет. Нужно еще отдельно рассмотреть случай с 0.

Задача G. Genealogic tree

Имя входного файла:	`g.in`
Имя выходного файла:	`g.out`
Ограничение по времени:	1 с
Ограничение по памяти:	256 Мб

На дне рожденья Иры было много родственников, так что знакомство с ними оказалось непростой задачей. Для того, чтобы получить общую картину, возникла идея составить генеалогическое дерево. В сумбурной обстановке как-то так получилось, что дерево оказалось не бинарным, хотя количество вершин было вполне логичным — $2^k - 1$.

— Что-то тут не так.

— Да ну?

— Я предлагаю все перепроверить.

— Конкретнее.

— Разобьем дерево на связные части по 2^i вершин и поручим каждому проверить свою часть.

— Почему именно так?

— Не знаю, люблю степени двойки и к тому же в сумме будет ровно то, что надо. Хотя тут есть маленькая проблема, это можно сделать кучей способов.

— Да, я помню судоку и зачетки. Сколько же на этот раз?

— У тебя великолепная память, дай мне несколько минут на подумать.

— Лови.

Формат входного файла

В первой строке дано число $N = 2^k - 1$ — количество вершин в дереве. В следующей строке дано $N - 1$ числе. a_i означает наличие ребра между вершинами $i + 1$ и a_i.

Формат выходного файла

Вывести одно число — количество способов разделить дерево ровно на k связных блоков размеров $1, 2, 4 \ldots 2^{k-1}$. Каждая вершина должна попасть ровно в один блок.

Ограничения

$0 \le k \le 12$
$1 \le a_i < i + 1$

Пример

g.in	g.out
7 1 1 2 2 3 3	4

Разбор задачи G. Genealogic tree

Задача решается с помощью динамического программирования.

$d(i, j)$ — количество способов разделить поддерево i на деревья размеров, определяемых маской j так, что все остальные элементы связаны с корнем поддерева в одну компоненту. Чтобы пересчитать значение динамики в вершине i, нужно добавить всех ее сыновей. При добавлении некоторого сына k перебираем полученную маску всех добавленных сыновей вершины i до $k - m_i$ и маску сына $k - m_k$. Если они не пересекаются, то можно прибавить к $d(i, m_i + m_k)_k$ число $d(i, m_i)_{k-1} \cdot d(k, m_k)$. Еще нужно учесть, что все поддерево i можно получить, если выбрать связанную с корнем компоненту размером степени двойки, если оставшаяся маска эту степень не содержит.

Задача H. Happy Birthday

Имя входного файла:	`h.in`
Имя выходного файла:	`h.out`
Ограничение по времени:	1 с
Ограничение по памяти:	256 Мб

Настал долгожданный день рождения. Осталось только одно испытание — добраться к Ире домой.

Самый популярный вид транспорта в городе — маршрутки. У каждой маршрутки есть два водителя, один из них любит один маршрут, а второй — другой. Правда место и время отправления маршрутки одинаковы для обоих водителей. Каждый день один из водителей работает, а второй отдыхает. Если Саша окажется на какой-то остановке, то он сразу же узнает, какие из водителей работают сегодня на маршрутках, отправляющихся с этой остановки. Однако до того, как он попадет на остановку, он знает только расписание возможного движения маршруток и вероятность того, что работает первый водитель. Саша живет возле остановки с номером 1 и может оказаться на ней в любое время, а Ира живет рядом с остановкой N. Требуется найти математическое ожидание времени прибытия к Ире на день рождения. При этом Саша никогда не воспользуется способом,

который может привести к тому, что он не попадет к Ире вообще, а среди всех остальных выбирает тот, который минимизирует ожидаемое время прибытия.

Важные факты:

Сеть движения маршруток представляет собой ациклический ориентированный граф.

С маршрутки на маршрутку можно пересаживаться мгновенно.

При определении оптимальной стратегии Саша использует в том числе и то, что по прибытии на каждую остановку он узнает сегодняшних шоферов.

О своей остановке Саша тоже изначально ничего не знает.

Формат входного файла

В первой строке даны два числа N и K — число остановок и действующих маршруток. Далее в каждой из K строк описана информация о маршрутках: семь целых чисел $u\ d\ p\ v_1\ a_1\ v_2\ a_2$ — номер остановки и время отправления, вероятность того, что работает первый водитель, место и время прибытия, если работает первый водитель, место и время прибытия, если работает второй водитель.

Формат выходного файла

Единственное число — математическое ожидание времени прибытия на остановку N с абсолютной или относительной погрешностью 10^{-6} или -1, если есть ненулевая вероятность туда не попасть.

Ограничения

$2 \le N \le 10^5$
$0 \le K \le 10^5$
$1 \le u, v_1, v_2 \le N$
$u \ne v_1$
$u \ne v_2$
$0 \le d, a_1, a_2 \le 1440$
$d < a_1$
$d < a_2$
$0 < p < 100$

Пример

h.in	h.out
5 6 1 60 50 2 200 3 150 1 100 25 2 160 3 150 1 200 50 5 350 4 300 2 180 50 5 300 4 280 3 400 80 5 600 5 660 4 350 50 5 500 5 550	423.4375000000000

Разбор задачи H. Happy Birthday

Рассмотрим простое экспоненциальное решение. В качестве состояния можно взять вершину, время и комбинацию водителей. Тогда, зная вероятность того, что случится такая комбинация, и оптимальный выход из нее, можно определить матожидание для текущего состояния. Заметим, что в отличие от количества комбинаций, количество выходов из них небольшое. Надо только правильно найти вероятность того, что будет осуществлен каждый из них. Поэтому все ребра в вершине будем хранить отсортированными по возрастанию матожидания. А добавлять в порядке уменьшения времени. Вероятность выбора некоторого ребра равна вероятности не выбрать все ребра с меньшим матожиданием, умноженная на вероятность активности самого ребра. Если в список попало ребро, парное которому уже есть в списке, то все ребра с большим матожиданием выбраны быть не могут. Их надо удалить из списка.

Остается только осуществлять операции добавления нового ребра и определения его влияния на другие ребра достаточно эффективно. Это можно сделать с помощью декартового дерева с множественным запросом на умножение на число, то есть вероятность, и получением зависящей от него величины, то есть матожидания. Еще для простоты имеет смысл хранить в вершине множество всех ее матожиданий, отсортированных по времени, в которое они начинают действовать. Это множество пополняется, когда рассматривается новое ребро, исходящее из данной вершины.

Задача I. I love Ira

Имя входного файла:	`i.in`
Имя выходного файла:	`i.out`
Ограничение по времени:	3 с
Ограничение по памяти:	256 Мб

Саша хотел сделать Ире совершенно необычный подарок на день рождения. Романтическое сообщение, выложенное решенными задачами, на самом популярном украинском сайте, посвященном спортивному программированию.

Задачи на сайте располагаются в виде прямоугольной таблицы, сданные задачи подсвечиваются приятным зеленым цветом. Это однажды натолкнуло на мысль зажечь в таблице только те ячейки, которые бы сформировали задуманную картинку или надпись...

И если с содержанием сообщения неоднозначностей не возникало, то его расположением в таблице можно было управлять. Конечно же от места расположения надписи зависело то, какие задачи придется решать. Поэтому Саша назначил каждой задаче оценку сложности — число от 0 до 9. И решил выбрать такое расположение, чтобы сумма оценок решенных им задач была максимальна. Надпись может располагаться в таблице где угодно, но только чтобы весь шаблон находился в пределах таблицы.

Формат входного файла

В первой строке даны два числа R и C — количество строк и столбцов в таблице. Далее в R строках описана карта задач. В каждой строке ровно по C символов — сложности задач. Сложность задачи измеряется цифрой от 0 до 9. В следующей строке даны два числа H и W — высота и ширина надписи. Далее в H строках дан шаблон надписи. Каждая строка состоит из W символов: **'#'** означает необходимость решения задачи, **'.'** означает, что задачу надо не решать.

Формат выходного файла

В единственной строке вывести максимальную сумму оценок сложности решенных задач.

Ограничения

$1 \le R, C \le 800$
$1 \le H \le R$
$1 \le W \le C$

Пример

i.in	i.out
22 18 000000000001000000 003001000000000000 000000000000000003 020000000010000000 090000000901000100 300400001911323500 000000000000000000 000000000000000000 000001000001000049 020100113210100000 200000006220004100 600009000217100009 000001000031004210 020101600100000190 110001002000000000 000092096071021990 000094000120010100 201003007430040300 100000010000100000 101040008000020001 404000000040043200 000000010000001000 17 18 ###...#..... .#....#..... .#....#..#..#.#.## .#....#.#.#.#.#.#. .#....#.#.#.#.#.## .#....#.#.#..#..#. ###...#..#...#..## ....###..... ...#..... ...#..#.##.##..... ...#...#.#...#.... ...#...#....##.... ...#...#...#.#.... ..###..#...##.#...	69

Разбор задачи I. I love Ira

Развернем таблицу в одну строку. То есть сначала будут идти цифры первой строки, потом второй и так далее. Шаблон тоже выпишем в отдельную строку, но в обратном порядке. Кроме того, между его строками вставим $C-W$ нулей. Это необходимо, чтобы символы шаблона четко соответствовали символам таблицы. Заметим, что если мы перемножим строку таблицы на строку шаблона как полиномы, то в результате коэффициентом при каждой степени будет сумма чисел таблицы при некотором положении шаблона в таблице. Тогда можно перебрать все позиции шаблона в таблице и посмотреть соответствующий коэффициент в полиноме. Среди всех таких коэффициентов надо взять максимум, это и будет ответом.

Чтобы эффективно реализовать произведение многочленов, можно воспользоваться быстрым преобразованием Фурье.

Задача J. Justice

Имя входного файла:	`j.in`
Имя выходного файла:	`j.out`
Ограничение по времени:	1 с
Ограничение по памяти:	256 Мб

— Давай сыграем в игру. Есть несколько кучек, в каждой из них сколько-то камешков. За один ход можно брать любое количество камешков из одной кучки, хоть все. Кто не может сделать ход, тот проиграл.
— Ладно, только я первая хожу.
— Ладно, тогда я выбираю сколько у нас будет камешков.
— Ладно, тогда я выбираю сколько будет кучек.
— А я тогда распределяю камешки по кучкам.
— Удачи.

Формат входного файла

Два числа N и K - количество камней и количество кучек.

Формат выходного файла

Если нельзя распределить N камешков ровно на K непустых кучек таким образом, что при оптимальной игре обоих выиграет второй игрок, вывести -1. Иначе вывести ровно K натуральных чисел a_i — размеры кучек.

Ограничения

$1 \leq N \leq 10^9$
$2 \leq K \leq 16$

Пример

j.in	j.out
4 2	2 2

Разбор задачи J. Justice

Прежде всего, рассмотрим все очевидные случаи, когда решения нет. Это либо $n < k$, либо $n = k + 1$, либо n — нечетное.

Если k — четное, то ответ есть: возьмем $k-2$ кучки размером 1 и две по $\frac{n-k-2}{2}$ камней. Если k — нечетное и $k > 3$, то можно взять три кучки размеров 1, 2 и 3 и дальше все как с четным k.

Отдельно рассмотрим $k = 3$. Пусть 2^s — максимальная степень двойки, на которую делится n. Тогда можно разбить на три кучки $\frac{n}{2}$, $\frac{n}{2} - 2^{s-1}$, 2^{s-1}. Но это не работает, если n является степенью двойки.

Докажем, что степени двойки нельзя разложить на три кучки. Допустим есть три числа a, b и c такие, что $a + b + c = 2^n$ и $a \oplus b \oplus c = 0$. Тогда в двоичном представлении в каждом разряде будет четное количество 1. Но как только встретится первый разряд, в котором есть 1, то там их будет две и при сложении в следующий разряд перенесется 1. Там ее нужно будет компенсировать, а это сделать нельзя, так как там должно быть четное количество 1. Существует только тривиальное решение, при котором одно из чисел равно 0. Нам это не подходит, мы пришли к противоречию.

Задача K. Knife to me

Имя входного файла:	`k.in`
Имя выходного файла:	`k.out`
Ограничение по времени:	1 с
Ограничение по памяти:	256 Мб

Праздничный торт был очень хорош, нежный кофейный бисквит с прослойками крема, залитый белым шоколадом — чудо кулинарного искусства. Одна проблема. Торт имеет форму призмы, в основании которой лежит правильный N-угольник — как раз по числу гостей. Но незапланированный Саша усложнил ситуацию. Теперь торт надо делить на $N + 1$ часть.

— Кто у нас тут любит геометрию?

— Ну ты тоже вроде на матфаке училась.

— Поразительная наглость. Ты во всем виноват — тебе и нож в руки.

— Тогда я предлагаю делать вертикальные разрезы.

— Ну ты прямо Капитан Очевидность.

— Это еще не все. Делим ровно на $N + 1$ часть так, чтобы объемы всех кусков были одинаковы.

— Ну ок.

— Это все еще не все. Делим так, чтобы площадь поверхности, залитой шоколадом, у всех кусков тоже была одинакова.

— Проще было самой разрезать.

Будем рассматривать проекцию торта на стол. Получается правильный многоугольник с центром в точке $(0, 0)$ и с одной из вершин в точке $(1, 0)$. Требуемые куски должны быть такими выпуклыми многоугольниками, что никакие три вершины куска не лежат на одной прямой. План разрезания должен удовлетворять условиям независимо от высоты торта. Напомним, что шоколадом залита только внешняя поверхность торта, то есть верхняя и все боковые грани.

Формат входного файла

Число N — количество гостей, не считая Саши.

Формат выходного файла

Ровно $N+1$ блок, в каждом из которых описан отдельный кусок. Описание куска начинается с числа K_i — количества вершин. Далее в K_i строках должны быть перечислены вершины куска в порядке обхода против часовой стрелки. Допускается погрешность 10^{-8}.

Ограничения

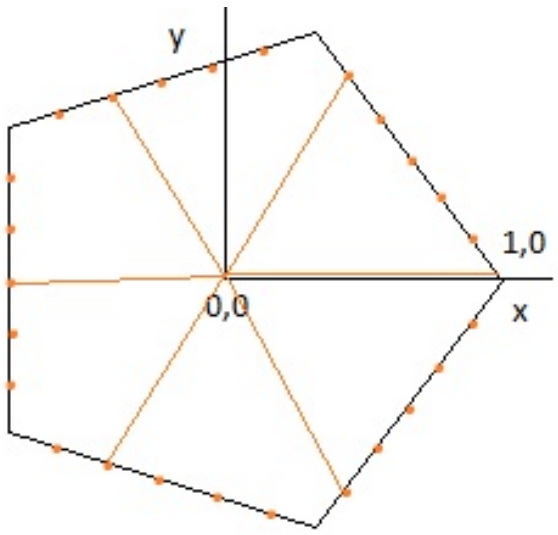
$3 \leq N \leq 100$
$3 \leq K_i \leq 100$

Пример

k.in	k.out
3	3 0.0000000000000 0.0000000000000 1.0000000000000 0.0000000000000 -0.1250000000000 0.6495190528383 4 0.0000000000000 0.0000000000000 -0.1250000000000 0.6495190528383 -0.5000000000000 0.8660254037844 -0.5000000000000 0.0000000000000 4 0.0000000000000 0.0000000000000 -0.5000000000000 0.0000000000000 -0.5000000000000 -0.8660254037844 -0.1250000000000 -0.6495190528383 3 0.0000000000000 0.0000000000000 -0.1250000000000 -0.6495190528383 1.0000000000000 -0.0000000000000

Разбор задачи K. Knife to me

Разделим все стороны многоугольника ровно на $N + 1$ часть каждую, как показано на рисунке. Все треугольники, образованные соседними точками и центром координат, имеют равную площадь. Очевидно, имеют одинаковую длину внешнего периметра.



Поэтому можно формировать куски, объединяя N последовательных тругольников.

Задача L. Lie to me

Имя входного файла:	`l.in`
Имя выходного файла:	`l.out`
Ограничение по времени:	1 с
Ограничение по памяти:	256 Мб

— Ничего личного, но что-то я не очень доверяю твоему способу разрезания торта.
— Ну и зря.
— Ну так докажи.
— Я не настолько крут, но вот мой ноутбук мне никогда не врет.
— Что-то я не вижу у тебя за спиной рюкзака.
— А, черт, неужели я его не взял!.. Шутка, взял, конечно.

Напомним, что план разрезания торта содержит следующую информацию: торт — правильный N-угольник с центром в точке $(0, 0)$ и одной из вершин в точке $(1, 0)$. Весь торт разрезан на $N + 1$ кусков, каждый из которых — выпуклый многоугольник.

Проверить, правда ли, что все куски — выпуклые многоугольники без трех точек на одной прямой, в порядке обхода против часовой стрелки. Проверить, все ли куски имеют равную площадь. Проверить, все ли куски имеют равную длину внешней границы, то есть той части границы, которая повторяет исходную границу торта. Правда ли, что все куски вместе составляют целый торт.

Формат входного файла

В первой строке дано число N — количество вершин торта. Далее идет описание $N+1$ куска. Описание одного куска начинается с числа K_i — количества вершин в куске. Далее в K_i строках даны вершины куска двумя своими координатами (произвольный набор точек).

Формат выходного файла

Вывести “Yes”, если описание соответствует корректному разрезанию торта, иначе — “No”. Гарантируется, что при ответе “No” показатели, обеспечивающие этот ответ, не менее 10^{-6}.

Ограничения

$3 \leq N \leq 100$
$3 \leq K_i \leq 100$
Все координаты — вещественные числа, не более чем с 15 знаками после запятой, по модулю не превышают 100.

Пример

l.in	l.out
3 3 0.000000000000 0.000000000000 1.000000000000 0.000000000000 -0.125000000000 0.649519052838 4 0.000000000000 0.000000000000 -0.125000000000 0.649519052838 -0.500000000000 0.866025403784 -0.500000000000 0.000000000000 4 0.000000000000 0.000000000000 -0.500000000000 0.000000000000 -0.500000000000 -0.866025403784 -0.125000000000 -0.649519052838 3 0.000000000000 0.000000000000 -0.125000000000 -0.649519052838 1.000000000000 -0.000000000000	Yes

Разбор задачи L. Lie to me

Нужно явно сделать то, что требуется в задаче. Проверить, являются ли введенные наборы точек выпуклыми многоугольниками в порядке обхода против часовой стрелки. Проверить, находятся ли они внутри исходного правильного многоугольника, для этого можно просто проверить, находятся ли все их вершины внутри правильного многоугольника. Определить площади всех многоугольников и проверить, равны ли они $\frac{N}{N+1}\frac{\sin\frac{2\pi}{N}}{2}$. Найти все стороны каждого многоугольника, которые полностью принадлежат границе правильного многоугольника. Убедиться, что сумма их длин равна $\frac{N}{N+1}2\sin\frac{\pi}{N}$. После всего этого нужно проверить, что все многоугольники попарно не пересекаются. Проверку на пересечение можно осуществлять следующим образом: проверить, не лежат ли вершины одного многоугольника строго внутри другого, проверить, нет ли нетривиальных пересечений сторон многоугольников и проверить каждую вершину одного

многоугольника, которая совпадает с вершиной другого или лежит на его стороне. Если все эти проверки прошли, то многоугольники не пересекаются. Если вообще все проверки прошли, то разрезание корректное, иначе — нет.

Задача M. MySpace

Имя входного файла:	`m.in`
Имя выходного файла:	`m.out`
Ограничение по времени:	1 с
Ограничение по памяти:	256 Мб

Ситуация, аналогичная той, что описана в задаче D, сложилась на сайте MySpace.

Вкратце, два бота должны были увеличивать рейтинг фотографий. Только в связи со сложностью интерфейса сайта или просто лени, их задачи были немного упрощены. Первый бот плюсует номера всех фотографий, которые имеют остаток от деления на A равный I. Второй бот считает сумму плюсов всех фотографий, имеющих остаток от деления на B равный J. Фотографии нумеруются начиная с 0.

Формат входного файла

В первой строке даны четыре числа N, A, B и Q — количество фотографий, величины шагов для ботов, которые не меняются на протяжении всей их жизни, и суммарное количество итераций. Далее в каждой из Q строк записана команда для бота:

`s` I — первый бот плюсует фотографии $I, I + A, I + 2A, I + 3A \ldots$

`g` J — второй бот суммирует плюсы фотографий $J, J + B, J + 2B, J + 3B \ldots$

До действия ботов фотографии имели рейтинг 0.

Формат выходного файла

Для каждой команды второго бота вывести ее результат в отдельной строке.

Ограничения

$1 \le N \le 10^5$
$1 \le A \le N$
$1 \le B \le N$
$0 \le Q \le 10^5$
$0 \le I < A$

$0 \le J < B$

Пример

m.in	m.out
10 2 5 7 s 0 g 0 g 1 g 2 s 1 g 3 g 4	1 1 1 2 2

Разбор задачи M. MySpace

Если число A больше чем $\sqrt{N}$, то запросы первого типа можно осуществлять явно, перебирая все изменяемые числа, прибавляя 1 к соответствующему остатку от деления на B. А при запросе второго типа просто посмотреть, сколько раз был инкрементирован некоторый остаток.

Если число B больше чем $\sqrt{N}$, то все наоборот: запросы первого типа выполняются за константу прибавлением 1 к текущему остатку. А запросы второго типа осуществляются явно перебором всех запрашиваемых чисел и суммированием значений в соответствующих остатках от деления на A.

Если оба числа меньше $\sqrt{N}$, то можно при запросе на изменение поступать так же, как и в предыдущем случае, а при запросе на поиск суммы перебирать все остатки от деления на A и считать, сколько раз каждый из них встретится в последовательности элементов, сумму которых надо найти. Эту величину можно предпосчитать.

Задача N. Now

Имя входного файла:	`n.in`
Имя выходного файла:	`n.out`
Ограничение по времени:	1 с
Ограничение по памяти:	256 Мб

Сейчас зима, во всех смыслах. И лишь воспоминания о прошедшем заставляют трезво смотреть на вещи.

— Неужели это конец?

— Кто знает...

— Ну тогда скорее к делу!

Дан неориентированный граф без петель и кратных ребер. Найти величину максимального паросочетания, то есть максимальный размер подмножества P ребер графа так, что любой вершине инцидентно не более одного ребра из P.

Формат входного файла

В первой строке даны два числа N и K — количество вершин и ребер в графе. Каждая из следующих K строк содержит по два числа u и v — описание одного ребра. Гарантируется, что граф вполне случайный.

Формат выходного файла

Вывести единственное число — величину максимального паросочетания.

Ограничения

$1 \leq N \leq 400$

$0 \leq K \leq \frac{N(N-1)}{2}$

Пример

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

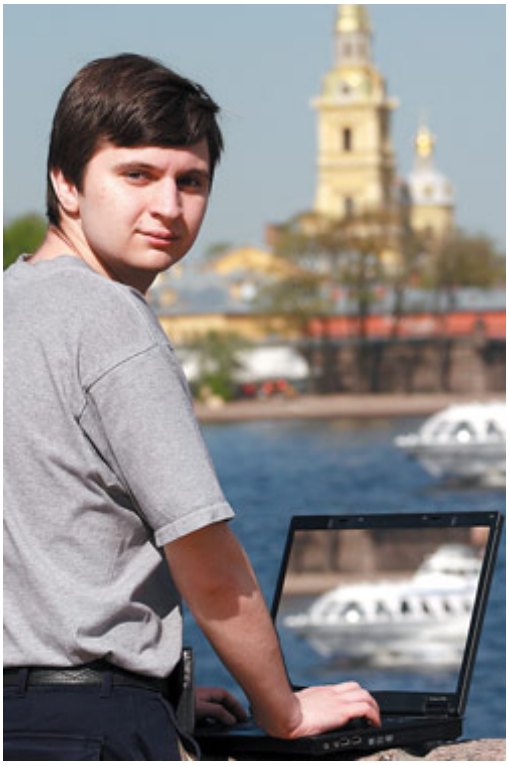
Разбор задачи N. Now

Важным фактом для решения задачи является случайность графа. В таком графе максимальное паросочетание можно легко найти рэндомизированным алгоритмом. Для этого надо просто запускать обход в глубину, который возмножно будет находить некоторый увеличивающий путь. Периодически надо перемешивать списки смежности ребер для каждой вершины, чтобы обход в глубину на следующей итерации не совпадал с предыдущим. С большой вероятностью за количество итераций порядка N будет найдено максимальное паросочетание.

День второй (19.02.2012 г.)
Контест Дворкина Михаила Эдуардовича

Об авторе...

Дворкин Михаил Эдуардович, исследователь в Лаборатории вычислительной биологии Академического университета РАН, преподаватель в лицее “Физико-техническая школа”, аспирант СПбГУ ИТМО. Раньше много участвовал в олимпиадах по программированию, сейчас же больше занимается их организацией, а накопленный опыт передает подрастающему поколению. На ТопКодере и в Живом журнале использует псевдоним Дарнли.



Основные достижения:

- Золотой медалист Чемпионата мира по программированию ACM ICPC 2007 года.

Теоретический материал. Приближенные алгоритмы для NP-полных задач

В некоторых случаях, требуется за приемлемое время решить задачу, про которую известно, что она NP-полна. Найти точное решение за полиномиальное время возможно лишь в случае, если $P = NP$ — а это открытая задача, одна из семи «задач тысячелетия».

Но что если для практической цели подойдет не только оптимальное решение задачи, но и некоторое решение, не сильно отстающее от оптимального? В случаях, когда это приемлемо, на помощь приходят приближенные алгоритмы — алгоритмы, работающие за полиномиальное время, и дающие ответ, не обязательно оптимальный, но дающий ответ хуже оптимального в известное число раз.

Несколько примеров таких алгоритмов для различных NP-полных задач рассмотрены в данном тексте.

Задача о вершинном покрытии (VERTEX COVER)

(Оптимизационная) задача VERTEX COVER формулируется следующим образом:
Вход. Неориентированный граф G.
Задача. Выбрать множество вершин S графа G как можно меньшей мощности, такое что для любого ребра e из графа G хотя бы один из его концов принадлежит S.

Предложим следующее приближенное решение данной оптимизационной задачи VERTEX COVER. Построим в графе G максимальное паросочетание M. И ответом на задачу VERTEX COVER положим множество концов ребер, входящих в это паросочетание. Таким образом, размер предложенного ответа (мощность предложенного множества) $OUR = 2 \cdot |M|$ (каждое из $|M|$ ребер дает две вершины к получаемому множеству).

Однако рассмотрим оптимальный ответ на задачу, пускай в нем OPT вершин. Эти вершины покрывают все ребра графа, в частности все ребра паросочетания M. А значит, для каждого ребра из M хотя бы один из его концов принадлежит оптимальному множеству, то есть $OPT \geq |M|$. Откуда:

$$\frac{OUR}{OPT} \leq \frac{2 \cdot |M|}{|M|} = 2$$

Предложенный ответ не более чем в два раза хуже теоретически достижимого оптимума.

Следует добавить замечание, не влияющее на теоретическую оценку, однако позволяющее добиться более хороших результатов при практическом применении. Дело в том, что вместо *максимального* паросочетания можно было использовать любое *насыщенное* паросочетание в графе G — то есть такое паросочетание, которое нельзя увеличить, прибавив к нему одно ребро.

Плюсов сразу несколько: во-первых, для нахождения любого насыщенного паросочетания годится тривиальный жадный алгоритм (каждый раз добавлять в паросочетание любое ребро, у которого оба конца не принадлежат паросочетанию), что проще для реализации. Во-вторых, мощность итогового множества не увеличится, а для многих графов — строго уменьшится.

Задача о разбиении на k кластеров (k-CLUSTERING)

(Оптимизационная) задача k-CLUSTERING формулируется следующим образом:
Вход. Неориентированный взвешенный граф G и натуральное число k.

Задача. Разбить вершины графа G на k множеств, так чтобы наибольший из диаметров этих множеств был как можно меньше.

(Диаметром множества вершин называется наибольшее среди попарных расстояний между вершинами этого множества.)

Во-первых, отметим, что в тривиальном случае, когда количество вершин $|V|$ не превосходит k, оптимальный ответ очевиден (назвать каждую вершнину отдельным кластером) и строится за полиномиальное время. Поэтому полагаем $k < |V|$.

Теперь выберем произвольную вершину a_1 и определим последовательно вершины a_2, ..., a_{k+1} следующим образом: вершина a_i — это вершина, расстояние от которой до ближайшей из вершин $\{a_1, a_2, \ldots, a_{i-1}\}$ максимальное (если таких вершин несколько, то выбирается произвольная из них). Найти очередную вершину a_i несложно: достаточно ввести фиктивную вершину s, расстояние от которой до a_1, a_2, ..., a_{i-1} положить равным нулю, после чего любым алгоритмом поиска кратчайших путей найти расстояния от s до всех вершин графа и выбрать самую дальнюю. Введем также обозначение: расстояние от a_i до множества $\{a_1, a_2, \ldots, a_{i-1}\}$ назовем r_i.

Заметим, что $r_2 \geq r_3 \geq \ldots \geq r_{k+1}$. Пусть $i < j$. Тогда, когда была выбрана вершина a_i, оказавшаяся на расстоянии r_i от множества $\{a_1, a_2, \ldots, a_{i-1}\}$, вершина a_j была от этого множества на меньшем либо равном расстоянии (иначе была бы выбрана именно она). Значит, от какой-то из вершин этого множества до a_j расстояние не превышает r_i, следовательно $r_j \leq r_i$.

Кроме того, на последней итерации, когда была выбрана вершина a_{k+1}, все остальные вершины были не дальше к множеству $\{a_1, a_2, \ldots, a_k\}$, чем она. То есть для любой вершины кроме $\{a_1, a_2, \ldots, a_k\}$ существует вершина в множестве $\{a_1, a_2, \ldots, a_k\}$ на расстоянии не больше r_{k+1}.

Из последнего утверждения рождается предлагаемый ответ на оптимизационную задачу k-CLUSTERING. Отнесем вершины $\{a_1, a_2, \ldots, a_k\}$ в кластеры с соответствующими номерами и назовем центрами кластеров, а каждую из остальных вершин отнесем в кластер с ближайшим к ней центром (в любой из них, если таковых больше одного).

При таком кластерообразовании каждая вершина либо является центром, либо отстоит от центра своего кластера на расстояние не больше r_{k+1}. Значит, внутри каждого кластера, между любыми двумя вершинами есть путь через центр кластера длины не больше $2r_{k+1}$. Итак, наибольший диаметр кластера в предложенной конфигурации $OUR \leq 2r_{k+1}$.

Но что же оптимальный ответ? Каков бы он ни был, в нем какие-то две вершины из множества $\{a_1, a_2, \ldots, a_{k+1}\}$ по принципу Дирихле окажутся в одном кластере. Пусть это вершины a_i и a_j, где $i < j$. Вспоминая выбор

вершины a_j на j-той итерации, сделаем вывод, что расстояние от a_j до a_i не меньше r_j, которое в свою очередь не меньше r_{k+1}. Таким образом диаметр этого кластера не меньше r_{k+1}, откуда

$$\frac{OUR}{OPT} \leq \frac{2r_{k+1}}{r_{k+1}} = 2$$

Задача коммивояжера (TRAVELLING SALESMAN) в графе с неравенством треугольника

(Оптимизационная) задача TRAVELLING SALESMAN формулируется следующим образом:
Вход. Неориентированный взвешенный граф G.
Задача. Найти простой цикл в графе G, который проходит через все вершины, и имеет как можно меньшую длину.

Рассмотрим эту задачу для графов, в которых выполнено неравенство треугольника — для любых трех вершин i, j, k выполнено $w_{ij} \leq w_{ik} + w_{kj}$ (здесь w — весовая функция, заданная на ребрах графа). Если в некотором графе оно не выполнено, можно применить процедуру транзитивного замыкания графа.

Найдем в графе G минимальное остовное дерево (алгоритм Прима / алгоритм Краскала). Назовем суммарную длину ребер, вошедших в него, MST.

Следующая фаза — добавить в минимальное остовное дерево несколько ребер так, чтобы в полученном графе степени всех вершин стали четными. Предложим два способа — простой и эффективный.

Способ №1 (простой). Удвоим все ребра. (Добавим кратное ребро к каждому ребре дерева). Полученный граф имеет суммарную длину ребер $2 \cdot MST$ и, естественно, все вершины в нем имеют четную степень (она только что удвоилась).

Способ №2 (эффективный). Рассмотрим подграф графа G, состоящий только из вершин, которые в минимальном остовном дереве имеют нечетную степень (это все листья плюс некоторые узловые вершины). Естественно, количество вершин в этом графе — четное. В этом графе найдем минимальное по весу полное паросочетание (поиск минимального по весу паросочетания во взвешенном недвудольном графе — задача нетривиальная, но разрешимая за полиномиальное время). Назовем суммарную длину ребер в найденном паросочетании M и добавим все ребра этого паросочетания к минимальному остовному дереву (возможно, породив кратные ребра в нем). У всех вершин, которые имели нечетную степень, степень увеличится при этом на единицу, и все степени станут четными.

Полученный одним из двух способов граф содержит все вершины G, является связным и не содержит вершин с нечетными степенями. Следовательно, в этом графе (за полиномиальное время) можно найти эйлеров цикл.

Просмотрим все вершины полученного эйлерова цикла по порядку, и будем добавлять каждую из них к (изначально пустому) ответу, если она там еще не присутствует. Так будет получена последовательность вершин, в которой каждая вершина графа G присутствует ровно один раз. Это и будет предложенный ответ на задачу коммивояжера (с суммарной длиной OUR). Отметим, что благодаря неравенству треугольника, процедура выписывания вершин, которые еще не встречались, сделает полученный цикл не более длинным, чем изначальный эйлеров цикл (все изменения заключаются в том, что вместо пути из нескольких ребер берется ребро, соединяющее его концы; при этом длина не увеличивается).

Теперь рассмотрим оптимальный цикл для коммивояжера, пусть его длина OPT. Выкинем из него произвольное ребро, получив путь длины $P < OPT$. Однако полученное множество ребер является не только путем, но и остовным деревом (является деревом и проходит через все вершины графа). А длина какого-то остовного дерева не может быть меньше длины минимального остовного дерева, не так ли? Следовательно, $OPT > P \geq MST$.

Осталось оценить длину предложенного цикла OUR.

Цикл, полученный способом №1. Суммарная длина ребер удвоенного минимального остовного дерева равна $2 \cdot MST$. Такая же, очевидно, и длина эйлерова цикла в этом графе. В результате выписывания вершин эйлерова цикла по одному разу, получится цикл длины не больше $2 \cdot MST$, откуда

$$\frac{OUR}{OPT} < \frac{2 \cdot MST}{MST} = 2$$

Цикл, полученный способом №2. Требуется оценить величину $MST + M$ — именно такова суммарная длина ребер эйлерова графа; длина итогового цикла по уже указанным причинам не превосходит эту величину.

Напомним, что M — это вес паросочетания, найденного в графе нечетных вершин минимального остовного дерева. Рассмотрим, как оптимальный цикл проходит через эти вершины. Пусть эти нечетные вершины в оптимальном цикле встречаются в таком порядке: $a_1, a_2, \ldots, a_{2m}$ (их, конечно, четное число). Цикл, состоящий из этих вершин в этом порядке есть «сокращение» цикла длины OPT, и по неравенству треугольника его длина не превышает OPT.

А теперь рассмотрим два множества ребер: $M_1 = \{(a_1, a_2), (a_3, a_4), \ldots, (a_{2m-1}, a_{2m})\}$ и $M_2 = \{(a_2, a_3), (a_4, a_5), \ldots, (a_{2m-2}, a_{2m-1}), (a_{2m}, a_1)\}$. Эти два множества ребер в сумме образуют рассмотренный выше цикл длины не больше OPT. Следовательно, хотя бы одно из них, пусть M_1, имеет суммарную длину не больше $\frac{OPT}{2}$. Но M_1 является полным паросочетанием в графе нечетных вершин! В то же время использованное ранее M является наименьшим по весу паросочетанием в том же графе, значит, вес M меньше либо равен весу M_1, не превыщающему $\frac{OPT}{2}$. Итак,

$$\frac{OUR}{OPT} \leq \frac{MST + M}{OPT} = \frac{MST}{OPT} + \frac{M}{OPT} < \frac{MST}{MST} + \frac{\frac{OPT}{2}}{OPT} = \frac{3}{2}$$

Задача о покрытии множествами (SET COVER)

(Оптимизационная) задача SET COVER формулируется следующим образом:
Вход. Семейство множеств S, являющихся подмножествами множества U.
Задача. Найти подмножество S, объединение которого равно U, и которое имеет как можно меньшую мощность.

Будем действовать жадно: начнем с пустого множества и будем выбирать каждый раз такое множество из S, в котором присутствует как можно большее число еще не покрытых элементов U.

Пусть перед очередной итерацией было непокрыто m элементов из U. Но нам известно, что есть оптимальное покрытие размера OPT, которое в частности покрывает эти m элементов. Следовательно, какое-то из множеств, входящих в оптимальное покрытие, по принципу Дирихле, покрывает не менее $\frac{m}{OPT}$ из этих элементов. А предложенный жадный алгоритм выберет множество, покрывающее как можно больше непокрытых элементов, то есть не менее $\frac{m}{OPT}$.

Значит, после этой итерации число непокрытых элементов уменьшится с m до

$$m' \leq m - \frac{m}{OPT} = m\left(1 - \frac{1}{OPT}\right) < m \cdot e^{-\frac{1}{OPT}}$$

И эта оценка позволяет оценить число требуемых итераций. В начале число непокрытых элементов равнялось $|U|$. Рассмотрим число непокрытых элементов после $OPT \ln |U|$ итераций.

$$m'' < |U| \cdot e^{-\frac{1}{OPT} \cdot OPT \ln |U|} = |U| \cdot e^{-\ln |U|} = |U| \cdot |U|^{-1} = 1$$

Осталось строго меньше одного непокрытого элемента, то есть все элементы покрыты. При этом было использовано (одна итерация — одно множество) $OPT \ln |U|$ множеств, и

$$\frac{OUR}{OPT} \leq \frac{OPT \ln |U|}{OPT} = \ln |U|$$

Полученный ответ отличается от оптимального не в ограниченное константой число раз, как в предыдущих случаях, а в $\ln |U|$. Круг применения такого приближения *уже*, однако, в некоторых ситуациях и логарифмическое приближение к оптимуму может оказаться весьма полезным.

Автор благодарит Сергея Владимировича Копелиовича за чрезвычайно плодотворные дискуссии на тему приближенных алгоритмов для NP-полных задач.

Задачи и разборы

Задача A. Points

Имя входного файла:	`a.in`
Имя выходного файла:	`a.out`
Ограничение по времени:	1 с
Ограничение по памяти:	64 Мб

Разместите в круге с единичным радиусом и центром в начале координат n точек так, как вам нравится.

Среди площадей всех треугольников с вершинами в этих точках выберем наименьшую. Эта величина — ваш счёт; максимизируйте его.

Ограничения

$11 \leq n \leq 50$.

Формат входного файла

Первая строка входного файла содержит целое число n.

Формат выходного файла

Выведите $2n$ чисел — n пар координат поставленных вами точек.

Пример

a.in	a.out
4	-1 0 0 -1.0 1 0 0 1.0

Пояснение

Счёт в данном примере: 1.0 (все треугольники имеют площадь 1, минимум равен 1).

Разбор задачи A. Points

Автор благодарит своего друга Дмитрия Карабаша (США, Курантовский институт математических наук) за обсуждение задач Сойфера-Эрдеша, одна из которых стала задачей для олимпиадного программирования.

Оптимальное решение данной задачи не известно до сих пор. Асимптотика результата (площади наименьшего треугольника) по всей видимости убывает пропорционально n^{-2}.

Участники соревнований могли применить следующие подходы.

Подход 1. Расстановка точек в соответствии с некой математической конфигурацией. Например, вершины правилного m-угольника, расположенные на единичной окружности, и вершины правильного $(n-m)$-угольника, расположенные на окружности радиуса $r < 1$.

Подход 2. Перебор произвольных расположений n точек. Даже «кидая» в круг точки произвольным образом, например, с равномерным по площади распределением, можно было добиться терпимых результатов (особенно при многократном итерировании). Несложное отсечение, которым не стоит пренебрегать: пускай на одной из предыдущих итераций была уже получена конфигурация с минимальной площадью S_{best}. Тогда при «бросании» точек в круг на очередной итерации не следует ставить точку, если она с двумя другими уже поставленными образует треугольник площади меньше S_{best} — лучше следать повторный «бросок», а если он не удался несколько раз подряд, то закончить данную итерацию.

Кроме того, можно отойти от равномерного распределения точек, например, «скосить» распределение от центра круга к окружности. Также, можно постановить, что m из n точек точно лежат на окружности — равномерно распределены по ней, либо образуют вершины правильного m-угольника.

Подход 3. Генетические и прочие эволюционные алгоритмы. Рассматривая n точек как популяцию, можно применить к таким популяциям законы естественного отбора и через (немаленькое) число поколений «вырастить» эволюционными алгоритмами популяцию с отличными характеристиками.

Подход 4. Метод симулированного отжига и прочие методы оптимизации. Применение именно таких методов позволило участникам соревнования добиться наилучших конфигураций.

Приведем одну из них. Авторы конфигурации — команда ONU_1_2_3, Одесский национальный университет.

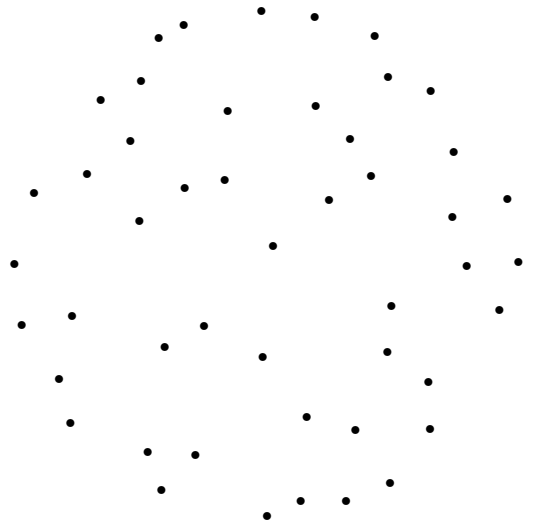


Рис. 1: Расстановка 48 точек в единичном круге

Задача B. Queens

Имя входного файла:	`b.in`
Имя выходного файла:	`b.out`
Ограничение по времени:	1 с
Ограничение по памяти:	64 Мб

Всем давно известна (а кому-то оскомину набила) задача о расстановке не бьющих друг друга шахматных ферзей на доске $n \times n$.

Давайте внесем немного агрессии. Но умеренной.

Расставьте на доске $n \times n$ как можно больше ферзей так, чтобы каждый бил ровно b других.

Ограничения

$4 \leq n \leq 52$, $1 \leq b \leq 4$.

Формат входного файла

Первая строка входного файла содержит целые числа n и b.

Формат выходного файла

Выведите позиции расставленных вами ферзей, разделенные пробелами. Столбцы называются от ‘a’ до ‘z’, затем от ‘A’ до ‘Z’.

Пример

b.in	b.out
2 3	a1 b1 b2 a2

Пояснение

Счёт в данном примере: 4 (количество ферзей).

Разбор задачи B. Queens

Автор благодарит Павла Кунявского, Александра Грановского, Дмитрия Егорова и Алексея Стаха — участников соревнования, придумавших наилучшие схемы расстановки ферзей, известные автору на данный момент.

Наилучшие (с асимптотической точки зрения) решения для случаев различных значений b таковы:

- $b = 1$: известна расстановка n ферзей;
- $b = 2$: известна расстановка $2n - 5$ ферзей;
- $b = 3$: известна расстановка $2n$ ферзей;
- $b = 4$: известна расстановка $3n - 4$ ферзей;

Задача C. Tangents

Имя входного файла:	c.in
Имя выходного файла:	c.out
Ограничение по времени:	1 с
Ограничение по памяти:	64 Мб

В ряд выписаны $\mathrm{tg}(1)$, $\mathrm{tg}(2)$, …, $\mathrm{tg}(n)$.

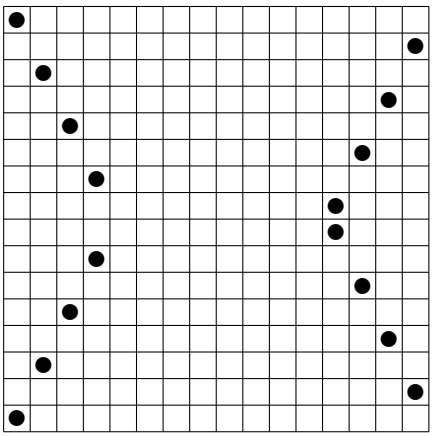


Рис. 2: Случай $b = 1$, расстановка n ферзей

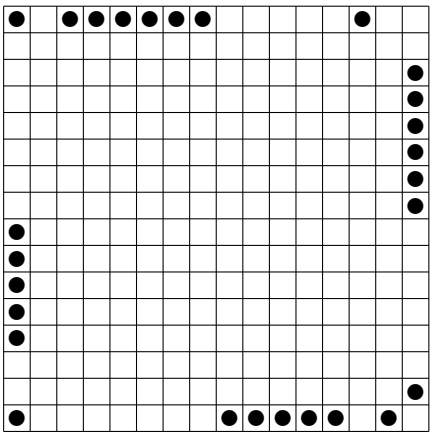


Рис. 3: Случай $b = 2$, расстановка $2n - 5$ ферзей

Поставьте между каждой парой соседних тангенсов знак ‘+’, ‘−’ или ‘*’ так, чтобы значение получившегося арифметического выражения было как можно ближе к числу n.

Ограничения

$5 \leq n \leq 365$.

Формат входного файла

Первая строка входного файла содержит целое число n.

Формат выходного файла

Выведите $n - 1$ расставленных вами арифметических знаков, без пробелов.

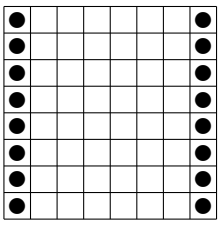


Рис. 4: Случай $b = 3$, расстановка $2n$ ферзей

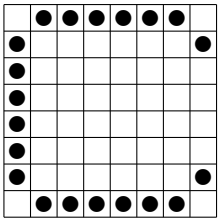


Рис. 5: Случай $b = 4$, расстановка $3n - 4$ ферзей

Пример

c.in	c.out
3	-+

Пояснение

Счёт в данном примере: 0.599901 (расстояние на числовой оси от значения выражения $(\mathrm{tg}(1) - \mathrm{tg}(2) + \mathrm{tg}(3))$ до числа $n = 3$).

Разбор задачи C. Tangents

Данная задача похожа по формулировке на задачу о рюкзаке (которая, кстати, является NP-полной).

Предложим подход к ее решению, аналогичный приближенной схеме решения задачи о рюкзаке.

После добавления каждого очередного тангенса (рассматриваем их слева направо) число возможных значений арифметического выражения утраивается (можно поставить один из трех знаков дейстий). Хранить все возможные значения 3^{n-1} не представляется возможным.

Предлагается хранить не всё множество достижимых значений, а лишь часть его. Дальше — простор для реализации.

Можно рассмотреть это множество, упорядоченное по возрастанию, и оставить из него лишь каждый k-й элемент, выбрав k так, чтобы в итоге множество вмещалось в отведенный объем оперативной памяти. Можно последовательно (с помощью структуры данных «куча», например), нахо-

дить пару соседних значений в этом множестве, разница между которыми как можно меньше, и удалять одно из них.

В конце, после рассмотрения $n - 1$ новых тангенсов, будет получено подмножество достижимых значений, в котором лишь следует найти величину, наиболее близкую к n.

Сергей Владимирович Копелиович также отмечает, что большую помощь может оказать использование метода «meet-in-the-middle». Есть предположение, что с его использованием, точность ответа может повыситься квадратично (если при фиксированном объеме оперативной памяти точность прямого прохода составить 10^{-q}, то точность «meet-in-the-middle» может составить до 10^{-2q}).

День третий (20.02.2012 г.)
Контест Неспирного Виталия и Лунева Антона

Об авторах...

Неспирный Виталий Николаевич, кандидат физико-математических наук (специальность 01.02.01 “Теоретическая механика”), докторант Института прикладной математики и механики НАН Украины, старший научный сотрудник отдела технической механики, ученый секретарь специализированного ученого совета по защите диссертаций на соискание ученой степени доктора наук (Д 11.193.01) по специальности 01.02.01, доцент кафедры теории упругости и вычислительной математики Донецкого национального университета, заместитель председателя жюри II-III этапов Всеукраинской олимпиады школьников по информатике в Донецкой области, председатель жюри по информатике Всеукраинской комплексной олимпиады по математике, физике и информатике “Турнир чемпионов”, координатор Донецкого сектора Открытого кубка по программированию, тренер команд математического факультета Донецкого национального университета.

Основные достижения:

- занял 1-е место на Всеукраинской студенческой олимпиады по информатике (Николаев, 2000);
- в составе команды математического факультета ДонНУ занял 1-е место на Всеукраинской ACM-олимпиады по программированию и 8-е в личном первенстве (Винница, 2001);
- подготовил призеров международных школьных олимпиад – Петр Луференко (2002), Роман Ризванов (2006), Вадим Янушкевич (2009);
- совместно с А.И.Парамоновым подготовил команду DonNU United, занявшую 7-е (2008), 4-е (2009) и 7-е место (2010) в ACM SEERC, а в 2011 году 8-е место (серебряные медали) в финале ACM ICPC.

Лунёв Антон Андреевич, аспирант Донецкого национального университета, специальность 01.01.01 “Математический анализ”.

Основные научные интересы:

- полнота и базисность систем корневых векторов некоторых классов обыкновенных дифференциальных операторов;
- точные константы в неравенствах для промежуточных производных;
- корневые решения обобщенного уравнения Маркова;
- Игра Грюнди и ее обобщения.

Основные достижения:

- трижды (2005, 2006, 2007) был награжден первым призом на международной математической олимпиаде для студентов IMC;
- в 2006, 2007 годах награжден серебряной медалью, а в 2008 - золотой, на международной студенческой математической олимпиаде в Иране;
- с 2005 по 2011 г. был участником команды Донецкого национального университета (Fortune, а затем United);
- на Всеукраинской студенческой олимпиаде по программированию (г. Винница) команда заняла 4-е (2009) и 2-е место (2010);
- в составе команды занял 7-е (2008), 4-е (2009) и 7-е место (2010) в ACM SEERC, а в 2011 году 8-е место (серебряная медаль) в финале ACM ICPC;
- по итогам 8-го Открытого Кубка (осень 2010) команда награждена дипломом первой степени;
- многократный призер серий индивидуальных соревнований по программированию SnarkNews Summer/Winter Series (Лето 2009 – 3-е место, Зима 2010 – 3-е место, Лето 2010 – 4-е место, Зима 2012 - 7-е место);

- участник онсайт-раунда Яндекс Open 2010, Topcoder Open 2011;
- автор ряда контестов на CodeChef.

В настоящий момент проходит стажировку в компании Facebook.

Теоретический материал. Конечные разности

В данной лекции мы попытаемся провести некоторые аналогии между стандартными понятиями математического анализа для функций заданных на непрерывных множествах с операциями над последовательностями (то есть функциями заданными только в целых точках). Наша цель – научиться вычислять различные суммы так же просто, как мы умеем выполнять интегрирование непрерывных функций. Большая часть материала взята из книги Р.Грэхема, Д.Кнута и О.Паташника “Конкретная математика”. Разумеется, в книге данный материал раскрыт более подробно с многочисленными конкретными примерами, содержит большое число упражнений и задач, которые было бы полезно попытаться решить. Здесь же представляется некоторая выжимка всего лишь одного из параграфов книги с несколько большим акцентом на конкретном методе суммирования с помощью конечных разностей.

Итак, если в математическом анализе мы как правило имеем дело с функциями $f(x)$, заданными на множестве вещественных чисел $i \in \mathbb{R}$, здесь мы рассматриваем бесконечные в две стороны последовательности a_i, где $i \in \mathbb{Z}$. В анализе вводится операция дифференцирования:

$$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h},$$

которая характеризует скорость изменения функции в каждой точке. Для последовательностей мы бы тоже могли рассмотреть подобное отношение, но, к сожалению, устремить h к нулю не удастся. Минимальное значение h, которое можно взять – это 1. Соответствующее значение будем называть конечной разностью и обозначать Δa:

$$\Delta a_i = a_{i+1} - a_i.$$

Собрав значения конечных разностей при различных i, получим некоторую последовательность, которую так же будем называть конечной разностью (путаницы не должно возникать равно, как в случае с производной: с одной стороны когда мы говорим “производная”, это может обозначать производную как функцию, с другой стороны значение производной в некоторой точке, как правило понятно, что имеется в виду в каждом конкретном случае из контекста, или может быть явно указано).

Приступая к изучению производных, прежде всего изучают чему она равна для степенной функции:

$$\frac{d}{dx} x^m = m x^{m-1}.$$

К сожалению, конечная разность для степенной последовательности уже не выражается так просто:

$$\Delta(i^m) = (i+1)^m - i^m = i^m + mi^{m-1} + \frac{m(m-1)}{2}i^{m-2} + \ldots + 1 - i^m =$$

$$= mi^{m-1} + \frac{m(m-1)}{2}i^{m-2} + \ldots + 1.$$

Однако можно видеть, что степень конечной разности от полинома является полиномом степени на 1 меньше. Все же существует некоторый аналог m-ой степени, для которой конечная разность выражается просто. Это так называемая нисходящая (или убывающая) степень, которая определяется по следующей формуле

$$x^{\underline{m}} = x(x-1)\ldots(x-m+1)$$

для любого целого неотрицательного m. Действительно, для последовательности таких степеней

$$\Delta i^{\underline{m}} = mi^{\underline{m-1}}$$

Если бы мы определили бы Δa_i не как прямую разность $a_{i+1} - a_i$, а как обратную – $a_i - a_{i-1}$ (то есть $h = -1$), то аналогичное свойство было бы справедливым уже для восходящей (или возрастающей) степени

$$x^{\overline{m}} = x(x+1)\ldots(x+m-1).$$

Обратной операцией к операции дифференцирования является интегрирование. Если нам задана функция $f(x)$, то мы называем первообразной для нее такую функцию $F(x)$, производная которой равна исходной, то есть $F'(x) = f(x)$. В нашем дискретном случае мы можем говорить о последовательности A_i, конечная разность от которой является заданной последовательностью a_i. Такую последовательность A_i мы будем называть антиразностью для a_i.

Основная теорема интегрального счисления утверждает, что если $F(x)$ – некоторая первообразная $f(x)$, то любая функция, представимая в виде $F(x) + C$, где C – некоторая постоянная, также является первообразной, и других первообразных быть не может. Этот факт отражается в виде формулы

$$\int f(x)dx = F(x) + C \Leftrightarrow F'(x) = f(x).$$

Аналогичная формула может быть записана для дискретного случая:

$$\sum a_i = A_i + C \Leftrightarrow \Delta A_i = a_i.$$

Здесь $\sum a_i$ – это неопределенная сумма для последовательности a_i, которая является классом всех возможных антиразностей последовательности a_i.

Перейдем теперь к определенному интегралу. Наиболее важное свойство, позволяющее его вычислять связано с формулой Ньютона-Лейбница:

$$\int_a^b f(x)dx = F(b) - F(a).$$

Аналогом в дискретном случае будет определенная сумма:

$$\sum_l^r a_i = A_r - A_l.$$

Однако, что обозначает $\sum_l^r$? Для того, чтобы сохранить полную аналогию с формулой Ньютона-Лейбница и гарантировать его справедливость, мы будем вынуждены исключить из суммы последний член, то есть $\sum_l^r$ следует понимать как $\sum\limits_{l\le i<r}$ при $l \le r$ либо как $-\sum\limits_{r\le i<l}$ при $r \le l$. В таком случае будем справедливым свойство

$$\sum_l^m a_i + \sum_m^r a_i = \sum_l^m a_i$$

при любых l, m и r.

Полученных свойств уже будет достаточно для того, чтобы найти сумму нисходящих степеней:

$$\sum_{0\le i<n} i^{\underline{m}} = \left.\frac{i^{\underline{m+1}}}{m+1}\right|_0^n = \frac{n^{\underline{m+1}}}{m+1}.$$

Эта формула будет справедливой и при отрицательных значениях m, если определить $i^{\underline{-k}}$ по формуле:

$$i^{\underline{-k}} = \frac{1}{(i+1)(i+2)\dots(i+k)}.$$

Однако все же при $m = -1$ мы будем иметь неопределенность. Равно как и в случае с интегралом от x^{-1} для которого получается специальная функция $\ln x$, для суммы стеней с показателем -1 вводится специальная последовательность:

$$H_i = \frac{1}{1} + \frac{1}{2} + \dots + \frac{1}{i}.$$

Теперь, когда найден аналог для $\ln x$, можем поискать аналог функции e^x. Основным свойством для нее является то, что ее прозводная равна опять же e^x. То есть мы хотим найти последовательность a_i, обладающую свойством $\Delta a_i = a_{i+1} - a_i = a_i$. Из последнего соотношения следует, что $a_{i+1} = 2a_i$. И очевидно последовательность $a_i = 2^i$ удовлетворяет полученной рекуррентности. То есть наиболее "натуральной" показательной последовательностью является 2^i.

В случае произвольного основания c степени имеем

$$\Delta c^i = (c-1)c^i,$$

что дает нам возможность вычисления суммы геометрической прогрессии:

$$\sum_{a \le i < b} c^i = \frac{c^i}{c-1}\Big|_a^b = \frac{c^b - c^a}{c-1}.$$

Рассмотрим еще одну аналогию. В анализе есть формула для вычисления производной произведения функций. Выведем ее аналог в дискретном случае:

$$\Delta(u_i v_i) = u_{i+1}v_{i+1} - u_i v_i = u_{i+1}v_{i+1} - u_i v_{i+1} + u_i v_{i+1} - u_i v_i =$$

$$= u_i \Delta v_i + v_{i+1}\Delta u_i = u_i \Delta v_i + E v_i \Delta u_i.$$

Здесь под E понимается оператор сдвига последовательности влево:

$$(Ev)_i = v_{i+1}.$$

В связи с этой формулой мы можем получить правило суммирования по частям. Если под суммой стоит произведение двух последовательностей, для одной из которых мы знаем антисумму, то для ее вычисления может быть в некоторых случаях полезна формула:

$$\sum u \Delta v = uv - \sum Ev \Delta u.$$

Заметим, что не все понятия непрерывного анализа имеют аналоги в дискретном случае. Например, правило дифференцирования сложных функции не переносится на дискретных случай, уже хотя бы потому, что далеко не всегда имеет смысл выражение a_{b_i}. По этой причине весьма затруднена замена переменных, за исключением некоторых простейших случаев замены типа i на $c \pm i$.

Последнее, что мы разберем (и чего нет в "Конкретной математике") – это разности более высоких порядков, и связанный с ними аналог формулы Тейлора.

Поскольку конечная разность для последовательности – это тоже последовательность. Значит для нее может быть также вычислена ее конечная разность. Это будет разность второго порядка:

$$\Delta^2 a_i = \Delta a_{i+1} - \Delta a_i = a_{i+2} - 2a_{i+1} + a_i.$$

Действуя таким образом дальше, мы можем получить разность любого порядка:

$$\Delta^n a_i = \Delta^{n-1} a_{i+1} - \Delta^{n-1} a_i.$$

В анализе есть формула Тейлора, которая выражает значение функции в произвольной точки через ряд, в который входят значения функции и ее производных в некоторой фиксированной точке. Для простоты будем считать, что эта фиксированная точка – 0. Тогда

$$f(x) = \sum_{k=0}^{\infty} \frac{f^{(k)}(0)}{k!} x^k.$$

Аналогично в дискретном случае мы можем записать следующий ряд

$$a_i = \sum_{k=0}^{\infty} \frac{\Delta^k a_0}{1^{\underline{k}}} i^{\underline{k}}.$$

Если оставить лишь несколько (скажем $m + 1$) первых членов этого ряда, мы получим не что иное, как интерполяционную формулу Ньютона. То есть мы сможем получить значение a_i, но с некоторой погрешностью. Очевидно интерполяционная формула будет точна для любых многочленов степени не выше m (поскольку любая разность порядка $m + 1$ и выше для такого многочлена будет равна нулю).

Кроме того, при правильном порядке вычислений мы сможем найти значение многочлена степени m в точке k за $O(km)$ операций сложения, если будем знать разности $\Delta^k a_0$ для $k = \overline{0, m}$. Для этого достаточно пересчитать значения разностей этих порядков в точке 1 (за $O(m)$ сложений), затем в точке 2 и т.д. Это может быть полезным в случае, когда операция умножения для чисел, над которыми выполняются вычисления, осуществляется на порядок дольше, чем сложение.

Задачи и разборы

Задача A. СуперНим High (только для высшей лиги)

Имя входного файла:	`a.in`
Имя выходного файла:	`a.out`
Ограничение по времени:	2 с
Ограничение по памяти:	8 Мб

Ним – это игра для двух игроков, которые по очереди берут предметы, разложенные на несколько кучек. Каждый игрок в свой ход может выбрать одну из непустых кучек и забирать из нее любое ненулевое количество предметов. Проигрывает тот из игроков, кто в свою очередь не сможет сделать ход (то есть последний предмет заберет на предыдущем ходе его соперник).

Данная игра математически полностью проанализирована. Проигрышными позициями в игре называются такие позиции, что при любых дальнейших ходах игрока, который должен делать ход из данной позиции, его соперник имеет возможность делать такие ходы, которые приведут к проигрышу игрока (то есть соперник имеет выигрышную стратегию). Выигрышными – такие, что, каковы бы ни были в дальнейшем действия соперника, игрок, который выполняет ход из данной позиции, имеет возможность делать такие ходы, которые приведут его к победе (то есть игрок имеет выигрышную стратегию). Известно, что выигрышность позиции определяется ним-суммой: $p_1 \operatorname{xor} p_2 \operatorname{xor} \ldots \operatorname{xor} p_N$, где p_i – количество предметов в i-ой кучке, xor – операция побитового исключающего “или” (сложение двоичных представлений чисел без учета переносов в следующие разряды). Если эта сумма отлична от нуля, то позиция выигрышная, если равна нулю – проигрышная.

Здесь мы будем иметь дело с огромным количеством кучек. Ваша задача – по заданным размерам кучек определить количество выигрышных вариантов первого хода первого игрока из начальной позиции (то есть такие ходы, при которых выигрышная стратегия все еще будет оставаться у первого игрока).

Формат входного файла

В первой строке входного файла содержится целое число M, определяющее количество групп кучек. Каждая из последующих M строк описывает соответствующую группу с помощью трех целых чисел a, b, N. Если расположить все кучки в группе в порядке неубывания размеров, то a – размер наименьшей кучки, b – размер наибольшей, N – количество кучек в группе, а разность между размерами двух соседних кучек постоянна.

Ограничения

$1 \le M \le 20000$, $0 \le a \le b \le 10^{18}$, $1 \le N \le 10^{18}$, $b - a$ кратно $N - 1$.

Формат выходного файла

Выведите единственное целое число – количество вариантов первого хода, которое допускает выигрышная стратегия.

Примеры

a.in	a.out
2 3 5 3 2 6 3	3
2 1 6 6 8 15 2	0

Задача B. СуперНим Junior (только для юниорской лиги)

Имя входного файла:	b.in
Имя выходного файла:	b.out
Ограничение по времени:	1 с
Ограничение по памяти:	8 Мб

Ним – это игра для двух игроков, которые по очереди берут предметы, разложенные на несколько кучек. Каждый игрок в свой ход может выбрать одну из непустых кучек и забирать из нее любое ненулевое количество предметов. Проигрывает тот из игроков, кто в свою очередь не сможет сделать ход (то есть последний предмет заберет на предыдущем ходе его соперник).

Данная игра математически полностью проанализирована. Проигрышными позициями в игре называются такие позиции, что при любых дальнейших ходах игрока, который должен делать ход из данной позиции, его соперник имеет возможность делать такие ходы, которые приведут к проигрышу игрока (то есть соперник имеет выигрышную стратегию). Выигрышными – такие, что, каковы бы ни были в дальнейшем действия соперника, игрок, который выполняет ход из данной позиции, имеет возможность делать такие ходы, которые приведут его к победе (то есть игрок имеет выигрышную стратегию). Известно, что выигрышность позиции определяется ним-суммой: $p_1 \operatorname{xor} p_2 \operatorname{xor} \ldots \operatorname{xor} p_N$, где p_i – количество предметов в i-ой кучке, xor – операция побитового исключающего “или” (сложение двоичных представлений чисел без учета переносов в следующие разряды). Если

эта сумма отлична от нуля, то позиция выигрышная, если равна нулю – проигрышная.

Здесь мы будем иметь дело с огромным количеством кучек. Ваша задача – по заданным размерам кучек определить количество выигрышных вариантов первого хода первого игрока из начальной позиции (то есть такие ходы, при которых выигрышная стратегия все еще будет оставаться у первого игрока).

Формат входного файла

В первой строке входного файла содержится целое число M, определяющее количество групп кучек. Каждая из последующих M строк описывает соответствующую группу с помощью двух целых чисел a, b. В группе будет $b - a + 1$ кучек с размерами от a до b включительно.

Ограничения

$1 \leq M \leq 100000$, $0 \leq a \leq b \leq 10^{12}$.

Формат выходного файла

Выведите единственное целое число – количество вариантов первого хода, которое допускает выигрышная стратегия.

Примеры

b.in	b.out
2 3 5 2 6	5
2 1 4 11 14	0

Разбор задачи СуперНим

Рассмотрим для начала обычный вариант игры Ним, когда количество кучек не слишком велико. Очевидно, что в случае, когда исходная позиция является проигрышной, у игрока не должно быть ни одного выигрышного хода. Если исходная позиция выигрышна, то выигрышными будут те и только те ходы, которые ведут в позиции, являющиеся проигрышными (для второго игрока, который будет вынужден делать ход из них), то есть позиции с ним-суммой 0. Пусть ним-сумма исходной позиции равна x и мы

делаем некоторый ход в кучке i, то есть изначально в ней было a_i предметов, стало $a'_i < a_i$, количество предметов в остальных кучках осталось без изменений. При этом новая ним-сумма должна равняться 0:

$$a_1 \oplus \ldots \oplus a_{i-1} \oplus a'_i \oplus a_{i+1} \oplus \ldots \oplus a_N = 0.$$

Добавим к обеим частям этого равенства a_i (т.е. применим операцию xor с указанным значением) и перегруппируем слагаемые:

$$a'_i \oplus a_1 \oplus \ldots \oplus a_N = a_i.$$

Но $a_1 \oplus \ldots \oplus a_N$ – это ним-сумма исходной позиции и она равна x. Учитывая это и свойство $x \oplus x = 0$, после добавления x к обеим частям равенства, получим:

$$a'_i = a_i \oplus x.$$

Таким образом, если в кучке i существует выигрышный ход, то он единственный – необходимо брать из нее $a_i - (a_i \oplus x)$ предметов. Хода не существует, если указанная величина будет отрицательной (нулевой эта величина быть не может при $x \neq 0$). Нетрудно доказать, что указанная величина будет положительной тогда и только тогда, когда a_i имеет включенным (равным 1) бит, который является самым старшим из включенных в числе x (обозначим номер этого бита l). Действительно будем сравнивать побитово числа a_i и $a_i \oplus x$, начиная со старших битов. Пока мы идем по битам с номерами большими l, отличий между a_i и $a_i \oplus x$ не обнаружится (поскольку число x в этих битах содержит 0 и значит операция $\oplus x$ ничего не изменяет в a_i). Теперь переходим к биту l. Так как этот бит включен в x, то применение операции $\oplus x$ изменяет значение этого бита в a_i. Если бит l числа a_i был равен 0, то в числе $a_i \oplus x$ он станет равен 1. Это означает, что $a_i \oplus x > a_i$ и мы не можем сделать выигрышный ход в кучке i. Если же бит l числа a_i был равен 1, то в числе $a_i \oplus x$ он станет равен 0, то есть $a_i \oplus x < a_i$ и имеется ровно один выигрышный ход в данной кучке.

Итак, пусть l – самый старший включенный бит ним-суммы x, или что то же самое $2^l \leq x < 2^{l+1}$. Тогда количество выигрышных ходов из позиции $(a_1, a_2, \ldots, a_N)$ – это количество таких i, что a_i имеет включенным бит l (или $a_i \operatorname{and} 2^l \neq 0$).

Значит для решения задачи, нужно быстро вычислять количество кучек группы, размеры которых имеют включенным некоторый бит. Если мы сделаем это для каждого бита, то сможем определить и ним-сумму (некоторый ее бит будет включенным тогда и только тогда, когда количество кучек с включенным этим битом нечетно).

Рассмотрим для начала вариант Junior задачи, в котором каждая группа представляла собой множество всех целых чисел от a до b. Очевидно, что

если мы вычислим количество куч с нужным битом для чисел от 0 до b и вычтем количество куч с тем же битом для чисел от 0 до $a-1$, то это и будет требуемым количеством. (Обратите особое внимание на случай $a = 0$!). Таким образом, мы еще немного упростили задачу – найти количество чисел с включенным битом i в диапазоне от 0 до b.

Разобьем все эти числа на группы по 2^{i+1} элементов: $0 \dots 2^{i+1}-1$, $2^{i+1} \dots 2 \cdot 2^{i+1}-1$, ... Всего таких групп будет $k = \lfloor b/2^{i+1} \rfloor$. И возможно, будет еще одна неполная группа, в которой будут находиться числа $k \cdot 2^{i+1} \dots b$. В каждой полной группе, очевидно, первая половина элементов имеют выключенным бит i, вторая половина включенным. Значит, из полных групп у нас набирается $k \cdot 2^i$ чисел с включенным битом i. В неполной группе необходимые нам числа будут лишь в том случае, если в ней больше, чем 2^i элементов. В таком случае будет в точности $b - k \cdot 2^{i+1} + 2^i - 1$ чисел, с включенным битом i. Подзадача решена за $O(1)$. Остается вычислять ее решения для каждой группы и каждого бита. Итоговая сложность будет $M(\log a + \log b)$.

Перейдем теперь к варианту High задачи. Здесь мы имеем дело с группой вида a, $a+z$, ..., $a+Nz$, где $z = (b-a)/(N-1)$. Наша главная подзадача как и прежде – как можно быстрее определить, сколько из этих чисел имеют включенным бит i. Можно сформулировать задачу и чуть иначе – сколько чисел из указанных имеют при делении на 2^{i+1} остаток, принадлежащий отрезку $[2^i, 2^{i+1}-1]$. Немного обобщим эту задачу: пусть требуется вычислить count(z, m, l, r, N) – количество чисел среди 0, z, $2z$, ..., Nz, которые при делении на m дают остаток из отрезка $[l, r]$. Мы будем считать, что $0 < z < m$, поскольку при $z = 0$ решение тривиально, а при $zgem$ ничего не изменится, если заменить z на $z \bmod m$. Ясно, что исходная задача сводится к обобщенной за счет некоторого сдвига отрезка. Однако следует заметить, что этот отрезок может разбиться на два отрезка вида $[0, r]$ и $[l, m-1]$. В этом случае есть смысл вычесть из количества всех чисел $(N+1)$, количество тех, которые принадлежат отрезку $[r+1, l-1]$.

Решаем задачу вычисления count(z, m, l, r, N). Узнаем количество таких чисел iz ($i = \overline{0, N}$), которые принадлежат отрезкам $[l, r]$, $[l+m, r+m]$, $[l+2m, r+2m]$ и т.д. Последний полный отрезок $[l+km, r+km]$, в который могут попадать числа iz должен удовлетворять условию $r + km \le Nz < r + (k+1)m$, откуда следует, что $k = \lfloor (Nz-r)/m \rfloor$. Кроме того, возможно попадание наших чисел в отрезок $[l + (k+1)m, Nz]$ при условии, что $l + (k+1)m \le Nz$. Количество чисел, кратных z на отрезке $[a, b]$ можно вычислить по формуле $\lfloor b/z \rfloor - \lfloor (a-1)/z \rfloor$. Сразу же вычислим количество таких чисел на неполном отрезке, если он имеется. Остается разобраться

с суммой

$$S = \sum_{0 \le i \le k} \left(\left\lfloor \frac{r + im}{z} \right\rfloor - \left\lfloor \frac{l + im - 1}{z} \right\rfloor \right).$$

Преобразуем суммируемое выражение. Из определения остатка от деления следует, что $\lfloor x/y \rfloor = (x - x \bmod y)/y$. Пользуясь этим соотношением, получаем

$$S = \sum_{0 \le i \le k} \left(\frac{r + im}{z} - \frac{l + im - 1}{z} - \frac{(r + im) \bmod z}{z} + \frac{(l + im - 1) \bmod z}{z} \right).$$

и далее

$$S = \sum_{0 \le i \le k} \left(\frac{r - l + 1}{z} + \frac{(l + im - 1) \bmod z - (r + im) \bmod z}{z} \right).$$

Заметим, что вторая дробь лежит строго в интервале $(-1, 1)$. И кроме того, сумма этих двух дробей должна быть целой (исходное выражение было целым). Таким образом, если мы округлим первую дробь в меньшую сторону, то ошибемся лишь в том случае когда вторая дробь будет положительной. То есть основная часть этой суммы будет равна $(k + 1)\lfloor (r - l + 1)/z \rfloor$, а добавка – количеству чисел i от 0 до k, при которых $(l + im - 1) \bmod z > (r + im) \bmod z$. Нетрудно понять, что это будет лишь тогда, когда числа im попадают в некоторый отрезок $[l', r']$ (читателю в качестве упражнения предлагается самостоятельно проделать необходимые вычисления и получить этот отрезок, мы дадим лишь окончательный ответ). В случае $(l - 1) \bmod z < r \bmod z$ границы отрезка определяются равенствами $l' = z - r \bmod z$, $r = z - 1 - (l - 1) \bmod z$, в случае $(l - 1) \bmod z > r \bmod z$ получается два отрезка $[0, z - 1 - (l - 1) \bmod z]$ и $[z - r \bmod z, z - 1]$, дополнением к которым до полного отрезка остатков $[0, m - 1]$ является отрезок $[z - (l - 1) \bmod z, z - 1 - r \bmod z]$. На этом отрезке и будет проводиться дальнейший подсчет. При равенстве $(l - 1) \bmod z = r \bmod z$ требуемое неравенство $(l + im - 1) \bmod z > (r + im) \bmod z$, очевидно, не выполняется никогда. Таким образом, нахождение добавки свелось к вычислению count($m \bmod z$, z, l', r', k). В случае $(l - 1) \bmod z > r \bmod z$ для получения добавки нужно будет вычесть полученное значение из $k + 1$.

Нетрудно заметить, что величины $m \bmod z$ и z определяют те числа, которые возникают при переходе к следующему шагу алгоритма Евклида, который работает за $O(\log m)$. Значит наши вычисления для решения основной подзадачи потребуют $O(\log b)$ операций. И тогда все решение будет иметь сложность $O(M \log^2 b)$.

Замечание. Следует быть предельно осторожным при вычислении значений типа $(l-1) \bmod z$ и $k = \lfloor (Nz-r)/m \rfloor$, поскольку для отрицательных чисел машинная целочисленная арифметика деления с остатком для отрицательных чисел не согласуется с естественным математическим определением этих операций. Поэтому первое выражение следует вычислять как $(l + z - 1) \bmod z$, а второе – $k = (Nz - r + m)\mathrm{div}(m - 1)$.

Задача C. Хромой король High (только для высшей лиги)

Имя входного файла:	`c.in`
Имя выходного файла:	`c.out`
Ограничение по времени:	1 с
Ограничение по памяти:	8 Мб

Рассмотрим бесконечную во все четыре стороны шахматную доску с квадратными клетками. Некоторую клетку этой доски назовем центром. Вертикали, которая проходит через центр, присвоим номер 0. Вертикалям, находящимся правее центральной, присвоим последовательно номера 1, 2, 3 и т.д., левее – -1, -2, -3 и т.д. Аналогично пронумеруем горизонтали (выше центральной – положительными числами, ниже – отрицательными). Координаты любой клетки тогда можно определить парой чисел – номером вертикали и номером горизонтали, в которой она находится. Пусть теперь в центре доски, то есть в клетке с координатами $(0, 0)$, стоит король. Он может перемещаться по стандартным шахматным правилам – в соседнюю клетку по горизонтали, вертикали или диагонали. Однако, некоторые из направлений являются запрещенными.

Требуется определить, за сколько ходов король сможет попасть в клетку с заданными координатами (x, y).

Формат входного файла

В первой строке входного файла заданы 8 чисел, определяющих возможность перемещения в соответствующем направлении. 1 обозначает разрешенное направление, 0 – запрещенное. Направления перечисляются в порядке обхода против часовой стрелки, начиная с положительного горизонтального (то есть вправо, вправо-вверх, вверх, влево-вверх, влево, влево-вниз, вниз, вправо-вниз). Во второй строке задаются координаты x и y клетки, в которую необходимо попасть.

Ограничения

$-10^{18} \le x, y \le 10^{18}$.

Формат выходного файла

Выведите единственное целое число – минимальное количество разрешенных ходов, которые потребуются королю для того, чтобы добраться из клетки $(0, 0)$ в клетку (x, y). В случае, если такого пути не существует, выведите число -1.

Примеры

c.in	c.out
1 1 1 1 1 1 1 1 3 4	4
1 0 1 0 1 0 1 0 -3 -4	7

Задача D. Хромой король Junior (только для юниорской лиги)

Имя входного файла:	d.in
Имя выходного файла:	d.out
Ограничение по времени:	1 с
Ограничение по памяти:	32 Мб

Рассмотрим бесконечную во все четыре стороны шахматную доску с квадратными клетками. Некоторую клетку этой доски назовем центром. Вертикали, которая проходит через центр, присвоим номер 0. Вертикалям, находящимся правее центральной, присвоим последовательно номера 1, 2, 3 и т.д., левее – -1, -2, -3 и т.д. Аналогично пронумеруем горизонтали (выше центральной – положительными числами, ниже – отрицательными). Координаты любой клетки тогда можно определить парой чисел – номером вертикали и номером горизонтали, в которой она находится. Пусть теперь в центре доски, то есть в клетке с координатами $(0, 0)$, стоит король. Он может перемещаться по стандартным шахматным правилам – в соседнюю клетку по горизонтали, вертикали или диагонали. Однако некоторые из направлений являются запрещенными.

Требуется определить за сколько ходов король сможет попасть в клетку с заданными координатами (x, y).

Формат входного файла

В первой строке входного файла заданы 8 чисел, определяющих возможность перемещения в соответствующем направлении. 1 обозначает разрешенное направление, 0 – запрещенное. Направления перечисляются в

порядке обхода против часовой стрелки, начиная с положительного горизонтального (то есть вправо, вправо-вверх, вверх, влево-вверх, влево, влево-вниз, вниз, вправо-вниз). Во второй строке задаются координаты x и y клетки, в которую необходимо попасть.

Ограничения

$-500 \leq x, y \leq 500$.

Формат выходного файла

Выведите единственное целое число – минимальное количество разрешенных ходов, которые потребуются королю для того, чтобы добраться из клетки $(0, 0)$ в клетку (x, y). В случае, если такого пути не существует, выведите число -1.

Примеры

d.in	d.out
1 1 1 1 1 1 1 1 3 4	4
1 0 1 0 1 0 1 0 -3 -4	7

Разбор задачи Хромой король

Для решения варианта Junior задачи достаточно запустить стандартный алгоритм поиска в ширину на квадрате от $-S$ до S по оси Ox и от $-S$ до S по оси Oy, где $S = |x_d| + |y_d| + 1$, где (x_d, y_d) – целевая клетка, используя лишь переходы в разрешенных направлениях. Очевидно (хотя строгое доказательство этого факта быть может не вполне тривиальным), что если некоторый путь в клетку (x_d, y_d) выходит за пределы этого квадрата, то он не может быть оптимальным. Можно еще ограничить множество, на котором будет выполняться поиск в ширину, до клеток, удовлетворяющих неравенству $|x| + |y| \leq S$. Однако, при указанных в задаче ограничениях любая разумная реализация поиска в ширину даже для первого варианта множества легко уложится в отведенное время.

Перейдем теперь к варианту High. Ясно, что не имеет значения порядок ходов – на доске у нас нет запрещенных клеток, а от перемены мест слагаемых сумма (даже для векторов) не изменяется. Поэтому не важно когда мы будем делать тот или иной ход, важно сколько ходов в каждом разрешенном направлении мы выполним.

Рассмотрим для начала задачу о том, можно ли вообще переместиться в заданном направлении на одну или на две клетки, используя быть может несколько разрешенных ходов (важно заметить, что если в некотором направлении мы не можем переместиться ни на одну, ни на две клетки, то не сможем двигаться в этом направлении совсем). Для этого согласно идее задачи Junior нам нужно предварительно запустить поиск в ширину на квадрате $[-3, 3] \times [-3, 3]$, запомнив полученные значения.

Теперь для перемещения в требуемую клетку (x_d, y_d) воспользуемся какими-либо двумя (необязательно разрешенными) направлениями движения (пусть одно из них задается вектором (dx_1, dy_1), а другое – (dx_2, dy_2), где все координаты имеют значения от -1 до 1). Найдем такие значения k_1 и k_2, что переместившись на k_1 клеток в первом направлении и на k_2 клеток во втором направлении, попадем в клетку назначения. Эти числа должны удовлетворять системе уравнений:

$$dx_1 k_1 + dx_2 k_2 = x_d,$$
$$dy_1 k_1 + dy_2 k_2 = y_d.$$

Определитель этой системы будет отличен от нуля для любой пары различных направлений, за исключением противоположных. Такую пару направлений мы не будем использовать никогда, поэтому решение будет найдено однозначно. Определим теперь сколько нам потребуется ходов, чтобы переместиться на k_1 клеток в первом направлении. Если $k_1 < 0$, то мы вообще не можем переместиться, или будем считать, что требуется бесконечное число ходов. Если $k_1 = 0$, то нам не нужно перемещаться и количество ходов равно 0. Если же $k_1 > 0$, то воспользуемся $\lfloor k_1/2 \rfloor$ раз последовательностью ходов, которая перемещает нас из клетки $(0, 0)$ в $(2dx_1, 2dy_1)$, длина которой у нас уже была предподсчитана. Если такое перемещение невозможно, но $\lfloor k_1/2 \rfloor \neq 0$, то можем снова выдать бесконечность. Если же удалось переместиться, то возможно потребуется еще одно перемещение (если $k_1 \bmod 2 \neq 0$) на одну клетку. Для этого нужно знать длину кратчайшей последовательности ходов из $(0, 0)$ в (dx_1, dy_1). Проделав аналогичные действия для второго направления, получим в сумме с предыдущим значением количество ходов, необходимых для попадания в клетку (x_d, y_d) при использовании пары заданных направлений. Остается теперь лишь перебрать все возможные пары таких направлений, исключая взаимнопротивоположные, и выбрать лучший результат.

Задача E. Изменение на отрезке High (только для высшей лиги)

Имя входного файла:	`e.in`
Имя выходного файла:	`e.out`
Ограничение по времени:	1 с
Ограничение по памяти:	8 Мб

Задан набор из N целых чисел $a_0, a_1, \ldots, a_{N-1}$. Изначально все эти числа равны 0. Далее поступают запросы на изменение и вывод. Для запроса на изменение задаются три числа l, r, d. По этому запросу к каждому из элементов a_i ($l \le i \le r$) необходимо прибавить значение d. Для запроса на вывод задается одно число i. По этому требуется вывести текущее значение элемента a_i.

Формат входного файла

В первой строке входного файла задается два целых числа N и M, обозначающих количество элементов и количество запросов соответственно. В последующих M строках задаются запросы. Запрос на изменение задается строкой вида "`A` *l r d*", запрос на вывод – строкой "`Q` *i*".

Ограничения

Все числа целые.
$1 \le N \le 10^6$, $0 \le M \le 10^6$, $0 \le l \le r < N$, $0 \le i < N$, $|d| \le 10^3$.

Формат выходного файла

Для каждого запроса на вывод выведите в отдельной строке текущее значение соответствующего элемента.

Пример

e.in	e.out
10 6 A 3 7 1 Q 4 A 1 5 2 Q 4 Q 1 Q 6	1 3 2 1

Задача F. Изменение на отрезке Junior (только для юниорской лиги)

Имя входного файла:	`f.in`
Имя выходного файла:	`f.out`
Ограничение по времени:	1 с
Ограничение по памяти:	8 Мб

Задан набор из N целых чисел $a_0, a_1, \ldots, a_{N-1}$. Изначально все эти числа равны 0. Далее поступают запросы на изменение и вывод. Для запроса на изменение задаются три числа l, r, d. По этому запросу к каждому из элементов a_i ($l \le i \le r$) необходимо прибавить значение d. Для запроса на вывод задается одно число i. По этому требуется вывести текущее значение элемента a_i.

Формат входного файла

В первой строке входного файла задается три целых числа N, M_A и M_Q, обозначающих количество элементов, количество запросов на изменение и на вывод соответственно. В последующих $M_A + M_Q$ строках задаются запросы. Запрос на изменение задается строкой вида "`A` l r d", запрос на вывод – строкой "`Q` i".

Ограничения

Все числа целые.
$1 \le N \le 10^6$, $0 \le M_A, M_Q \le 10^6$, $0 \le l \le r < N$, $0 \le i < N$, $|d| \le 10^3$.
Гарантируется, что по крайней мере одно из чисел N, M_A или M_Q не будет превосходить 50.

Формат выходного файла

Для каждого запроса на вывод выведите в отдельной строке текущее значение соответствующего элемента.

Пример

f.in	f.out
10 2 4 A 3 7 1 Q 4 A 1 5 2 Q 4 Q 1 Q 6	1 3 2 1

Разбор задачи Изменение на отрезке

Задача могла бы быть решена с помощью классического дерева отрезков с операцией изменения на отрезке, выполняя запросы сверху и сохраняя невязки (подробнее смотрите, например, лекцию А.С. Станкевича на Зимней школе 2008 года). Все запросы выполнялись бы за $O(logN)$, что вполне удовлетворительно по времени. Но, к сожалению, у нас слишком маленькое ограничение на память – едва ли его хватило бы даже для сохранения значений в полном дереве (не говоря уже о невязках). Однако можно заметить, что запросы на получение у нас происходят не на отрезке, а всего лишь для одного элемента. Попробуем изменить представление данных так, чтобы запрос на изменение работал с фиксированным количеством элементов, но, быть может, запрос на получение будет теперь обращаться к отрезку.

Такое представление легко получить, если воспользоваться конечными разностями (в нашем случае удобнее взять обратные разности). То есть будем хранить значения $\Delta a_i = a_i - a_{i-1}$ (полагаем, что есть фиктивный элемент a_{-1}, равный 0). Тогда при запросе на изменение на отрезке $[l, r]$ изменяются лишь два значения: Δa_l, которое увеличивается на d, и Δa_{r+1}, которое уменьшается на d (будьте внимательны с запросами, у которых $r = N - 1$). Запрос же на получение элемента a_i будет в терминах разностей сводиться к нахождению суммы величин Δa на отрезке $[0, i]$. Таким образом, мы можем использовать дерево отрезков на сумму для N элементов Δa, но лучше (чтобы не удваивать количество требуемой памяти) воспользоваться деревом Фенвика. Сложность алгоритма составит $O(M \log N)$.

В варианте Junior задачи при $N \leq 50$ или $M_A \leq 50$ можно не использовать специальных структур, а все действия выполнять на обычном массиве, в котором хранятся элементы a_i. Запрос на изменение будет выполняться за $O(N)$, но зато запросы на вывод (которых много) за $O(1)$ и общее время работы будет иметь порядок $O(M_A N + M_Q)$. Для случая $M_Q \leq 50$ достаточно будет просто перейти к разностному представлению Δa и выполнять запросы на изменение за $O(1)$, а запросы на вывод (то есть получение суммы начального отрезка разностей) за $O(1)$. Время работы в этом случае составит $O(M_A + M_Q N)$.

Задача G. K-цифровое число High (только для высшей лиги)

Имя входного файла:	`g.in`
Имя выходного файла:	`g.out`
Ограничение по времени:	1 с
Ограничение по памяти:	8 Мб

Назовем число K-цифровым, если количество различных цифр в его десятичной записи (без учета незначащих ведущих нулей) не превышает K.

По заданному числу x найдите ближайшее к нему K-цифровое число.

Формат входного файла

В единственной строке задается два целых числа K и x без незначащих ведущих нулей.

Ограничения

$1 \le K \le 10$, $0 \le x \le 10^{10^6}$.

Формат выходного файла

Выведите такое K-цифровое число y, чтобы величина $|y-x|$ имела минимально возможное значение. Если таких чисел несколько, можно выводить любое из них.

Примеры

g.in	g.out
2 23456	23333
1 8691	8888

Задача H. K-цифровое число Junior (только для юниорской лиги)

Имя входного файла:	`h.in`
Имя выходного файла:	`h.out`
Ограничение по времени:	1 с
Ограничение по памяти:	8 Мб

Назовем число K-цифровым, если количество различных цифр в его десятичной записи (без учета незначащих ведущих нулей) не превышает K.

По заданному числу x найдите ближайшее к нему K-цифровое число.

Формат входного файла

В единственной строке задается два целых числа K и x без незначащих ведущих нулей.

Ограничения

$1 \le K \le 10$, $0 \le x \le 10^8$.

Формат выходного файла

Выведите такое K-цифровое число y, что величина $|y - x|$ имеет минимально возможное значение. Если таких чисел несколько, можно выводить любое из них.

Примеры

h.in	h.out
2 23456	23333
1 8691	8888

Разбор задачи K-цифровое число

Вариант Junior этой задачи мог бы быть решен простым перебором. Если исходное число x является K-цифровым, то его и следует выдать. Если же это не так, то следует проверить значения $x-1$ и $x+1$. Если хотя бы одно из этих чисел удовлетворяет условию задачи, то следует выдать его в качестве ответа. В противном случае будем проверять значения $x-2$ и $x+2$ и т.д. Оценим время работы такого алгоритма. Пусть число n – количество цифр в числе x, а d – его старшая цифра. Очевидно, что число $y' = d \cdot \sum_{0 \le i \le n-1} 10^i$, состоящее из n цифр d, является одноцифровым (а значит и K-цифровым при любом K). Значит ближайшее число K-цифровое число y^* будет отличаться от x не более, чем на $|y' - x|$, что легко оценить величиной 10^{n-1}. Проверка же того, сколько различных цифр есть в конкретном числе, потребует времени порядка $n+10$. Поскольку величина n сравнима с $\log_{10} x$, общая сложность алгоритма оценивается величиной $O(x \log x)$.

Такой алгоритм, разумеется, не пройдет при решении варианта High задачи. Нужна идея, которая позволила бы решить задачу примерно за длину числа, то есть за $O(\log x)$. Попробуем найти ближайшее K-цифровое число r, не меньшее x, и ближайшее K-цифровое число l, не превышающее x. Тогда останется выбрать наилучший вариант из этих двух, что можно

сделать если вычислить знак выражения $(r-x)-(x-l)=r+l-2x$. Если указанная величина будет положительной, то в качестве ответа следует взять l, если отрицательной – r, если нулевой – любое из чисел l и r является верным ответом.

Укажем алгоритм определения числа r. Будем проходить по цифрам числа x, начиная со старших, к младшим. Ясно, что чем больше оригинальных старших цифр числа x мы сохраним в r, тем на меньшую величину они будут отличаться. Итак, проходим по цифрам x, сохраняя информацию о том, какие цифры у нас встречались и сколько раз (как увидим в дальнейшем, количество тоже важно). Если мы дошли до конца числа и количество различных встретившихся цифр не превзошло K, можем утверждать, что $r=x$. Если же на какой-то позиции (пусть это i) встретилась $(K+1)$-ая цифра (пусть она равна d), то по крайней мере с этого места r должно отличаться от x. Ясно, что на этом месте должна быть одна из тех K цифр, которые встречались раньше, которые больше d. Очевидно, наиболее выгодным вариантом является наименьшая из таких цифр (если, конечно, есть хотя бы одна такая). Поставив эту цифру в позицию i числа r, а во все позиции за i наименьшую из всех K ранее встречавшихся (без ограничений), мы получим минимально возможное число.

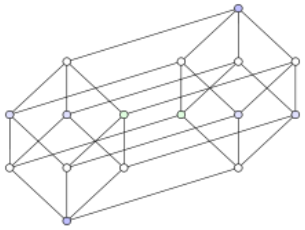
Теперь рассмотрим вариант, когда d больше всех цифр, встречавшихся до сих пор (до позиции i). В этом случае, мы уже не можем сохранить все цифры до позиции i (то есть на позициях от 1 до $i-1$), а значит следует попробовать изменить цифру, которая стоит на позиции $i-1$: пусть теперь d – цифра, стоящая на позиции $i-1$. Тогда прежде всего мы уменьшаем хранимое количество цифр d. При этом может оказаться, что она и была всего одна. Значит количество оставшихся цифр равно $K-1$. Тогда мы можем смело ставить в позицию $i-1$ цифру $d+1$ (если только $d\neq 9$), а за ней – либо наименьшую из всех уже имеющихся цифр (если цифра $d+1$ новая), либо цифру 0 (если $d+1$ уже была во множестве использованных цифр, мы имеем запас для введения новой цифры). Если же после убирания цифры d из $(i-1)$-го разряда, все же оставались еще и другие цифры d, то у нас нет возможности вводить новые цифры и на позицию $i-1$ следует подбирать цифру, большую d, но из множества тех цифр, которые встречались уже на более ранних позициях, а последующие заполнить минимальной. То есть выполнить в точности то же, что мы пытались уже сделать при изменении i-ой позиции. Если по той или иной причине не удалось произвести изменение начиная с $(i-1)$-ой позиции, то нужно перейти в позицию $i-2$ и попробовать изменить уже ее. Если не удастся в позиции $i-2$ увеличить цифру, то возвращаемся еще дальше – в позицию $i-3$. В худшем случае мы сможем дойти до позиции 1 и уже там поставить большую цифру обязательно сможем. Следует отметить, что количество

разрядов числа r не изменится.

Поиск числа l выполняется аналогично. Единственное, что следует отметить, что в этом случае старший разряд может стать равным 0. То есть количество разрядов числа может уменьшиться на 1. И нужно аккуратно проследить за тем, чтобы не вывести лишний ведущий 0 в случае, когда число l отлично от 0.

Задача I. Гиперкуб High (только для высшей лиги)

Имя входного файла:	`i.in`
Имя выходного файла:	`i.out`
Ограничение по времени:	4 с
Ограничение по памяти:	8 Мб



N-мерным гиперкубом (или N-кубом) со стороной $a > 0$ называется фигура в N-мерном евклидовом пространстве, которая строится следующим образом. Выберем в N-мерном пространстве какую-нибудь $(N-1)$-мерную гиперплоскость. Построим в ней $(N-1)$-куб. От каждой вершины этого куба проведем отрезок длины a перпендикулярно выбранной плоскости в одном и том же направлении. Концы этих отрезков будут, очевидно, также лежать в $(N-1)$-мерной плоскости, параллельной исходной, и образовывать $(N-1)$-куб. Выпуклая оболочка всех получившихся вершин (и исходного, и нового $(N-1)$-куба) образует N-мерный гиперкуб. По определению 0-куб – это точка, которая является единственной вершиной этого куба. Нетрудно увидеть, что 1-куб – отрезок на прямой, 2-куб – квадрат на плоскости, 3-куб – обычный трехмерный куб в трехмерном пространстве. Гиперкубы большей размерности увидеть несколько сложнее, но тем не менее формально можно построить по определению.

k-мерной гранью (или k-гранью) N-мерного гиперкуба называется такое пересечение его с k-мерной гиперплоскостью, которое не содержит внутренних точек граней большей размерности. Каждый N-куб имеет одну N-грань, совпадающую с ним самим. Можно доказать, что k-грани представляют собой k-кубы. 0-грани – это вершины N-куба, 1-грани – его ребра и т.д.

Требуется вычислить количество k-мерных граней N-мерного гиперкуба.

Формат входного файла

В единственной строке задаются три целых числа N, K и p.

Ограничения

$0 \le K \le N \le 10^{18}$, $K \le 2000$, $1 \le p \le 10^{18}$.

Формат выходного файла

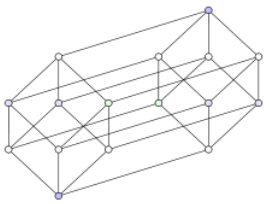
В единственную строку выведите $K+1$ число: остаток от деления на p количества 0-граней, 1-граней, ... K-граней N-куба.

Примеры

i.in	i.out
0 0 10	1
1 1 10	2 1
2 2 10	4 4 1
3 3 10	8 2 6 1

Задача J. Гиперкуб Junior (только для юниорской лиги)

Имя входного файла:	j.in
Имя выходного файла:	j.out
Ограничение по времени:	1 с
Ограничение по памяти:	8 Мб



N-мерным гиперкубом (или N-кубом) со стороной $a > 0$ называется фигура в N-мерном евклидовом пространстве, которая строится следующим образом. Выберем в N-мерном пространстве какую-нибудь $(N-1)$-мерную гиперплоскость. Построим в ней $(N-1)$-куб. От каждой вершины этого куба проведем отрезок длины a перпендикулярно выбранной плоскости в одном и том же направлении. Концы этих отрезков будут, очевидно, также лежать в $(N-1)$-мерной плоскости, параллельной исходной, и образовывать $(N-1)$-куб. Выпуклая оболочка всех получившихся вершин (и исходного, и нового $(N-1)$-куба) образует N-мерный гиперкуб. По определению 0-куб – это точка, которая является единственной вершиной этого куба. Нетрудно увидеть, что 1-куб – отрезок на прямой, 2-куб – квадрат на плоскости, 3-куб – обычный трехмерный куб в трехмерном пространстве. Гиперкубы большей размерности увидеть несколько сложнее, но тем не менее формально можно построить по определению.

k-мерной гранью (или k-гранью) N-мерного гиперкуба называется такое пересечение его с k-мерной гиперплоскостью, которое не содержит внутренних точек граней большей размерности. Каждый N-куб имеет одну

N-грань, совпадающую с ним самим. Можно доказать, что k-грани представляют собой k-кубы. 0-грани – это вершины N-куба, 1-грани – его ребра и т.д.

Требуется вычислить количество k-мерных граней N-мерного гиперкуба.

Формат входного файла

В единственной строке задаются три целых числа N, K и p.

Ограничения

$0 \le K \le N \le 10000$, $K \le 2000$, $1 \le p \le 10^{18}$.

Формат выходного файла

В единственную строку выведите $K+1$ число: остаток от деления на p количества 0-граней, 1-граней, ... K-граней N-куба.

Примеры

j.in	j.out
0 0 10	1
1 1 10	2 1
2 2 10	4 4 1
3 3 10	8 2 6 1

Разбор задачи Гиперкуб

Пусть $G(N,k)$ – количество k-граней N-куба. Выведем для этой величины рекуррентную формулу, исходя из рекуррентного определения N-куба. N-куб состоит из двух $(N-1)$-кубов, каждая из k-граней этих кубов, является также и k-гранью N-куба. Но кроме того, поскольку $(N-1)$-куб по построению приобретает еще одно измерение, то каждая его $(k-1)$-грань определяет k-грань N-куба. Отсюда получаем формулу:

$$G(n,k) = 2G(n-1,k) + G(n-1,k-1).$$

Начальные условия легко определятся непосредственным рассмотрением 0-куба: $G(0,0) = 1$, $G(0,k) = 0$ для $k \neq 0$.

Вычисление значений $G(N,k)$, для $k = \overline{0,K}$ по этим формулам потребует порядка $O(NK)$ операций, что вполне приемлемо для варианта Junior.

Рассмотрим теперь другой (комбинаторный) подход к определению $G(N,k)$. Введем систему координат с началом в одной из вершин N-куба,

а ребра исходящие из нее выберем в качестве базиса. Тогда любая вершина N-куба будет иметь координаты $(x_1, x_2, \ldots, x_N)$, где каждое x_i равно 0 или 1. Отсюда сразу следует, что количество вершин (то есть 0-граней) N-куба равно 2^N. Если позволить одной из координат меняться от 0 до 1, то получим некоторое одномерное множество, соединяющее две смежные вершины. Очевидно, это будет не что иное как ребро. А общее количество ребер будет определяться количеством вариантов выбора меняющейся координаты (N), и количеством вариантов выбора значений неменяющихся координат (2^{N-1}). То есть количество ребер (1-граней) N куба будет равно $N2^{N-1}$. Общий случай k-граней будет характеризоваться k изменяющимися координатами, и $N-k$ фиксированными. Количество таких граней составит $C_N^k 2^{N-k}$. Вычислить значения 2^{N-k} можно за $O(\log N + K)$, используя широко известный бинарный алгоритм возведения в степень первый раз, а дальше домножая на 2. Остается научиться быстро находить C_N^k.

Первый подход. Треугольник Паскаля. Основан на формуле

$$C_N^k = C_{N-1}^k + C_{N-1}^{k-1}.$$

Вычисление по этой формуле даст асимптотическую оценку времени работы порядка $O(NK)$, что ничуть не лучше, чем было при использовании рекуррентного соотношения для $G(N,k)$.

Второй подход. Бинарный алгоритм. Основан на следующей идее. При нечетном N мы используем формулу треугольника Паскаля. При четном N используется формула

$$C_N^k = \sum_{0 \le i \le k} C_{N/2}^i C_{N/2}^{k-i}.$$

Эта формула может быть получена разбиением всех вариантов выбора k элементов из N-элементного множества на классы, в каждом из которых мы выбираем некоторое количество i из одной фиксированной половины множества (допустим, первые $N/2$ элементов), а $k-i$ – из другой. Алгоритм, работающий на этой идее, будет иметь асимптотику $O(K^2 \log N)$.

Третий подход. Вычисление по формуле $C_N^K = N^{\underline{K}}/1^{\overline{K}}$. Поскольку нам нужно вычислять по модулю p (необязательно простому), то с операцией деления следует быть осторожным. Чтобы избежать деления по модулю, выполним деление до вычислений по модулю. А именно, выпишем в некотором массиве Q все числа от N до $N-K+1$. Все числа от 1 до K разложим на простые множители (очевидно, общее их количество можно оценить величиной порядка $K \log K$). Теперь для каждого простого множителя найдем такое значение в массиве Q, которое делится на этот простой

множитель, и выполним требуемое деление. Тогда останется перемножить все значения Q по модулю p и получить требуемый ответ. Сложность алгоритма – $O(K^2 \log K)$. При правильном выборе порядка деления на множители и многократной сборке результата умножением чисел из Q, этот алгоритм может быть приспособлен и к вычислению всех значений C_N^k, не ухудшая асимптотическую оценку сложности.

Задача K. Кривая дракона High (только для высшей лиги)

Имя входного файла:	`k.in`
Имя выходного файла:	`k.out`
Ограничение по времени:	1 с
Ограничение по памяти:	64 Мб

Кривая дракона – это бесконечная ломаная из звеньев единичной длины, которая строится следующим образом. Выбирается некоторая точка на плоскости (например, точка $(0, 0)$) и одно из четырех направлений, параллельных координатным осям. Левый(правый) дракон порядка n строится так:

- если n равно 0, то отложить отрезок длины 1 от текущей точки в текущем направлении, переместившись в конечную точку;
- в противном случае построить левого дракона порядка $n - 1$ от текущей точки в текущем направлении, повернуться на 90 градусов налево(направо) в его конечной точке и построить правого дракона порядка $n - 1$.

Поскольку левый дракон порядка n содержит в качестве своего начала левого дракона порядка $n - 1$, то вполне корректно определяется левый дракон бесконечного порядка. Именно эта ломаная и называется кривой дракона.

Построим из точки $(0, 0)$ кривые дракона во всех четырех направлениях. Есть теорема, доказанная Д.Кнутом, о том, что эти кривые не пересекаются (за исключением касания в вершинах) и полностью покрывают целочисленную сетку на плоскости. В данной задаче от вас потребуется проверить эту теорему, а именно по заданному единичному отрезку сетки определить, какому из четырех драконов он принадлежит.

Формат входного файла

В единственной строке входного файла задаются 4 целых числа: две координаты одного конца некоторого отрезка, параллельного одной из осей координат, и две координаты другого конца.

Ограничения

Все числа не превышают по модулю 10^9. Расстояние между точкам равно 1.

Формат выходного файла

В первой строке выходного файла выведите номер драконовой кривой, которой принадлежит соответствующее звено (1 – дракон, отложенный в положительном направлении оси Ox, 2 – в положительном направлении оси Oy, 3 – в отрицательном направлении оси Ox, 4 – в отрицательном направлении оси Oy). Во второй строке выведите номер этого звена в соответствующей ломаной. Гарантируется, что это число не превосходит 10^{12}.

Примеры

k.in	k.out
1 0 1 1	1 2
2 -2 2 -1	3 8

Задача L. Кривая дракона Junior (только для юниорской лиги)

Имя входного файла:	`l.in`
Имя выходного файла:	`l.out`
Ограничение по времени:	1 с
Ограничение по памяти:	64 Мб

Кривая дракона – это бесконечная ломаная из звеньев единичной длины, которая строится следующим образом. Выбирается некоторая точка на плоскости (например, точка $(0, 0)$) и одно из четырех направлений, параллельных координатным осям. Левый(правый) дракон порядка n строится так:

- если n равно 0, то отложить отрезок длины 1 от текущей точки в текущем направлении, переместившись в конечную точку;
- в противном случае построить левого дракона порядка $n - 1$ от текущей точки в текущем направлении, повернуться на 90 градусов налево(направо) в его конечной точке и построить правого дракона порядка $n - 1$.

Поскольку левый дракон порядка n содержит в качестве своего начала левого дракона порядка $n-1$, то вполне корректно определяется левый дракон бесконечного порядка. Именно эта ломаная и называется кривой дракона.

Построим из точки $(0,0)$ кривые дракона во всех четырех направлениях. Есть теорема, доказанная Д.Кнутом, о том, что эти кривые не пересекаются (за исключением касания в вершинах) и полностью покрывают целочисленную сетку на плоскости. В данной задаче от вас потребуется проверить эту теорему, а именно по заданному единичному отрезку сетки определить, какому из четырех драконов он принадлежит.

Формат входного файла

В единственной строке входного файла задаются 4 целых числа: две координаты одного конца некоторого отрезка, параллельного одной из осей координат, и две координаты другого конца.

Ограничения

Все числа не превышают по модулю 10^9. Расстояние между точкам равно 1.

Формат выходного файла

В первой строке выходного файла выведите номер драконовой кривой, которой принадлежит соответствующее звено (1 – дракон, отложенный в положительном направлении оси Ox, 2 – в положительном направлении оси Oy, 3 – в отрицательном направлении оси Ox, 4 – в отрицательном направлении оси Oy). Во второй строке выведите номер этого звена в соответствующей ломаной. Гарантируется, что это число не превосходит 10^7.

Примеры

1.in	1.out
1 0 1 1	1 2
2 -2 2 -1	3 8

Разбор задачи Кривая дракона

Следует отметить, что заданный отрезок может проходиться как в одном, так и в другом направлении. Таким образом, будем искать, на каком

шаге построения драконовой ломаной мы проходим от первой из заданных точек в направлении второй, либо от второй точки в направлении первой. Для того, чтобы не строить все 4 драконовы ломанные, совместим все ломаные с первой поворотом. Это будет означать, что мы будем искать не 2 набора точка-направление, а 8 (эти наборы получаются из уже определенных ранее двух поворотами на 90, 180 и 270 градусов по часовой стрелке соответственно).

В варианте Junior задачи достаточно будет просто начать строить кривую дракона порядка хотя бы 24. И если на каком-то шаге ее построения встретится по крайней мере один из 8 найденных наборов точка-направление, то выдать номер звена и номер кривой (который определяется просто углом поворота, который был выполнен для получения указанного набора.

Для решения варианта High воспользуемся техникой "Meet in the middle". Поскольку существует ограничение на количество ходов – 10^{11}, нам достаточно было бы рассмотреть драконову кривую порядка 37. Однако всю целиком ее мы построить не успеем.

Для решения нам были бы полезны такие операции: получение для двух наборов точка-направление набора, который получится если после достижения первой точки-направления из точки $(0, 0)$ с направлением, совпадающим с положительным направлением оси Ox, построить участок ломаной, который при старте из $(0, 0, Ox^+)$ привел бы во вторую точку-направление. Назовем эту операцию сложением. Пусть первый набор будет (x_1, y_1, d_1), а второй – (x_2, y_2, d_2). Тогда нужно будет выполнить x_2 шагов в направлении d_1, а затем y_2 шагов в направлении, повернутом на 90 градусов против часовой стрелки относительно d_1. Направление же изменится относительно d_1 на угол, определяемый углом между d_2 и осью Ox^+:

$$\begin{aligned} x' &= x_1 + x_2 \cdot dx[d_1] + y_2 \cdot dx[d_1 + 90^\circ], \\ y' &= y_1 + x_2 \cdot dy[d_1] + y_2 \cdot dy[d_1 + 90^\circ], \\ d' &= d_1 + d_2. \end{aligned}$$

Кроме того потребуется и обратная операция – по двум точкам-направлениям (x_1, y_1, d_1) и (x_2, y_2, d_2) найти смещение-направление (x', y', d'), которое при откладывании его от (x_1, y_1, d_1) приведет в (x_2, y_2, d_2). Назовем эту операцию вычитанием. Получить выражения для величин (x', y', d') будет несложно, если разрешить уравнения, которые выписаны для операции сложения,

Построим кривую дракона порядка 18 и запомним в ассоциативном массиве для каждого набора точка-направление номер соответствующего звена в левом драконе порядка 18. Теперь мы знаем, в какой точке-направлении

закончился этот левый дракон(18). Кроме того, нам понадобится знать также, в какой точке-направлении заканчивается правый дракон того же порядка. Это может быть сделано, заметив, что если построен левый дракон, то после разворота на 180 градусов и построении от этой точки правого дракона, мы проделаем тот же путь, только в обратном направлении. Другой же способ основан на том, что точка-направление D_i^L для левого дракона i должна определяться по формуле $D_{i-1}^L + (0, 0, +90^\circ) + D_{i-1}^R$, а для правого соответственно $D_{i-1}^L + (0, 0, -90^\circ) + D_{i-1}^R$.

Теперь начнем строить левого дракона порядка 37, но как только в рекурсивном спуске мы достигаем точки, где должна была бы вызываться функция построения левого или правого дракона порядка K, мы будем вызывать операцию сложения текущей точки-направления с D_{18}^L или D_{18}^L. Однако перед этим нужно будет вычесть из каждой из 8 искомых точек-направлений текущей точки-направления, после чего проверить (за счет ассоциативного массива), нет ли в соответствующем драконе 18-го порядка требуемой точки-направления. Для правого дракона можно сначала выполнить перемещение, а потом там развернуться на 180° и только тогда уже пользоваться ассоциативным массивом для нахождения номера звена. С учетом добавления k раз по 2^{18}, где k – количество пройденных уже драконов 18-го порядка.

Сложность алгоритма можно будет тогда оценить величиной порядка $\sqrt{L} \log L$, где L – максимальный номер звена, который может быть ответом (в нашем случае $L = 10^{11}$).

Задача M. Ферзи High (только для высшей лиги)

Имя входного файла:	`m.in`
Имя выходного файла:	`m.out`
Ограничение по времени:	1 с
Ограничение по памяти:	128 Мб

На d-мерной шахматной доске, каждое измерение которой составляет N, в различных ячейках стоят K ферзей. Ферзь может перемещаться на любое натуральное количество клеток в любом прямом или диагональном направлении. Более точно, ферзь может попасть из ячейки с координатами $(x_1, x_2, \ldots, x_d)$ в ячейку с координатами $(x'_1, x'_2, \ldots, x'_d)$ тогда и только тогда, когда среди чисел $|x_i - x'_i|$ есть по крайней мере одно ненулевое, и все ненулевые значения равны между собой. Будем говорить, что один ферзь угрожает другому, если существует ход из ячейки первого ферзя в ячейку второго. Очевидно, что это отношение симметрично, то есть если один ферзь угрожает другому, то и другой первому. Будем говорить, что

один ферзь находится под боем у другого, если второй угрожает первому и все ячейки между ними свободны. Это отношение также симметрично.

Требуется определить количество пар ферзей, угрожающих друг другу и находящихся под боем друг у друга.

Формат входного файла

В первой строке задаются три целых числа d, N, K. В каждой из последующих K строк записано по d целых чисел $x_1, \ldots, x_d$, определяющих координаты ячейки соответствующего ферзя.

Ограничения

$1 \le d \le 5$, $1 \le N \le 1000$, $0 \le K \le 40000$, $1 \le x_i \le N$.

Формат выходного файла

В первой строке выведите количество пар ферзей, угрожающих друг другу, во второй – количество пар ферзей, находящихся под боем друг у друга.

Примеры

m.in	m.out
2 8 5 1 1 3 3 3 1 1 3 5 5	8 7
4 5 4 1 1 1 1 3 1 3 1 5 3 5 3 4 4 4 4	4 4

Задача N. Ферзи Junior (только для юниорской лиги)

Имя входного файла:	n.in
Имя выходного файла:	n.out
Ограничение по времени:	1 с
Ограничение по памяти:	128 Мб

На шахматной доске размера $N \times N$ в различных клетках стоят K ферзей. Ферзь может перемещаться на любое натуральное количество клеток

в горизонтальном, вертикальном или диагональном направлении. Более точно, ферзь может попасть из клетки с координатами (x, y) в клетку с координатами (x', y') тогда и только тогда, когда числа $|x - x'|$ и $|y - y'|$ равны между собой и отличны от нуля, либо одно из них равно нулю, а другое отлично от нуля. Будем говорить, что один ферзь угрожает другому, если существует ход из ячейки первого ферзя в клетку второго. Очевидно, что это отношение симметрично, то есть если один ферзь угрожает другому, то и другой первому. Будем говорить, что один ферзь находится под боем у другого, если второй угрожает первому и все клетки между ними свободны. Это отношение также симметрично.

Требуется определить количество пар ферзей, угрожающих друг другу и находящихся под боем друг у друга.

Формат входного файла

В первой строке задаются два целых числа N, K. В каждой из последующих K строк записано по два целых числа x, y, определяющих координаты соответствующего ферзя.

Ограничения

$1 \leq N \leq 10^6$, $0 \leq K \leq 10^5$, $1 \leq x, y \leq N$.

Формат выходного файла

В первой строке выведите количество пар ферзей, угрожающих друг другу, во второй – количество пар ферзей, находящихся под боем друг у друга.

Примеры

n.in	n.out
8 5 1 1 3 3 3 1 1 3 5 5	8 7
5 4 1 1 3 1 5 3 4 4	4 4

Разбор задачи Ферзи

Нетрудно было бы написать проверку пар ферзей на угрозу за $O(K^2d)$, выполняя проверку требования, указанного условия. Однако, если не использовать дополнительной памяти, очевидная проверка на "под боем- потребовала бы порядка $O(K^3d)$, что уже почти наверняка не успеет вложиться в определенное время.

К счастью, ограничения на размеры доски таковы, что можно заводить дополнительные массивы. Воспользуемся ими для того, чтобы для каждой линии (вертикали, горизонтали или диагонали) сосчитать количество ферзей, стоящих на этой линии. Тогда ответ на количество пар, бьющих друг друга ферзей будет получен, если просуммировать для всех непустых линий количество ферзей на них без одного.

Количество же угрожающих друг другу ферзей можно тогда явно вычислить как сумму выражений $K_i(K_i-1)/2$ по всем непустым линиям (все ферзи, стоящие на одной линии, очевидно, угрожают друг другу).

Таким способом решается задача как в варианте Junior, так и в High. Однако в варианте High потребуются более сложные структуры данных, поскольку линий может быть настолько много, что завести на них всех какой-то массив было бы неудачной идеей. Поэтому достаточно сохранить значения лишь для тех линий, которые содержат хотя бы одного ферзя. Для этого подойдут структуры map или hash_map. Кроме того, следует отметить, что каждая ячейка принадлежит $p = (3^d - 1)/2$ линиям в различных направлениях. Для того чтобы не запутаться в этих линиях, важно определить в каком-то смысле наименьшую ячейку (например, лексикографически минимальную) и направление, в котором от этой ячейки идет линия. Этих данных достаточно для того, чтобы однозначно определять линию. Направление легко задать вектором вида $dx = (dx_1, dx_2, \ldots, dx_N)$, где $dx_i \in [-1, 1]$, при этом есть хотя бы одно ненулевое значение dx_i. Более того, поскольку направления dx и $-dx$ задают одну и ту же линию (если брать их от одной ячейки), то можно выбирать направления так, чтобы первое ненулевое смещение dx_i было равно $+1$.

Сложность алгоритма может быть выражена как $O(pKd \log pK)$.

Задача O. K-стороннее домино

Имя входного файла:	`o.in`
Имя выходного файла:	`o.out`
Ограничение по времени:	1.5 с
Ограничение по памяти:	8 Мб

Стандартный набор домино содержит 28 костяшек. Костяшка представляет собой прямоугольник, разделенный на 2 части. Каждая часть может содержать одно число из множества $\{0, 1, 2, 3, 4, 5, 6\}$. При этом числа на обоих частях могут совпадать. В наборе есть все возможные костяшки и никакие две костяшки не содержат одну и ту же пару чисел.

Рассмотрим набор домино, в котором костяшки разделяются на K частей. Числа, которые содержатся на частях костяшек, будут выбираться из некоторого множества A, состоящего из N элементов. Две костяшки считаются одинаковыми, если множества чисел, записанных на них, совпадают с учетом кратности. Набор содержит все возможные костяшки без повторений.

Определите количество костяшек в наборе и общую сумму всех чисел на них.

Формат входного файла

В первой строке задаются два целых числа N, K. Во второй строке задаются числа a_i множества A.

Ограничения

$1 \le K \le 10^4$, $1 \le N \le 10^6$, $0 \le a_i \le 10^9$.
Все a_i различны.

Формат выходного файла

В единственную строку выведите два числа – количество костяшек в наборе и сумму всех чисел на них.

Пример

o.in	o.out
7 2 0 1 2 3 4 5 6	28 168

Разбор задачи K-стороннее домино

Фактически задача математическая. Нужно выразить в замкнутой форме два значения: количество различных неупорядоченных наборов K чисел с повторения из множества A, а также общую сумму чисел во всех этих наборах. Как только это будет сделано, в дальнейшем нужно будет реализовать эти формулы в программе, учитывая, что ответы могут иметь довольно большую длину.

К решению можно применить несколько различных подходов. Применим подход, связанный с вычислением сумм. Каждому неупорядоченному набору мы можем поставить в соответствии набор чисел $(i_1, i_2, \ldots, i_K)$ такой, что $1 \le i_K \le i_{K-1} \le \ldots \le i_2 \le i_1 \le N$. Тогда количество наборов будет определяться суммой:

$$S_1 = \sum_{0<i_1\le N} \sum_{0<i_2\le i_1} \ldots \sum_{0<i_K\le i_{K-1}} 1.$$

Раскроем в явном виде самую внутреннюю сумму

$$\sum_{0<i_K\le i_{K-1}} 1 = i_{K-1} - 0 = i_{K-1}$$

Перейдем к следующей сумме:

$$\sum_{0<i_{K-1}\le i_{K-2}} \sum_{0<i_K\le i_{K-1}} 1 = \sum_{0<i_{K-1}\le i_{K-2}} i_{K-1} = \frac{i_{K-1}^{\overline{2}}}{2}\Bigg|_{i_{K-1}=0}^{i_{K-1}=i_{K-2}} = \frac{i_{K-2}^{\overline{2}}}{2}.$$

Нетрудно сделать предположение о том, что вся K-кратная сумма будет равна

$$S_1 = \frac{N^{\overline{K}}}{1^{\overline{K}}}$$

и доказать это индукцией по количеству сумм K.

Перейдем теперь к вычислению суммы чисел на костяшках:

$$S_2 = \sum_{0<i_1\le N} \sum_{0<i_2\le i_1} \ldots \sum_{0<i_K\le i_{K-1}} (a_{i_1} + a_{i_2} + \ldots + a_{i_K}).$$

Введем следующие обозначения:

$$(Sa)_i = \sum_{0<j\le i} a_i,$$

$$(SSa)_i = \sum_{0<j\le i} (Sa)_i.$$

Нетрудно видеть, что в таком случае:

$$\Delta(Sa)_i = (Sa)_i - (Sa)_{i-1} = a_i,$$
$$\Delta(SSa)_i = (SSa)_i - (SSa)_{i-1} = (Sa)_i,$$

где под Δ понимается операция взятия обратной разности.

Раскрываем сумму по i_K:

$$\sum_{0<i_K\le i_{K-1}} (a_{i_1} + a_{i_2} + \ldots + a_{i_K}) = (a_{i_1} + a_{i_2} + \ldots + a_{i_{K-1}})i_{K-1} + (Sa)_{i_{K-1}}.$$

Переходим к сумме по i_{K-1}:

$$\sum_{0<i_{K-1}\le i_{K-2}} (a_{i_1}+a_{i_2}+\ldots+a_{i_{K-1}})i_{K-1}+(Sa)_{i_{K-1}} = (a_{i_1}+a_{i_2}+\ldots+a_{i_{K-2}})\frac{\overline{i_{K-2}^2}}{2}+$$

$$+ \sum_{0<i_{K-1}\le i_{K-2}} a_{i_{K-1}}i_{K-1} + SSa_{i_{K-2}}.$$

Заменим в оставшейся сумме $a_{i_{K-1}}$ на $\Delta(Sa)_{i_{K-1}}$ и применим формулу суммирования по частям:

$$\sum_{0<i_{K-1}\le i_{K-2}} i_{K-1}\Delta(Sa)_{i_{K-1}} = (Sa)_{i_{K-1}}(i_{K-1}+1)\Big|_{i_{K-1}=0}^{i_{K-1}=i_{K-2}} -$$

$$- \sum_{0<i_{K-1}\le i_{K-2}} (Sa)_{i_{K-1}}\Delta(i_{K-1}+1) = (Sa)_{i_{K-2}}(i_{K-2}+1) - (SSa)_{i_{K-2}}.$$

Подставляя полученное выражение, получим замкнутое выражение для второй внутренней суммы:

$$\sum_{0<i_{K-1}\le i_{K-2}} (\ldots) = (a_{i_1} + a_{i_2} + \ldots + a_{i_{K-2}})\frac{\overline{i_{K-2}^2}}{2} + (Sa)_{i_{K-2}}(i_{K-2}+1).$$

Как видим величины $(SSa)_{i_{K-2}}$ сократились. Это же будет справедливо и при индуктивном переходе, который позволит нам доказать формулу

$$S_2 = (Sa)_N \frac{(N+1)^{\overline{K}}}{1^{\overline{K}}}.$$

Остается лишь записать программу, которая выполнит соответствующие вычисления по приведенным формулам в числах, имеющих большую длину.

Задача P. Различные попарные суммы

Имя входного файла:	`p.in`
Имя выходного файла:	`p.out`
Ограничение по времени:	1 с
Ограничение по памяти:	8 Мб

Дано натуральное число n. Требуется построить последовательность различных натуральных чисел $a_1, a_2, \ldots, a_n$, не больших $2n^2 + 4n\sqrt{n}$, такую, что все их попарные суммы различны. Другими словами, все $n(n-1)/2$ чисел вида $a_i + a_j$, $1 \le i < j \le n$ должны быть различны. Гарантируется, что такая последовательность существует.

Формат входного файла

В единственной строке входного файла задано натуральное число $n \le 5000$.

Формат выходного файла

В единственную строку выходного файла выведите через пробел числа $a_1, a_2, \ldots, a_n$. Если решений несколько выведите любое.

Примеры

p.in	p.out
2	1 2
3	3 1 2
8	1 2 3 5 8 13 21 34

Разбор задачи Попарно различные суммы

Легко проверить, что если строить такую последовательность жадно, то есть на очередном шаге выбирать наименьшее натуральное число, большее всех предыдущих, добавление которого не приводит к появлению совпадающих попарных сумм, мы получим последовательность Фибоначчи $1, 2, 3, 5, 8, 13, 21, 34, \ldots$. Но она растет очень быстро и уже при $n = 14$ не удовлетворяют условию задачи. По-видимому, не существует универсальной бесконечной возрастающей последовательности натуральных чисел $a_1, a_2, \ldots, a_n, \ldots$ такой, что для любого n первые ее n членов удовлетворяют условию задачи. Пример здесь строится для каждого n по-своему. Опишем это построение. Итак, пусть $n \in \mathbb{N}$. Пусть p – минимальное простое число, которое не меньше n. Тогда элементы искомой последовательности $a_0, a_1, \ldots, a_{n-1}$ можно определить по формуле

$$a_k = 2pk + (k^2 \bmod p) + 1, \quad k = \overline{0, n-1}. \tag{1}$$

(Нам здесь удобней нумеровать члены последовательности, начиная с нуля.)

Докажем, что она действительно подходит. Ясно, что

$$2pk < a_k \le 2pk + p, \quad k = \overline{0, n-1}.$$

Поэтому все элементы этой последовательности различные натуральные числа. Экспериментально можно установить, что $p \le n + 2\sqrt{n} - 1$ при $n \le 5000$ (скорее всего это верно для всех $n \in \mathbb{N}$). Поэтому

$$a_k \le a_{n-1} \le p(2n-1) \le (n+2\sqrt{n}-1)(2n-1) < 2n^2+4n\sqrt{n}, \quad k = \overline{0, n-1}.$$

Осталось проверить, что все попарные суммы различны. При $n \le 2$ это очевидно. Пусть $n \ge 3$. Тогда $p \ge 3$ и значит является нечетным числом. Уменьшим все a_i на 1 – ясно, что все попарные суммы тогда уменьшатся на 2 и значит, если бы были повторения для исходных сумм, то эти повторения были бы и в новых суммах.

Так как $0 \le (x^2 \bmod p) < p$ при $x \in \mathbb{Z}$, то

$$\left\lceil \frac{a_j + a_k}{2p} \right\rceil = j + k + \left\lfloor \frac{(j^2 \bmod p) + (k^2 \bmod p)}{2p} \right\rfloor = j + k. \tag{2}$$

Поэтому, если $a_j + a_k = a_u + a_v$ для некоторых $j, k, u, v \in \{0, 1, \ldots, n-1\}$, то из (2) следует, что $j + k = u + v$. Из (1) теперь следует, что $j^2 + k^2 \equiv u^2 + v^2 \pmod p$. Откуда в силу нечетности p имеем $jk \equiv uv \pmod p$. Пусть $j + k = u + v = s$. Тогда $k = s - j$, $v = s - u$ и $j(s-j) \equiv u(s-u) \pmod p$. Откуда $(u-j)(s-u-j) \equiv 0 \pmod p$. Поэтому либо $j \equiv u \pmod p$ откуда $j = u$ и $k = v$, либо $j \equiv s - u \equiv v \pmod p$ откуда $j = v$ и $k = u$. В любом случае получаем, что неупорядоченные пары $\{j, k\}$ и $\{u, v\}$ совпадают. Поэтому все попарные суммы различны.

День девятый (21.02.2012 г.)
Контест Копелиовича Сергея Владимировича

Об авторе...

Копелиович Сергей Владимирович родился в Санкт-Петербурге в 1989 году. Закончил физ-мат лицей №30. В 4-6 классах занимался в кружке по математике. Олимпиадами по программированию начал активно заниматься в 9-м классе. Золотые медали на IOI в 2005 (Польша) и 2006 (Мексика) годах. В ACM соревнованиях выступал в команде Burunduchki. Бронзовая и золотая медали в 2008 (Канада) и 2009 (Швеция) годах.

Сейчас второй год подряд студент 5-го курса Санкт-Петербургского государственного университета. Член научного комитета Всероссийской школьной олимпиады по информатике и сборов по подготовке к международной олимпиаде школьников.

С 2008-2009 учебного года регулярно читает лекции в СПбГУ. Ссылки на планы лекций можно посмотреть здесь http://kruzhok.spbgu.ru/09e и здесь http://195.19.228.2/~sk1/mm/345/

В 2010-2011 году год работал стажером в санкт-петербургском офисе Яндекса.

Теоретический материал. Про алгоритм Йена и динамику

Алгоритм Йена

1. Формулировка: пусть все простые пути из s в t отсортированы по весу, нужно найти первые (минимальные) K.
2. Для начала научимся искать второй минимальный путь. Найдем первый алгоритмом Дейкстры. Второй где-то отличается от первого. Т.е. он сперва повторяет его префикс, а потом сворачивает, и дальше как-то идет к t. Давайте переберем место, в котором второй путь будет

отличаться от первого. Пусть первый путь = $v_1, v_2, \cdots, v_k$. Скажем, что вершины $v_1, v_2, \cdots, v_i$ мы прошли, а затем пошли в любую другую вершину, но не v_{i+1}. Для этого запустим Дейкстру из вершины v_i, запретив ей проходить по вершинам $v_1, v_2, \cdots, v_i$ (там мы уже были) и по ребру (v_i, v_{i+1}). Из полученных путей выберем минимум.

3. На самом деле в предыдущем пункте мы разбили все простые пути из s в t на классы и в каждом классе выбрали минимальный элемент. Класс путей — все пути, начинающиеся на $v_1, v_2, \cdots, v_i$ для фиксированного i и не проходящие по ребру (v_i, v_{i+1}). Классы не пересекаются и в объединении дают все пути, кроме минимального (его мы уже выбрали). Чтобы находить K-й путь нужно в каждый момент времени хранить все классы, на очередном шаге выбирать минимальный путь, и класс, к которому относится выбранный путь, разбивать на более мелкие классы (это делается так же, как когда мы искали второй путь). При этом запрещенными могут оказаться уже несколько ребер.

4. Всего за K итераций у нас появится $O(KV)$ классов. Оценка времени работы алгоритма: $O(KV \cdot Dijkstra)$ Оценка памяти: $O(KV \cdot V)$. Т.к. для хранения одного класса нужно исользовать $O(V)$ памяти. Память можно сократить до $O(K \cdot V)$, т.к. на самом деле не нужно хранить все классы, достаточно хранить только K минимальных.

5. Применение Йена для алгоритма Краскала (MST). Отсортируем один раз в самом начале ребра (сортируем по возрастанию веса). Теперь остовное дерево — последовательность ребер. Какие-то i первых мы можем точно взять, $i+1$-е запретить брать, а теперь сперва насильно добавить ребра, которые мы точно берем, а для оставшихся запустить Краскала (ребро берем, если оно не запрещено и не образует цикл).

6. Применение Йена для алгоритма Эдмондса (matching). Занумеруем все ребра от 1 до E. Теперь в любом паросочетании ребра можно упорядочить по номерам ребер. Применяем ту же идею: первые i берем, $i+1$-е ребро запрещаем, для оставшегося графа запускаем Эдмондса.

Динамика на ацикличном графе

1. Динамическое программирование (*Динамика*) — это всегда задача на ацикличном графе. Если граф содержит циклы, то применяют или Дейкстру, или Форд-Беллмана, или поиск в ширину. Исключением являются графы с простой структурой, где наличие циклов можно обойти простыми хаками.

2. Стандартные задачи, которые решает динамическое программирование, с точки зрения графов таковы: суммарное число путей, минимальный путь, максимальный путь, все из s в t, все за время $O(E)$.

3. В прошлом пункте мы предположили, что s и t — вершины. Бывает много конечных и начальных состояний — множества вершин S и T. Алгоритм от этого сложнее не становится, меняется база динамики.

4. Пусть на каждом ребре написана буква, тогда каждому пути соответствует строка. Лексикографически минимальный (*LexMin*) путь из s в t можно легко найти за $O(VE)$ (функция динамики = строка).

5. Если для каждой вершины символы, написанные на ее исходящих ребрах, различны, то можно жадно перебирать символы в порядке возрастания, а строку целиком не хранить, только ссылку на первое ребро. Получится решение за $O(E)$.

6. Пусть мы хотим найти путь минимальной длины (веса = 1, `bfs`), а из таких путей уже LexMin. В этом случае также есть решение за $O(E)$, а именно такое: `bfs` разбил граф на слои (слой — это вершины с одинаковым расстоянием до конца). Кратчайший путь — любой путь по слоям вперед. Будем вести динамику по слоям. Наша цель — занумеровать вершины целыми числами от 1 до N так, чтобы сравнение на больше, меньше, равенство строк, определяемых путем к вершине было равносильно сравнению номеров вершин. Пусть для слоя $i+1$ мы уже знаем номера вершин. Чтобы посчитать номера для i-го слоя, нужно увидеть, что строка из вершины в t — это пара [`первый символ`, `номер вершины из слоя i+1`]. Отсортируем и занумеруем эти пары. Конец.

7. В произвольном случае задача LexMin имеет решение за время $O(E \log V)$. Первая часть решения: не хранить для вершины всю строку, а только ссылку на первый символ. Сравнивать строки мы умеем все еще только за $O(V)$, а вот лишней памяти мы уже не используем. К тому же *ссылки на первый символ* образует дерево, сходящееся в вершину t. Чтобы быстро сравнивать строки, будем хранить для каждой вершины v хэш строки от v до t, а также таблицу двоичных подъемов ($p[v,k]$ = вершина, в которой мы окажемся, если пройдем от v по ссылкам вперед 2^k шагов). Если предположить $p[t,0] = t$ (корень ссылается на себя самого), то крайних случаев не будет, и для пересчета двоичных подъемов достаточно одной формулы: $p[v,k] = p[p[v,k-1],k-1]$. Теперь мы умеем сравнивать строки на

больше, меньше, равно за $O(\log N)$ и решение работает за обещанное время $O(E \log V)$.

Комментарии к лекции Саши Миланина

1. Простой рандомизированный алгоритм дает уже почти максимальное паросочетание. Его проблема в том, что он может ошибиться на $O(1)$ и последние, самые сложные, дополняющие пути просто не найти.

2. Чтобы восполнить эту проблему, запустим в конце любую, самую простую, реализацию алгоритма Эдмондса (сжатие соцветий). Даже если один дополняющий путь мы будем искать за время $O(V^3)$, поскольку дополняющих осталось мало, общее время работы будет также $O(V^3)$.

Комментарии к лекции Миши Дворкина: Cover

1. Алгоритм с оценкой два: построим насыщенное паросочетание M (такое, что никакое ребро нельзя добавить в него просто так, ничего не перестраивая). Возьмем все $2|M|$ концов. Это покрытие. Также понятно, что размер минимального покрытия $\geq |M|$.

2. `min Cover` = `max Independent Set` (т.к. дополнением к любому покрытию является независимое множество вершин и наоборот). Мы получили возможность, получив хорошее решение одной из двух задач, сразу применять его ко второй.

3. Очень хорошо работает следующая простая жадность: брать каждый раз вершину, покрывающую максимальное число еще не покрытых ребер (т.е. имеющую максимальную степень в остаточном графе).

4. Улучшим нашу жадность. Если есть вершина степени один, то нужно обязательно взять или ее, или ее единственного соседа. Очевидно, что выгодно брать соседа. Если мы в какой-то момент можем применить это отсечение, применяем.

5. Чтобы наша жадность работала совсем хорошо, нужно превратить ее в перебор. Т.е. брать не вершину max степени, а перебирать все вершины в порядке убывания степени. И так, пока время работы не превысит одну секунду. Сам по себе перебор уже лучше жадности, но есть последнее улучшение: отсечение по ответу. Пусть сейчас мы уже умеем строить покрытие размера $Best$. На текущем уровне рекурсии мы уже

взяли X вершин, но еще не покрыли все ребра. Очевидно, что если есть K независимых (не пересекающихся по концам) ребер, то нужно взять хотя бы еще K вершин. Отсечение:
`if` $Best \leq X + K$ `then return`.

Комментарии к лекции Миши Дворкина: Salesman Problem

1. Чтобы найти путь, нужно найти цикл и выкинуть одно самое дорогое ребро.

2. На практике решение этой задачи нужно для больших N. Т.е. если алгоритм имеет плохую асимптотику и не работает даже для $N = 2000$, он нам слабо интересен.

3. Алгоритм построения: будем из цикла длины k получать цикл длины $k + 1$. Изначально у нас есть цикл длины 1. Для того, чтобы увеличить цикл, выберем вершину и вставим ее между двумя соседними вершинами цикла. Выберем вершину и место для вставки так, чтобы длина цикла увеличилась как можно меньше.

4. Оценка времени. Если реализовывать описанную выше идею в лоб, то получится время $O(N^3)$. Наша задача быстро выбирать пары (вершина, место). Давайте также как в алгоритмах Дейкстры и Прима для каждой вершины помнить оптимальное место вставки и пересчитывать, если цикл меняется. Таким образом можно получить время $O(N^2)$ (если вы увидели только $O(N^2 \log N)$ — это нормально, но подумайте еще, $O(N^2)$ тоже существует).

5. Локальные оптимизации: на самом деле любой хороший алгоритм решения задачи коммивояжера состоит из двух фаз — поиск хорошего начального приближения (это мы уже умеем делать) и собственно улучшение текущего ответа. Т.е. локальные оптимизации. Существует два вида простых оптимизаций. Первый: для любой пятерки подряд идущих вершин мы можем перебрать все 120 вариантов и выбрать оптимальный, если что-то улучшилось — хорошо. Второй: пусть мы хотим решить задачу для точек на плоскости. Тогда ребра — это отрезки. Если какие-то два отрезка найденного цикла пересекаются, можно их развернуть так, что пересечение пропадет, а длина уменьшится.

Комментарии к лекции Миши Дворкина: Рюкзак

1. Будем решать чуть другую задачу: даны K рюкзаков, нам нужно уложить в них **все** вещи. Задача, обсуждавшаяся на Мишиной лекции сводится к данной бинпоиском по K. Мы также можем предположить, что рюкзаки имеют разные размеры. Размер i-го рюкзака W_i.

2. Общий алгоритм решения: отсортировали вещи в каком-то порядке, отсортировали рюкзаки в каком-то порядке, жадно перебираем рюкзаки и в каждый рюкзак каждую вещь или кладем (если влезает), или не кладем (вещи мы, конечно, перебираем всегда в одном и том же порядке).

3. Чтобы получить классное решение, сделаем так: будем запускаться четыре раза (можно и вещи, и рюкзаки сортировать как по убыванию размера, так и по возрастанию). Каждый из четырех запусков — это перебор, основанный на описанной выше жадности. Каждому из четырех переборов дадим по 0.25 секунд.

Задачи и разборы

Задача A. Сервера

Имя входного файла:	`a.in`
Имя выходного файла:	`a.out`
Ограничение по времени:	0.5 с
Ограничение по памяти:	256 Мб

Примечание

Источник — Всесибирская олимпиада, очный тур (29 ноября 2009).server

Компьютерная сеть в некотором доме строилась по принципу присоединения нового компьютера к последнему из уже подключенных. Никакие два компьютера, будучи подключенными в сеть, между собой дополнительно никак не связывались. Таким образом, в сеть были объединены последовательно N компьютеров. Соседи обменивались информацией между собой, но в какой-то момент поняли, что им нужны прокси-серверы. Компьютерное сообщество дома решило установить прокси-серверы ровно на

K компьютеров. Осталось только решить, какие именно компьютеры выбрать для этой цели. Главным критерием является ежемесячная стоимость обслуживания серверами всех компьютеров.

Для каждого компьютера установлен тариф его обслуживания, выраженный в рублях за метр провода. Стоимость обслуживания одного компьютера каким-то сервером равна тарифу компьютера, умноженному на суммарную длину провода от этого компьютера до сервера, которым он обслуживается.

Ваша задача — написать программу, которая выберет такие K компьютеров, чтобы установить на них прокси-серверы, что общие затраты на обслуживание всех компьютеров были бы минимальными

Формат входного файла

В первой строке входного файла записано два целых числа N и K — количество компьютеров в сети и количество прокси-серверов, которые нужно установить ($1 \leq K \leq N \leq 2000$).

Все компьютеры в сети пронумерованы числами от 1 до N по порядку подключения.

Во второй строке записано одно целое число T_1 — тариф обслуживания первого компьютера.

В следующих N – 1 строках записано через пробел по два целых неотрицательных числа L_i, T_i — информация об остальных компьютерах в сети по порядку номеров. L_i — длина провода, соединяющего i — ый компьютер с соседним с меньшим номером, T_i — тариф обслуживания данного компьютера ($2 \leq i \leq N$). Все L_i и T_i не превышают 10^6.

Формат выходного файла

В первую строку выходного файла необходимо вывести одно целое число — минимальную стоимость обслуживания всех компьютеров всеми серверами. Во второй строке должны быть записаны через пробел K номеров компьютеров, на которые необходимо установить серверы. При существовании нескольких вариантов размещения разрешается вывести любой.

Пример

a.in	a.out
3 1 10 2 2 3 3	19 1
3 2 10 2 2 3 3	4 1 3

Разбор задачи A. Сервера

Это сложная задача.

∘ Предподсчитаем $cost[L, R]$ — стоимость покрыть отрезок $[L..R]$ одним сервером.

∘ Будем считать динамику $f[i, k]$ — стоимость покрыть i серверами первые k компьютеров. $f[i, k] = f[i-1, prev]+cost[prev+1, k]$ нужно только найти $prev$. Можно перебрать все от 1 до $k-1$, тогда получится решение за $O(N^3)$. Можно лучше $prev[i, k-1] \le prev[i, k] \le prev[i+1, k]$. Т.е. перебирать нужно от и до уже посчитанных величин $prev$. Утверждается, что суммарная длина таких отрезков — $O(N^2)$ (доказательство было на устном разборе, также его можно найти где-то в трехтомнике Кнута).

Задача B. Wordperiod

Имя входного файла:	`b.in`
Имя выходного файла:	`b.out`
Ограничение по времени:	1 с
Ограничение по памяти:	256 Мб

Примечание

Источник — чемпионат MIPT (май 2011).wordperiod

Назовём периодом слова s такое слово t, длина которого не превосходит длины слова s и для которого существует такое натуральное число k, что слово s является префиксом слова t^k (то есть слова, полученного конкатенацией k копий слова t). Например, периодами слова **xyzxyzx** являются слова **xyz**, **xyzxyz**, **xyzxyzx**.

Пусть имеется некоторое слово w длины l. Рассмотрим l слов длины $l-1$, $i-$е из которых получено из слова w вычёркиванием его $i-$й буквы. Для каждого из этих слов найдём период наименьшей длины. Выведите наименьшее из получившихся l чисел.

Формат входного файла

В первой строке входного файла задано целое число d ($1 \le d \le 10$) — количество тестовых примеров. В последующих d строках заданы тестовые примеры, по одному на строку. В начале $i-$го тестового примера идёт число n_i ($2 \le n_i \le 200\,000$) — длина l слова w. Далее через пробел следует слово w, состоящее из l строчных латинских букв.

Формат выходного файла

Для каждого тестового примера выведите в отдельной строке одно число — минимальную длину периода слова, полученного выбрасыванием из исходного слова одной буквы.

Пример

b.in	b.out
1 8 ababcaba	2

Пояснение

Для слова w из тестового примера имеем следующие слова, наименьшие периоды и их длины:

- **babcaba — babca** — длина 5;
- **aabcaba — aabcab** — длина 6;
- **abbcaba — abbcab** — длина 6;
- **abacaba — abac** — длина 4;
- **abababa — ab** — длина 2;
- **ababcba — ababcb** — длина 6;
- **ababcaa — ababca** — длина 6;
- **ababcab — ababc** — длина 5.

Соответственно, наименьшая из длин равна 2, что и является ответом на тестовый пример.

Разбор задачи B. Wordperiod

Давайте научимся за $O(\log N)$ отвечать на вопрос, подходит ли период T.

Сперва зафиксируем `max` префикс и суффикс с таким периодом. Префикс `max` $i : S[1..i-T] = S[T+1, i]$. Суффикс `max` $i : S[N-i+T..N] = S[N-i, N-T]$. Теперь, попробуем эти 2 куска склеить.

Для всего этого нам нужны бинпоиск по i и предподсчитанные хэши, чтобы сравнивать строки на равенство за $O(1)$.

Задача C. Arithmetic

Имя входного файла:	`c.in`
Имя выходного файла:	`c.out`
Ограничение по времени:	0.5 с
Ограничение по памяти:	256 Мб

Примечание

Источник — XV чемпионат СПбГУ (22 октября 2006).arithmetic

Вы только что получили задание от Kitten Computing, известного разработчика программного обеспечения. Вы не можете поверить в ваше счастье — множество ваших друзей подавали туда заявку раньше, но они либо провалились на интервью, либо не смогли выполнить их первое задание и были уволены в первый месяц их работы в Kitten Computing .

Сейчас вы и сами получили ваше первое задание, в команде которая разрабатывает программное обеспечение для нового арифметического кодирования. Напомним, что во время арифметического кодирования, строка, которую нужно закодировать, переходит в интервал (α, β) реальной строки, а затем любая дробь p/q на этом интервале и есть результат кодирования.

Ваше задание — найти эту дробь p/q , так чтобы числитель и знаменатель были как можно меньше, для заданных рациональных чисел α и β. Вам не нужно беспокоится о происхождении α и β : программное обеспечения для создания этих чисел написал другой человек из вашей команды.

Формат входного файла

Каждая строка входного файла содержит четыре целых числа P_1, Q_1, P_2, Q_2, таких, что $\alpha = P_1/Q_1$ и $\beta = P_2/Q_2$. Все числа неотрицательные и не превышают 10^{18}, знаменатели Q_1 и Q_2 отличны от нуля. Гарантируется, что $\alpha < \beta$. Во входном файле содержится не более 10^4 строк.

Формат выходного файла

Для каждой строки ввода вы должны выдать два натуральных числа — P и Q, разделенные одним пробелом, такие, что $\alpha < P/Q < \beta$. При этом Q должно быть минимально возможным. Если есть несколько оптимальных Q, выводите дробь с минимальным P.

Пример

c.in	c.out
3 10 7 20 0 1 1 1	1 3 1 2

Разбор задачи C. Arithmetic

Все дроби от 0 до 1 можно получить следующим образом. Сперва у нас есть две дроби $\frac{0}{1}$ и $\frac{1}{1}$. Затем из двух соседних дробей $\frac{a}{b}$ и $\frac{c}{d}$ мы можем получить дробь $\frac{a+c}{b+d}$. Если получить таким образом все дроби со знаменателями не больше N, то:

○ В последовательности встречаются все несократимые дроби со знаменателями не больше N.

○ Никаких других (например, сократимых) дробей в последовательности нет.

○ Дроби выписаны в порядке возрастания.

Решением же предложенной задачи является спуск по дереву дробей, описанному выше. Изначально мы знаем, что ответ лежит между $(0,1)$и$(1,1)$. В каждый момент времени у нас есть пара (a,b) и (c,d) — дроби между, которыми лежит ответ. Переходим к паре $(a+k\cdot c, b+k\cdot d)$ или $(k\cdot a+c, k\cdot b+d)$ в зависимости от того, в какую сторону выгодней идти.

P.S. Это только один из взглядов на задачу. Также известен такой подход “рассмотрим дроби, как вектора на плоскости и на очередном шаге будем определенным образом делать замену координат”.

Задача D. Таможенные правила

Имя входного файла:	`d.in`
Имя выходного файла:	`d.out`
Ограничение по времени:	0.5 с
Ограничение по памяти:	256 Мб

Деньги на бочку!

Из заявки Диогена на грант

Примечание

Источник — XIX чемпионат СПбГУ (9 декабря 2007).tax
Author: Vitaly Valtman
Description Author: Vitaly Valtman
Text Author: Oleg Hristenko

Наконец, приз, составляющий N золотых монет, был получен, и команда Поссилтума отбыла назад, в измерение Пент. По дороге к Аазу, Скиву, Гвидо и Нунцио присоединились попутчики из других команд, прибывших из Пента. Так что возвращавшаяся делегация составляла K человек.

Однако по пути им встретилось достаточно неприятное королевство, таможенное законодательство которого было не только очень строгим, но и очень запутанным.

В случае, если у прибывшего путешественника оказывалось с собой менее A золотых монет, путешественник отправлялся под арест за нищенство.

В случае, если путешественники прибыли группой, и у каких-то двух путешественников в группе количество золотых монет равно $t \cdot a$ и $t \cdot b$ соответственно, где t — целое число, большее единицы, а a и b — натуральные числа, то считается, что имел место сговор с целью подрыва местной валюты (установления её курса в t), и оба путешественника отправляются под арест за экономическую диверсию. Естественно, что с конфискацией орудия преступления (то есть золота).

Так что перед Аазом и Скивом встала задача — можно ли распределить сумму приза между всеми участниками делегации так, чтобы никто не был арестован при прохождении таможенного досмотра?

Формат входного файла

Входной файл содержит три натуральных числа. $1 \le N \le 1\,000\,000\,000$, $2 \le K \le 10$ и $1 \le A \le N$.

Формат выходного файла

Если пройти таможню без ареста можно, то в первой строке выведите слово «YES», а во второй — K чисел a_i — выданное каждому участнику количество золотых монет.

В противном случае выведите единственную строку «NO».

Примеры

d.in	d.out
6 3 1	YES 1 2 3
6 3 2	NO
6 2 1	YES 1 5

Разбор задачи D. Таможенные правила

Нам нужно выбрать K взаимнопростых чисел a_i, в сумме дающих N. Также нужно, чтобы все числа были $\geq M$.

Напишем перебор. Пусть мы уже взяли i чисел, тогда $a_i \leq a_{i+1} \leq \frac{N'}{K-i}$ (где N' — остаток, т.е. $N - \sum_{j=1}^{i} a_j$. Первое неравенство говорит о том, что числа мы перебираем в порядке возрастания, второе о том, что если сумма $K - i$ чисел равна N', то минимальное из них явно не больше $\frac{N'}{K-i}$.

Получилось следующее решение: `for` $a_i := \frac{N'}{K-i}$ `downto` a_i. Здесь кроме всего прочего важно, что перебор идет в убывающем порядке.

Задача E. Covering Points

Имя входного файла:	`e.in`
Имя выходного файла:	`e.out`
Ограничение по времени:	3 с
Ограничение по памяти:	256 Мб

Примечание

Источник — XX чемпионат СПбГУ (23 март 2008).covers
Author: Andrew Lopatin
Text Author: Andrew Lopatin
Description: DP

Сегодня, Вася отправляется в научно-исследовательский институт сфер и кругов (RISC). Этот институт изучает все задачи, связанные со сферами и кругами, и теперь Васю просят что-то сделать с кругами.

Даны N точек на плоскости, необходимо покрыть их все K кругами минимально возможного радиуса. Кругам разрешено касаться и пересекаться друг с другом. Не могли бы вы помочь Васе справиться с заданием?

Формат входного файла

Входной файл состоит из одного или нескольких тестов. Каждый тест начинается со строки, содержащей два целых числа N и K ($1 \le N \le 15$, $1 \le K \le N$), а затем N строк, каждая из которых описывает одну точку её целыми координатами x_i и y_i. Точки могут совпадать. Все координаты не превышают 1000 по абсолютной величине. Ввод завершается тестом $N = K = 0$, который учитывать не нужно. Общая сумма чисел N по всем тестам в одном входном файле не превышает 150.

Формат выходного файла

Для каждого теста сначала выведите минимальный радиус, затем приведите пример покрытия. Если есть несколько решений — выводите любое. Радиусы нужно вывести с точностью не менее 6 знаков после запятой. Чекер будет осуществлять проверку “попала ли точка в круг” с точностью 10^{-6}. Мы рекомендуем выводить числа с максимальной точностью.

Заметьте, что круг с нулевым радиусом — точка.

Примеры

e.in

```
3 2
0 0
0 1
0 2
3 1
0 0
0 1
1 0
0 0
```

e.out

```
Case 1: The minimal possible radius is 0.5
  circle 1 at (0.0, 0.5)
  circle 2 at (0.0, 2.0)

Case 2: The minimal possible radius is
0.7071067811865476
  circle 1 at (0.5, 0.5)
```

Разбор задачи E. Covering Points

Первое решение, которое приходит в голову: бинпоиск по ответу, получить $O(N^2)$ возможно полезных кругов с радиусом R и написать динамику по подмножествам с $\Theta(2^N \cdot N)$ состояниями за время $\Theta(2^N \cdot N^3)$.

Основная идея — бинпоиск не нужен. Можно изначально посчитать $O(N^3)$ возможно полезных кругов с разными радиусами, отсортировать их в порядке возрастания радиуса и считать динамику $minK[p = 0..2^N] - 1$ — минимальное количество кругов, которыми можно покрыть множество точек p. Теперь будем добавлять наши $O(N^3)$ кругов в порядке возрастания радиуса. Добавление одного круга (пересчет динамики) работает за $O(2^N)$. Если в какой-то момент множество $2^N - 1$ покрыто не более, чем K кругами, то ответ — радиус последнего добавленного круга.

Задача F. Гамильтонов цикл в полном графе

Имя входного файла:	`f.in`
Имя выходного файла:	`f.out`
Ограничение по времени:	0.5 с
Ограничение по памяти:	256 Мб

Примечание

Источник— из задач спецкурса 3-го курса СПбГУ (весна 2011).fullham
Original idea : Folklore
Text : Sergey Kopeliovich
Tests : Sergey Kopeliovich

Дан граф из N вершин, в котором степень любой вершины не меньше $\frac{N}{2}$. Ваша задача — найти гамильтонов цикл.

Формат входного файла

На первой строке входного файла записано целое число N ($3 \le N \le 4\,000$) — количество вершин в графе. На следующих N строках записана матрица смежности. Т.к. матрица смежности симметрична, а на диагонали всегда стоят нули, на i-й строке записаны $i - 1$ символ — нули и единицы. Если j-й символ i-й строки равен единице, значит есть ребро между вершинами i и j.

Гарантируется, что в графе есть гамильтонов цикл и, что степень каждой вершины не меньше $\frac{N}{2}$.

Формат выходного файла

Выведите перестановку из N чисел — номера вершин в порядке гамильтонова цикла.

Пример

f.in	f.out
4 1 11 101	1 2 3 4

Разбор задачи F. Гамильтонов цикл в полном графе

Сперва построим гамильтонов цикл в полном графе. Это $1, 2, \cdots, N$. Теперь начнем удалять ребра, которых на самом деле нет. Удалять будем по одному.

Подробно можно прочитать тут:
http://195.19.228.2/~sk1/download/books/Asanov.djvu.

Задача G. PreQueL

Имя входного файла:	`g.in`
Имя выходного файла:	`g.out`
Ограничение по времени:	2 с
Ограничение по памяти:	256 Мб

Примечание

Источник — Дальневосточные областные соревнования NEERC
Far Eastern NEERC Subregional Contests.prequel

В некоторых упрощенных СУБД называемых PreQueL, единственный допустимый тип названия - CHAR (1) (один символ), а кроме того, его название должно состоять только из английских заглавных букв (от A до Z). Таблица может содержать до 9 столбцов, пронумерованных от 1 до 9. Таблицы называются строчными английскими буквами (от a до z).

Единственно возможный запрос к базе данных сначала `соединяет` все таблицы (см. пояснение к задаче в конце условия); потом выбирает некоторые строки в соответствии с условиями. Бывают условия двух видов: `<column> = <value>` или `<column1> = <column2>`, например,

`a2=A` или `b1=c4`. Все условия должны выполняться одновременно, как если бы они были соединены с помощью оператора AND.

Вы должны написать обработчик запросов к PreQueL, который, с учетом данных ему таблиц и набора условий, будет выдавать результат запроса, то есть те строки которые удовлетворяют всем условиям. Строки в ответе должны быть упорядочены лексикографически по алфавиту.

Формат входного файла

Первая строка входного файла содержит два целых числа — количество таблиц T и количество условий, D.

Начиная со второй строки даются T таблиц. В первой строке описания таблицы даются размеры таблицы — RN и CN. Далее идёт RN строк, состоящих ровно из CN символов каждая. После таблицы идут D строк с условиями. Ограничения: $1 \le T \le 26$, $1 \le D \le 50$, $1 \le C \le 9$, $1 \le R \le 1000$.

Формат выходного файла

Выходной файл содержит строки результата, каждую строку таблицы выведите на отдельной строке файла. Гарантируется, что ни один входной запрос не выдаст более 1000 строк.

Пример

g.in	g.out
2 2 3 2 AX BX BY 2 3 ACD BCC a1=b1 a2=X	AXACD BXBCC

Пояснение

○ Строки в выведенном файле должны быть упорядочены по возрастанию.

○ Если мы `соединяем` таблицы `A` и `B` без применения каких-либо условий, число строк в результате будет $|A| \cdot |B|$.

○ Пусть у нас есть три таблицы (каждая таблица представлена в виде списка строк): {A, B}, {C, D}, {XX, YY, ZZ}. Их `соединением` будет

таблица из 12 строк {ACXX, ACYY, ACZZ, ADXX, ADYY, ADZZ, BCXX, BCYY, BCZZ, BDXX, BDYY, BDZZ}.

Разбор задачи G. PreQueL

Сразу применяем все $a_i = X$. Соединяем таблицы по одной по очереди. Когда соединяем две таблицы, применяем все $a_i = b_j$ такие, что обе таблицы (о которых идет речь в уравнении) уже включены в соединение.

Задача H. Sigma-функция на отрезке

Имя входного файла:	`h.in`
Имя выходного файла:	`h.out`
Ограничение по времени:	1.5 с
Ограничение по памяти:	256 Мб

Примечание

Источник из задач к экзамену 1-го курса СПБГУ (осень 2010).lrsigma

Нужно научиться считать $\sum_{i=L}^{R} \sigma(i)$. Где $\sigma(i)$ — сумма натуральных делителей числа i.

Формат входного файла

Последовательность из не более чем 10^5 запросов. Каждый запрос записан на отдельной строке. Формат запроса прост: числа L, R ($1 \le L \le R \le 5 \cdot 10^6$).

Формат выходного файла

Для каждого запроса нужно вывести одно число — $\sum_{i=L}^{R} \sigma(i)$.

Пример

h.in	h.out
3 10	83

Разбор задачи H. Sigma-функция на отрезке

Идея 1: посчитаем для каждого N F_N — сумму от 1 до N. Тогда ответ для отрезка $[L..R]$ — $F_R - F_{L-1}$.

Идея 2: сколько раз число x входит в $\sigma i, i = 1..N$? Ответ: N/x, т.е. нужно посчитать сумму по всем x величины $round\_down(N/x) \cdot x$.

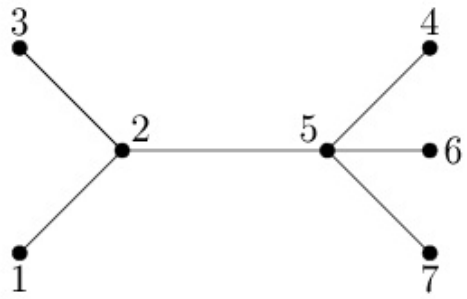
Задача I. Autotourism

Имя входного файла:	`i.in`
Имя выходного файла:	`i.out`
Ограничение по времени:	2 с
Ограничение по памяти:	256 Мб

Примечание

Источник — чемпионат MIPT (май 2011).autotourism

В Байтландии существуют n городов, соединённых n - 1 дорогами с двусторонним движением таким образом, что из каждого города можно проехать в любой другой по сети дорог. Длина каждой дороги равна 1 километру.



Бензобак автомобиля позволяет проехать без заправки m километров. Требуется выбрать маршрут, позволяющий посетить наибольшее количество различных городов без дозаправки. При этом начинать и заканчивать маршрут можно в произвольных городах.

Формат входного файла

В первой строке входного файла заданы два целых числа n и m ($2 \le n \le 500\,000$, $1 \le m \le 200\,000\,000$) — количество городов в стране и количество километров, которое автомобиль может проехать без дозаправки. В последующих n - 1 строках описаны дороги. Каждая дорога

задаётся двумя целыми числами a и b ($1 \le a, b \le n$) — номерами городов, которые она соединяет. Длина каждой дороги равна 1 км.

Формат выходного файла

Выведите одно число — максимальное количество городов, которое можно посетить без дозаправки.

Пример

i.in	i.out
7 6 1 2 2 3 2 5 5 6 5 7 5 4	5

Пояснение

5 городов можно посетить, например, по схеме $4 \to 5 \to 7 \to 5 \to 6 \to 5 \to 2$ или по схеме $3 \to 2 \to 1 \to 2 \to 5 \to 6 \to 5$.

Разбор задачи I. Autotourism

Используя $2(N-1)$ литров бензина мы можем обойти все дерево.

Если длина самого длинного пути в дереве = K ребер, то мы можем обойти $i+1$ вершину, потратив i литров бензина для всех i от 0 до K.

На самом деле, чтобы посетить больше чем $K+1$ вершин, на каждую следующую придется тратить 2 литра бензина. Т.е. чтобы посетить $i > K+1$ вершин нам нужно $(i-K-1) \cdot 2 + K$ литров бензина.

Задача J. Perfect Powers

Имя входного файла:	`j.in`
Имя выходного файла:	`j.out`
Ограничение по времени:	0.5 с
Ограничение по памяти:	256 Мб

Примечание

Источник — IX чемпионат СПБГУ (17 октября 2004).powers

Число x называется *perfect power*, если существуют такие целые числа a и b, что $x = a^b$, $b > 1$. Ваше задание — вычислить количество таких чисел внутри отрезка $A \le x \le B$.

Формат входного файла

Два целых числа A и B, по абсолютной величине не превосходящие 10^{14}.

Формат выходного файла

Одно целое число.

Пример

j.in	j.out
3 14	3
25 50	5

Разбор задачи J. Perfect Powers

В этой задаче важно было правильно разобрать случаи 0, 1, -1, отрицательные A и B, одно из двух отрицательно, второе положительно.

Математическая же идея решения следующая: научимся считать, сколько чисел x из отрезка $[1..N]$ имеют представление a^k. Для каждого k посчитаем f_k — число таких x (это просто $round\_down(N^{1/k})$). Теперь ответ — это $f_2 + f_3 + f_5 - f_6 + f_7 - f_10 + f_11 + f_13 - f_14 - f_15 \cdots$. Какой коэффициент перед f_i? Это *функция Мебиуса* от i (см. википедию).

Задача K. Spell Checker

Имя входного файла:	`k.in`
Имя выходного файла:	`k.out`
Ограничение по времени:	0.3 с
Ограничение по памяти:	256 Мб

Примечание

Источник — IX чемпионат СПБГУ (17 октября 2004).spell

Университет магии Kurufinwe в Alkarinqwe широко известен как один из самых сильных университетов магии в стране эльфов Laureling. Сейчас Архимаги этого университета вплотную подошли к разгадке тайны древнего языка магических элементалей Qwen'Iaur. У них уже есть список всех

слов этого языка, но комбинации этих слов не всегда дают работающее заклинание.

К счастью, Архимаги нашли глубоко в подземельях их замка древний артефакт Сарандол, также известный как Оракл. Легенды прошлого гласят, что этот артефакт способен создавать работающие заклинания! Но он был создан одним из величайших магов древности Белигуром Великим, и у него есть особой механизм защиты. Оракл выдает заклинание в виде строки без каких либо пробелов между словами, так что оно может быть прочитано только как алфавитная строка S длины L. Но это только первая проблема. Вторая заключается в том, что артефакт иногда генерирует последовательности, которые не могут быть составлены из слов языка Qwen'Iaur, т.е. не существует такой последовательности слов a_1, a_2, ..., a_k, что $S = a_1 a_2 \dots a_k$, где все слова a_i присутствуют в языке Qwen'Iaur (одно и то же слово может быть использовано более одного раза). Но если хотя бы одно такое представление существует, то можно заключить, что Оракл сгенерировал строку, которая является работающим заклинанием.

Ваше задание — написать магическую программу для вашего магического компьютера, которая проверит существование хотя бы одного такого представления.

Формат входного файла

В первой строке ввода содержится одно целое число $1 \le N \le 1\,000$ — количество слов в словаре. N следующих строк содержат слова языка Qwen'Iaur. Последняя строка входного файла — строка S, сгенерированная Сарандолом. Все слова в словаре имеют длину не более 100 символов. $L \le 10\,000$. Слова и строки не чувствительны к регистру и могут состоять только из английских букв и двух символов `'` и `` ` `` (эти символы имеют коды ASCII 39 и 96).

Формат выходного файла

Если строка может быть прочитана как рабочее заклинание, выведите `YES`, иначе выведите `NO`.

Пример

k.in	k.out
4 all bar Akh kar bakarkarbar	NO
4 all bar Akh kar AllakhAkhbar	YES

Разбор задачи K. Spell Checker

Решение = динамика.

Состояние и функция: f_i — можем ли мы получить i-й префикс заклинания.

Переход: храним все слова из словаря в боре. Пусть мы можем получить i-й перфикс, тогда можно спуститься по бору i-м суффиксом, таким образом за не более чем 100 операций мы поймем, какими словами словаря можно продолжить i-й префикс заклинания.

Задача L. Valsum

Имя входного файла:	l.in
Имя выходного файла:	l.out
Ограничение по времени:	0.5 с
Ограничение по памяти:	256 Мб

Примечание

Источник — тренировки студенческого кружка СПБГУ (23 марта 2009).valsum

Origin: 20100923 - First A2 Training - representation of integers and reals

Author: Ivan Kazmenko

Найдите сумму значений функции

$$f(x) = x + \frac{1}{x}$$

в нескольких целых точках.

Формат входного файла

В первой строке входного файла задано целое число n — количество точек ($1 \le n \le 50$). В следующей строке заданы n целых чисел $x_1, x_2, \ldots, x_n$ через пробел — точки, значения функции в которых нужно просуммировать ($0 < |x_i| \le 10^9$).

Формат выходного файла

Выведите в выходной файл одно число — сумму значений функции $f(x)$ в заданных точках. Ответ считается правильным, если абсолютная или относительная погрешность не превышает 10^{-9}.

Примеры

l.in	l.out
3 1 2 3	7.83333333333333
2 1 -1	0

Разбор задачи L. Valsum

Чтобы получить Accepted, достаточно отдельно просуммировать X_i (в целых числах, т.е. без погрешности), и отдельно просуммировать $1/X_i$ (уже в вещественных, но погрешность все равно не больше 1).

Задача M. Sprite

Имя входного файла:	`m.in`
Имя выходного файла:	`m.out`
Ограничение по времени:	0.5 с
Ограничение по памяти:	256 Мб

Примечание

Источник — школьная задача кружки ФМЛ 30 и ФТШ 566 (осень 2009).sprite

8б класс решил на слет взять много Спрайта. Для этого они собрались сконструировать переносной холодильник $a \times b \times c$, который будет вмещать ровно n кубических банок Спрайта размером $1 \times 1 \times 1$ Чтобы лимонад доехал как можно более холодным, они хотят минимизировать теплопотери; то есть минимизировать площадь поверхности.

Например, если емкость холодильника должна равняться 12, то возможны следующие варианты:

$322 \to 32$
$431 \to 38$
$621 \to 40$
$1211 \to 50$

В этом примере оптимальным является холодильник 322.

Помогите 8б найти оптимальный холодильник в общем случае.

Формат входного файла

Число n ($1 \le n \le 10^6$)

Формат выходного файла

Три числа a, b, c ($1 \le n \le 10^6$) — размеры наилучшего холодильника. Числа нужно выводить в порядке возрастания.

Пример

m.in	m.out
12	2 2 3
13	1 1 13
1000000	100 100 100

Разбор задачи M. Sprite

Пусть $a \le b \le c$, тогда $a^3 \le N, a \cdot b^2 \le N$, значит два цикла `for` по a и по b работают за $O(N^{1/2} \cdot N^{1/3})$

Задача N. Suffarray

Имя входного файла:	`n.in`
Имя выходного файла:	`n.out`
Ограничение по времени:	1 с
Ограничение по памяти:	256 Мб

Примечание

Источник из задач спецкурса 3-го курса СПБГУ (весна 2011).suffarray

Дана строка, требуется построить суффиксный массив для этой строки. Суффиксный массив — лексикографически отсортированный массив всех суффиксов строки. Каждый суффикс задается целым числом — позицией начала.

Строка s лексикографически меньше строки t, если есть такое i, что $s_i < t_i$ и $s_j = t_j$ для всех $j < i$. Или, если такого i не существует и строка s короче строки t.

Здесь s_i — код i-го символа строки s.

Формат входного файла

Файл состоит из единственной строки. Эта строка — **английский литературный текст**. Длина текста не превосходит 10^5. Коды всех символов в тексте от 32 до 127.

Формат выходного файла

Выведите N чисел — суффиксный массив данной строки.

Пример

n.in	n.out
99 bottles of beer.	14 3 11 19 2 1 15 4 16 17 9 13 8 12 5 18 10 7 6

Разбор задачи N. Suffarray

Решение — вызвать любую сортировку за $N \log N$. Внутри сортировки сравнивать строки в лоб — до первого несовпадения.

Задача O. Foxes

Имя входного файла:	o.in
Имя выходного файла:	o.out
Ограничение по времени:	0.3 с
Ограничение по памяти:	256 Мб

Примечание

Источник — IX чемпионат СПБГУ (17 октября 2004).foxes

В Чукотском национальном заповеднике есть n песцов. Каждый песец в определенной степени пушист. Пушистости песцов равномерно распределены от 0 до 1. Не так давно Чукотский национальный зоопарк направил запрос на $m \leq n$ наиболее пушистых песцов.

Общество охраны природы получило информацию, что только l песцов с наименьшей пушистостью будут выставлены в зоопарке, и $k = m - l$ наиболее пушистых будут использованы на шубы для глав зоопарка. Общество хочет подать в суд на зоопарк, и им нужна информация о песцах, которые будут убиты, и о песцах, которые будут оставлены в живых. На самом деле будет достаточно оценки среднего значения средней пушистости песцов в каждой группе. Напишите программу, которая вычисляет эти два числа.

Можно считать, что пушистости песцов независимы.

Формат входного файла

Три целых числа $1 \leq n, k, l \leq 10^8$.

Формат выходного файла

Первое число — это средняя пушистость песцов, которых собираются убить, второе — средняя пушистость песцов, которые будут выставлены в зоопарке. Числа выведите с точностью 6 знаков после запятой.

Пример

o.in	o.out
4 2 2	0.7000000000 0.3000000000
8 2 2	0.8333333333 0.6111111111

Разбор задачи O. Foxes

Оба требуемых в условии числа считались примерно одинаково:
Ответ 1 = $\frac{N}{N+1} + \frac{N-1}{N+1} + \cdots + \frac{N-K+1}{N+1}$.
Ответ 2 = $\frac{N-K}{N+1} + \frac{N-K-1}{N+1} + \cdots + \frac{N-K-L+1}{N+1}$.

Задача P. Cucarach

Имя входного файла:	`p.in`
Имя выходного файла:	`p.out`
Ограничение по времени:	0.3 с
Ограничение по памяти:	256 Мб

Примечание

Источник — тренировка команд СПБГУ (14 марта 2007).cucarach

Каждую полночь в квартире ученого Васи начинается ужас. Сотни . . . , о нет! ТЫСЯЧИ тараканов вылазят из каждой дырки к его обеденному столу, уничтожая все крошки и объедки! Вася ненавидит тараканов. Он очень долго думал и сделал Супер-ловушку, которая привлекает всех тараканов в большой зоне после активации. Он планирует активировать ловушку сегодня ночью. Но есть проблема. Эта очень эффективная ловушка с её очень большой зоной работы поглощает огромное количество энергии. Так что, Вася планирует минимизировать время работы этой ловушки. Он собрал информацию о всех местах, в которых живут тараканы. Также он заметил, что все тараканы двигаются только по линиям его скатерти с постоянной скоростью (мы можем предположить, что эта скорость равна 1, так что таракан, расположенный в одной из секций, может за 1 единицу времени переместится на любую соседнюю секцию (по вертикали или горизонтали)). Вася решил активировать его ловушку в одной из секций. Когда ловушка активирована, все тараканы будут двигаться к секции, содержащей ловушку, так быстро, как только смогут. Поэтому в любой момент времени после активации тараканы двигаются к секции, в которой находится ловушка, максимально уменьшая расстояние до неё. Если есть два пути с одинаковым расстоянием, то таракан выберет любой. Напишите программу для Васи, которая выбирает секцию, минимизирующую время, необходимое для уничтожения всех тараканов. Конечно, ваша программа будет считать, что скатерть будет плоскостью с декартовой системой координат и секции — точки с целыми координатами.

Формат входного файла

В первой строке входного файла содержится число мест, в которых живут тараканы N ($1 \le N \le 10000$). Следующие N строк содержат x и y — координаты мест, в которых живут тараканы (целые числа не больше 10^9 по абсолютному значению).

Формат выходного файла

Вам необходимо вывести только два целых числа x и y — координаты секции, которая минимизирует время работы. Если есть более одного решение — выведите любое из них.

Пример

p.in	p.out
2 1 1 3 3	2 2

Разбор задачи P. Cucarach

Идея 1: бинпоиск по ответу.

Идея 2: за время R таракан сможет оказаться в любой точке ромба со стороной R.

Идея 3: развернем все на 45 градусов $(x, y) \to (x + y, x - y)$ и проверим, что прямоугольники имеют точку пересечения.

Задачи на тему лекции

Задача A. Длиннейший путь

Имя входного файла:	`a.in`
Имя выходного файла:	`a.out`
Ограничение по времени:	1 с
Ограничение по памяти:	256 Мб

Дан ориентированный граф без циклов. Требуется найти в нем длиннейший путь.

Формат входного файла

Первая строка входного файла содержит два натуральных числа n и m — количество вершин и дуг графа, соответственно. Следующие m строк содержат описания дуг по одной на строке. Ребро номер i описывается двумя натуральными числами b_i и e_i — началом и концом дуги, соответственно ($1 \le b_i, e_i \le n$).

Входной граф не содержит циклов и петель.

$n \le 10\,000$, $m \le 100\,000$.

Формат выходного файла

Первая строка выходного файла должна содержать одно натуральное число — количество дуг в длиннейшем пути.

Пример

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

Задача B. Спички

Имя входного файла:	`b.in`
Имя выходного файла:	`b.out`
Ограничение по времени:	1 с
Ограничение по памяти:	256 Мб

Вы умеете выкладывать на столе некоторые цифры от 0 до k с помощью какого-то количества спичек. Ваша задача — построить максимально боль-

шое число из n спичек. Например, если вам нужно 6 спичек для цифры 0, 7 спичек для цифры 1 и 8 спичек для цифры 2, то максимальное число, которое можно сложить из 21 спички — это 210.

Формат входного файла

В первой строке даны числа n и k ($1 \le n \le 50$, $0 \le k \le 9$). Во второй строке задано $k+1$ число от 1 до 50 — количество спичек, требуемое на цифры от 0 до k. Гарантируется, что из n спичек можно выложить хотя бы одну из цифр.

Формат выходного файла

Выведите максимальное число, которое можно построить. Лидирующие нули в выводе недопустимы. Можно использовать не все n спичек, а только часть.

Пример

b.in
21 2 6 7 8
b.out
210
b.in
30 2 5 23 24
b.out
20
b.in
1 3 1 5 3 2
b.out
0
b.in
50 9 1 1 1 1 1 1 1 1 1 1
b.out
99999999999999999999999999999999999999999999999999

Примечание

У этой задачи есть решение за $O(nk)$.

Задача C. Ideal Path

Имя входного файла:	`c.in`
Имя выходного файла:	`c.out`
Ограничение по времени:	2 s
Ограничение по памяти:	256 Мб

New labyrinth attraction is open in New Lostland amusement park. The

labyrinth consists of n rooms connected by m passages. Each passage is colored into some color c_i. Visitors of the labyrinth are dropped from the helicopter to the room number 1 and their goal is to get to the labyrinth exit located in the room number n.

Labyrinth owners are planning to run a contest tomorrow. Several runners will be dropped to the room number 1. They will run to the room number n writing down colors of passages as they run through them. The contestant with the shortest sequence of colors is the winner of the contest. If there are several contestants with the same sequence length, the one with the *ideal path* is the winner. The path is the ideal path if its color sequence is the lexicographically smallest among shortest paths.

Andrew is preparing for the contest. He took a helicopter tour above New Lostland and made a picture of the labyrinth. Your task is to help him find the ideal path from the room number 1 to the room number n that would allow him to win the contest.

Примечание

A sequence $(a_1, a_2, \ldots, a_k)$ is lexicographically smaller than a sequence $(b_1, b_2, \ldots, b_k)$ if there exists i such that $a_i < b_i$, and $a_j = b_j$ for all $j < i$.

Формат входного файла

The first line of the input file contains integers n and m — the number of rooms and passages, respectively ($2 \le n \le 100\,000$, $1 \le m \le 200\,000$). The following m lines describe passages, each passage is described with three integer numbers: a_i, b_i, and c_i — the numbers of rooms it connects and its color ($1 \le a_i, b_i \le n$, $1 \le c_i \le 10^9$). Each passage can be passed in either direction. Two rooms can be connected with more than one passage, there can be a passage from a room to itself. It is guaranteed that it is possible to reach the room number n from the room number 1.

Формат выходного файла

The first line of the output file must contain k — the length of the shortest path from the room number 1 to the room number n. The second line must contain k numbers — the colors of passages in the order they must be passed in the ideal path.

Пример

c.in	c.out
4 6 1 2 1 1 3 2 3 4 3 2 3 1 2 4 4 3 1 1	2 1 3

Задача D. Strange People

Имя входного файла:	d.in
Имя выходного файла:	d.out
Ограничение по времени:	0.5 s
Ограничение по памяти:	4 Mb

Примечание

Source — Winter 2002 Author: Nikolay Marinov
http://acm.sgu.ru/problem.php?contest=0&problem=145

Once upon a time in one kingdom lives strange people. In this kingdom to travel from city to city there was a taxes. But people don't want to travel cheaper from one place to another. They want to travel from place to place by K-th simple expensive path between this cites. Your task is to find how much money they must prepare to go from one given city to another (also given) if they travel using K'th simple expensive path between these two cites. And you must print that path. (if there exist many solutions output any of them) It is guarantee that K'th expensive path between this cites exist. If there is road between city X and Y than there is road between Y and X. Between two cites exists no more than 1 road.

Ограничения

4 <= N - number of cites <= 100
M - number of roads
1 <= taxes between two cites <= 10000
1 <= K - the K'th expensive path to find <= 500

Формат входного файла

N M K

x y mon |
x y mon | ... | M lines
x y mon |
xs xe

where xs is start city and xe is end city x and y is pairs of integer that are numbers of two cites and mon is tax between this cites

Формат выходного файла

Z dc
x1 x2 x3 x4 .. xn

where Z is the cost of the K'th expensive path and x1 x2 ... xn is the path dc is number of cites in (x1 x2 x3 ... xn)

Пример

d.in	d.out
5 10 3 1 2 6 1 3 13 1 4 18 1 5 35 2 3 14 2 4 34 2 5 17 3 4 22 3 5 15 4 5 34 1 5	35 2 1 5

День пятый (22.02.2012 г.)
Контест Жукова Дмитрия Ивановича

Об авторе...

Жуков Дмитрий Иванович, родился в 1986 году. Окончил Орловский Государственный Технический Университет. Неоднократный победитель и призер международных, российских и региональных олимпиад и соревнований по программированию. Разработчик в компании Яндекс.

Основные достижения:

- дипломы 3-ей и 1-ой степени на всероссийской школьной олимпиаде по информатике;
- В 2008 году в составе команды ОрелГТУ занял 13-е место, что является лучшим результатом выступления команд ОрелГТУ на финалах Чемпионата Мира по программированию;
- победитель командной олимпиады по программированию II Международного Молодёжного фестиваля информационных технологий;
- 2-е место в личном зачёте и 3-е командное место в VIII Всесибирской олимпиаде по программированию имени Поттосина 2007 года, победитель индивидуального турнира I Открытой олимпиады ЮФУ по программированию;
- в составе команды ОрелГТУ стал финалистом ACM ICPC – 2007, 7-е место на зимних сборах 2008 года в Петрозаводске, стал победителем Южного четвертьфинала NEERC 2007 года, занял 3-е место на летних сборах 2007 года в Петрозаводске;
- финалист Google Code Jam 2009;
- занял 2-е место на онсайт-раунде VIII Открытого Кубка им. Е.В. Панкратьева по программированию;

- 3-е место и диплом I степени на соревнованиях SnarkNews Winter Series-2011.
- финалист Facebook Hacker Cup 2011.

Задача A. Арифметическая прогрессия

Имя входного файла:	`a.in`
Имя выходного файла:	`a.out`
Ограничение по времени:	1 с
Ограничение по памяти:	256 Мб

Требуется найти арифметическую прогрессию из натуральных чисел $a_1, a_2, \ldots, a_n$, с разностью d, обладающую свойством: $a_k^2 + 1$ — простое при всех $k = 1, 2, \ldots, n$. Среди всех таких прогрессий следует выбрать ту, которая состоит из максимального числа элементов.

Разность прогрессии d означает, что для всех $k = 2, 3, \ldots, n$ выполняется $a_k - a_{k-1} = d$.

Формат входного файла

Входной файл содержит несколько тестов. В каждой строке записано целое число d — разность прогрессии ($1 \le d \le 9999$). В десятичной записи числа d цифра 0 не встречается. Все числа во входном файле различны.

Формат выходного файла

Для каждого теста выведите в выходной файл по одной строке, содержащей два числа. Первое число — максимальная длина арифметической прогрессии. Второе число — её первый элемент. Среди всех прогрессий максимальной длины выберите прогрессию с наименьшим первым элементом.

Пример

a.in	a.out
6 311	3 4 1 1

Разбор задачи A. Арифметическая прогрессия

Для решения задачи сначала оценим максимальную длину арифметической прогрессии. Пусть разность прогрессии d — нечётное число. Тогда

либо $a_1^2 + 1$ — чётное, либо $a_2^2 + 1$ — чётное. Оба они могут быть простыми только в случае, если $a_1^2 + 1 = 2$, т.е. $a_1 = 1$. В этом случае $a_3^2 + 1$ будет чётным числом больше двух, следовательно оно не будет простым. Таким образом, для нечётного d нам нужно проверить, является ли число $(1 + d)^2 + 1$ простым. Если да, то максимальная длина прогрессии равна двум, иначе — одному. Первый элемент прогрессии и в том, и в другом случае будет равен единице.

Теперь рассмотрим случай, когда d — чётное. Тогда оно может иметь следующие остатки от деления на 10, 2, 4, 6, 8. Выпишем остатки от деления на 5 чисел $0^2 + 1$, $1^2 + 1$, $\cdots$, $9^2 + 1$. Получим 1, 2, 0, 0, 2, 1, 2, 0, 0, 2. Рассмотрим случай $d \mod 10 = 2$ (остальные рассматриваются аналогично). Заметим, что среди чисел $a_1^2 + 1$, $(a_1 + d)^2 + 1$, $(a_1 + 2d)^2 + 1$ хотя бы одно делится на 5. Следовательно, максимальная длина прогрессии может быть равна трём, если $a_1 = 2$, а числа $(a_1 + d)^2 + 1$, $(a_1 + 2d)^2 + 1$ — простые. Иначе (если $a_1 \neq 2$) максимальная длина не больше двух.

Аналогичными рассуждениями получаем ограничение на длину для случаев $d \mod 10 = 4$, $d \mod 10 = 6$, $d \mod 10 = 8$ — 3, 3 и 2 соответственно (для случая $a_1 \neq 2$, если $a_1 = 2$, то на 1 больше).

Следовательно, в решении надо проверить, получается ли максимально возможная длина, если $a_1 = 2$, иначе находим перебором такое минимальное a_1, при котором длина прогрессии равна 2, 3, 3 или 2 для случаев $d \mod 10 = 2$, $d \mod 10 = 4$, $d \mod 10 = 6$ или $d \mod 10 = 8$ соответственно.

Такое решение будет работать быстро, если в начале программы один раз проверить на простоту числа вида $k^2 + 1$ для $k = 1, 2, \ldots, 30000$. Несложно проверить, запустив программу, что для $d < 10000$ в прогрессиях, являющихся ответом, не будет элементов, больших 30000.

Задача B. Мирные кони

Имя входного файла:	`b.in`
Имя выходного файла:	`b.out`
Ограничение по времени:	3 с
Ограничение по памяти:	256 Мб

Рассмотрим следующую задачу. Есть стандартная шахматная доска размером 8 на 8 клеток. Необходимо расставить на ней N коней, так, чтобы они не били друг друга, при условии, что на i-ой горизонтали должно стоять ровно r_i коней, а на j-ой вертикали должно стоять ровно c_j коней, где сумма всех r_i равна сумме всех c_j и равна N. Шахматный конь — это фигура, ход которой состоит в перемещении её на две клетки в горизонтальном

или вертикальном направлении, а затем ещё на одну — в перпендикулярном направлении. Считается, что под боем находятся только те клетки, в которых конь может закончить свой ход. В каждой клетке может стоять не более одного коня.

Требуется по заданным значениям r_i найти такие значения для c_j, чтобы исходная задача имела ровно одно решение.

Формат входного файла

В первой строке входного файла записаны восемь чисел — $r_1, r_2, \ldots, r_8$ ($0 \le r_i \le 8$).

Формат выходного файла

В выходной файл выведите через пробел искомые восемь чисел — $c_1, c_2, \ldots, c_8$ ($0 \le c_j \le 8$). Если невозможно подобрать требуемым образом эти числа, то выведите через пробел восемь чисел -1. Если подходящих ответов несколько, то выведите из них наименьший лексикографически (т.е. с минимально возможным значением c_1, если таких несколько, то с минимально возможным значением c_2 и т.д.).

Примеры

b.in	b.out
3 3 3 3 3 3 3 3	0 8 0 0 8 0 0 8
4 4 4 4 4 4 4 4	-1 -1 -1 -1 -1 -1 -1 -1
2 3 4 3 2 3 4 3	0 2 4 2 4 4 4 4

Разбор задачи B. Мирные кони

Задача предполагает переборное решение. Будем рекурсивно перебирать клетки доски в порядке возрастания строк, а при равенстве — по возрастанию столбцов.

Для каждой клетки будем проверять, можем ли мы в неё поставить коня так, чтобы его никто не бил и чтобы количество коней в i-ой строке не превышало r_i. Таким образом мы можем перебрать все корректные расстановки коней, удовлетворяющие ограничениям на количество фигур в строках.

Для каждой такой расстановки посчитаем количество коней в каждом из столбцов. Это будет какой-то вектор из восьми чисел. Его можно закодировать одним числом — взять сумму его элементов, умноженных на соответствующую степень девятки (каждое число в векторе от 0 до 8).

Заведём глобальный массив на 9^8 элементов. В этом глобальном массиве увеличиваем на единицу элемент с соответствующим индексом. После того, как перебрали все расстановки, осталось найти в этом массиве единичный элемент и получить по его индексу искомые значения для c_i, либо сказать, что решения не существует.

Задача C. Четыре точки

Имя входного файла:	`c.in`
Имя выходного файла:	`c.out`
Ограничение по времени:	1 с
Ограничение по памяти:	256 Мб

На плоскости заданы четыре точки. Требуется построить окружность, равноудалённую от всех этих точек.

Расстояние от точки A до окружности — это минимальное из расстояний от точки A до какой-либо точки B, лежащей на этой окружности.

Формат входного файла

В первой строке входного файла содержится одно число T — количество тестов ($1 \le T \le 10^4$). В каждой из следующих T строк записаны координаты четырёх точек x_1, y_1, x_2, y_2, x_3, y_3, x_4, y_4 — целые числа, не превосходящие 100 по абсолютному значению.

Формат выходного файла

Для каждого теста выведите в отдельной строке число K — количество различных окружностей, удовлетворяющих условию. Далее в K строках выведите по три числа — координаты центра окружности и её радиус (x_i, y_i, r_i). Если возможно построить более 42 различных окружностей, выведите вместо числа K строку "-1". Сами окружности в этом случае выводить не следует.

Допускается абсолютная погрешность не более 10^{-5}.

Пример

c.in	c.out
3 0 0 0 0 0 0 0 0 4 0 4 16 0 8 8 8 0 0 1 1 0 1 1 0	-1 5 4 11 8 4 5 8 -2 8 6 10 8 6 4 8 6 -1

Разбор задачи C. Четыре точки

Заметим, что если исходные точки лежат на одной окружности, то в ответ входит бесконечное количество окружностей. В частности, если есть две совпадающие точки, то в ответ нужно вывести -1.

Если точки не лежат на одной окружности, то искомая окружность разбивает их на два множества — те, что лежат внутри, и те, что лежат снаружи. Получаем два случая — когда с одной стороны лежат три точки, а с другой — одна, и когда с обеих сторон лежит по две точки.

Случай первый. Переберём, какие три точки лежат с одной стороны. Обозначим их A, B, C, а четвёртую точку — D, Очевидно, что центр искомой окружности совпадает с центром окружности, описанной вокруг треугольника ABC. Чтобы его найти, нужно взять точку пересечения серединных перпендикуляров к отрезкам AB и AC. Если эти перпендикуляры не пересекаются, то вокруг этих точек нельзя описать окружность. Иначе, запоминаем точку их пересечения P в качестве центра искомой окружности, а её радиус вычисляем, как $\frac{|PA|+|PD|}{2}$.

Случай второй. Переберём, какая точка лежит с той же стороны от искомой окружности, что и первая. Обозначим две точки с одной стороны A и B, а с другой — C и D. Центр окружности, ранвноудалённой от двух точек, при условии, что они лежат либо обе внутри, либо обе снаружи, лежит на прямой, являющейся серединным перпендикуляром к отрезку, соединяющему эти две точки. Соответственно, центр искомой окружности может быть в точках, принадлежащих одновременно обоим серединным перпендикулярам для отрезков AB и CD.

Если эти перпендикуляры совпадают, то выводим в ответ -1. Если они не пересекаются, то к ответу ничего не добавляем. Иначе, как и в первом случае, берём в качестве искомого центра точку их пересечения P, а радиус вычисляем, как $\frac{|PA|+|PD|}{2}$.

Таким образом мы нашли все возможные окружности, либо узнали, что их существует бесконечное количество.

Задача D. Римские числа

Имя входного файла:	`d.in`
Имя выходного файла:	`d.out`
Ограничение по времени:	1 с
Ограничение по памяти:	256 Мб

Дана строка из символов '**I**', '**V**', '**X**', '**L**', '**C**', '**D**' и '**M**'. Требуется разбить её на несколько строк так, чтобы каждая получившаяся строка представляла собой корректное римское число, а сумма этих чисел была бы минимально возможной.

Корректным римским числом будем называть число, которое получается по следующим правилам. Сначала выписывается символ '**M**' столько раз, сколько целых тысяч содержится в числе. Например, в числе 2045 содержится две целых тысячи, и его запись начнётся с символов "`MM`". После этого берём остаток от деления его на 1000. В зависимости от того, столько целых сотен содержит результат, добавляем к римской записи следующую строку: "`C`" — для одной сотни, "`CC`" — для двух, "`CC`", "`CCC`", "`CD`", "`D`", "`DC`", "`DCC`", "`DCCC`", "`CM`" — для трёх, четырёх, ..., для девяти сотен соответственно. Остаток от деления числа 2045 на 1000 равен 45. Целых сотен в нём нет, поэтому к римской записи ничего не добавляем.

Далее берём остаток от деления числа на 100 и, в зависимости от количества целых десятков в результате, по тем же правилам добавляем к римской записи "`X`", "`XX`", "`XXX`", "`XL`", "`L`", "`LX`", "`LXX`", "`LXXX`" или "`XC`" (либо ничего не добавляем, если целых десятков нет). Соответственно, для числа 2045 мы должны добавить к римской записи строку "`XL`".

После этого берётся остаток от деления на 10 исходного числа, и, в зависимости от результата, добавляется "`I`", "`II`", "`III`", "`IV`", "`V`", "`VI`", "`VII`", "`VIII`", "`IX`" — для $1, 2, \ldots, 9$ соответственно, либо, в случае нулевого остатка, ничего не добавляется. Для числа 2045 по этому правилу мы должны прибавить строку "`V`".

В итоге для числа 2045 получаем римскую запись "`MMXLV`".

Также на римские числа вводится дополнительное ограничение, что никакой символ в их записи не может встречаться более трёх раз подряд. То есть будем считать, что корректная римская запись существует только у чисел от 1 до 3999 включительно.

Формат входного файла

Во входном файле содержится одна непустая строка длиной не более 20000 символов, состоящая из заглавных латинских букв, указанных в условии (‘**I**’, ‘**V**’, ‘**X**’, ‘**L**’, ‘**C**’, ‘**D**’ и ‘**M**’).

Формат выходного файла

В выходной файл выведите одно число — минимально возможную сумму чисел.

Примеры

d.in	d.out
IVI	5
IIIIV	7
CDILMVX	1466
M	1000
IX	9

Разбор задачи D. Римские числа

Задачу проще всего решать при помощи динамического программирования. Обозначим d_i — минимальную сумму, которую можно получить, разбивая на подстроки префикс исходной строки S длины i. Будем считать $d_0 = 0$.

Теперь будем в цикле по i от 1 до длины строки S вычислять значение d_i. Для этого можно перебирать подстроки S, заканчивающиеся в i-ом символе, и для каждой такой строки проверять, является ли она корректной римской записью какого-либо числа x от 1 до 3999. Если находим такое число x с римской записью длины l, то обновляем значение динамики $d_i = \min(d_i, d_{i-l} + x)$.

Но можно заметить, что любое римское число можно разбить на сумму его тысяч, сотен, десятков и единиц и складывать их по отдельности. Ответ от этого не изменится. Также можно заметить, что на отдельные слагаемые можно разбивать любые строки, кроме “IV”, “IX”, “XL”, “XC”, “CD” и “CM”, которые соответствуют числам 4, 9, 40, 90, 400 и 900.

Следовательно, в решении достаточно инициализировать на каждом шаге d_i, как $d_{i-1} + val(s_i)$, где $val(a)$ — число, которое в римской записи представляет собой строку, состоящую из одного единственного символа a. И затем проверять, если последние два символа образуют строку из набора, указанного выше, то обновляем d_i значением $d_{i-2} + f(s_{i-1}, s_i)$, где $f(a, b)$ равно 4, 9, 40, 90, 400 или 900 соответственно.

Ответом на задачу будет значение d_n, где n — длина строки S.

Задача E. Камни

Имя входного файла:	`e.in`
Имя выходного файла:	`e.out`
Ограничение по времени:	2 с
Ограничение по памяти:	256 Мб

Двое играют в игру. Изначально есть одна кучка из N камней. За один ход разрешается взять любую кучку и разбить её на несколько так, чтобы количества камней в новых кучках отличались бы не менее, чем на K. Например, при $K = 2$ кучку из 9 камней можно разбить на кучки $(1, 8)$, $(2, 7)$, $(3, 6)$ и $(1, 3, 5)$. Тот, кто не сможет сделать очередной ход, проигрывает. При заданном N определите все K ($0 \leq K \leq N$), при которых выигрывает второй игрок.

Формат входного файла

В первой строке входного файла записано натуральное число N ($1 \leq N < 100$).

Формат выходного файла

В первую строку выходного файла выведите общее количество искомых чисел, а во вторую — сами числа в порядке возрастания.

Пример

e.in	e.out
9	3 6 8 9

Разбор задачи E. Камни

Чтобы найти все такие K, при которых выигрывает второй игрок, определим победителя игры для каждого значения K от 0 до N.

Пусть значение K зафиксировано. Тогда для определения победителя воспользуемся функцией Гранди. Для тех позиций, из которых нельзя сделать ход, значение функции гранди равно нулю. Для остальных позиций нам следует перебрать все возможные ходы, которые может сделать игрок, посчитать xor функций гранди для всех независимых игр, на которые распадается исходная игра, запомнить это число для каждого хода и выбрать в качестве значения функции Гранди для текущей позиции минимальное

целое неотрицательное число, которое не встречается в множестве запомненных чисел.

Позицией X считается одна кучка, состоящая из X камней. Будем вычислять функцию Гранди для всех позиций по порядку от 0 до N. Для того, чтобы перебрать все ходы из позиции X, можно, например, перебрать все различные разбиения числа X на слагаемые, где любые два слагаемых отличаются не меньше, чем на K. Затем для каждого разбиения посчитать функцию Гранди для получившейся суммы игр. Для проигрышных позиций значение функции Гранди будет равно нулю. В случае ненулевых значений позиция будет выигрышной.

Такое решение работает при заданных органичениях достаточно быстро при $K > 0$. Случай $K = 0$ можно рассмотреть отдельно. Очевидно, что если $K = 0$, то при $X = 1$ первый игрок проигрывает, так как сразу не может сделать ход. А при больших X первым же ходом можно разбить исходную кучку на X штук, в каждой из которых будет ровно по одному камню. Из такой позиции второй игрок не сможет сделать ход и первый побеждает.

Такое решение в данных ограничениях укладывается по времени. Предлагается подумать, можно ли ускорить получение значений функции Гранди для всех возможных ходов из текущей позиции заменой полного перебора всех разбиений на слагаемые на перебор с меморизацией.

Задача F. Две строки

Имя входного файла:	`f.in`
Имя выходного файла:	`f.out`
Ограничение по времени:	1 с
Ограничение по памяти:	256 Мб

Даны две строки S_1 и S_2, состоящие из строчных латинских букв. В каждой из этих строк выбирается некоторая начальная позиция (k_1 и k_2). Затем, начиная с этой позиции, по порядку выписываются все символы строки. После выписывания последнего символа переходят к первому и продолжают выписывать подряд все символы. Например, если строка S_1 равна “`CAB`”, а $k_1 = 2$ (нумерация начинается с единицы), то получаем строку “`ABCABCABCABCABC...`”. Если строка S_2 равна “`BCACAC`”, а $k_2 = 3$, то получаем строку “`ACACBCACACBCACAC...`”. Так получаем две бесконечные последовательности из символов. Обозначим их T_1 и T_2.

Требуется для заданных строк S_1 и S_2 найти такие k_1 и k_2, чтобы значе-

ние выражения $\lim\limits_{n\to\infty} \dfrac{\sum\limits_{i=1}^{n} eq(T_1[i], T_2[i])}{n}$ было максимально возможным (здесь $T_1[i]$ — i-ый символ строки T_1, $eq(a, b)$ равно 1, если символы a и b совпадают, или 0, если a и b различаются). То есть нужно максимизировать среднее количество совпавших символов в полученных последовательностях.

Формат входного файла

В первой строке входного файла записана непустая строка S_1. Во второй строке входного файла записана непустая строка S_2. Длина каждой строки не превосходит 2000 символов.

Формат выходного файла

В выходной файл выведите искомые числа k_1 и k_2 ($1 \le k_1 \le length(S_1)$, $1 \le k_2 \le length(S_2)$).

Примеры

f.in	f.out
CAB BCACAC	1 2
ABRACADABRA TEST	1 1
AAAB BABA	1 2

Разбор задачи F. Две строки

Обозначим длину первой строки — N, а длину второй строки — M. Также обозначим наибольший общий делитель чисел N и M как K. Для того, чтобы вычислить среднее количество совпавших символов в полученных бесконечных строках, достаточно посчитать это значение для первых $\frac{N \cdot M}{K}$ пар символов.

Понятно, что в первой строке можно начинать выписывать символы всегда с первого, а для второй строки нужно найти позицию, с которой следует начинать.

Допустим, мы начали с $(j+1)$-ой позиции. Разобьём первую строку на блоки длины K. Тогда в сумму совпавших войдёт возможное совпадение $(j+1)$-го символа второй строки с первым символом каждого из блоков первой строки. Также в сумму войдёт возможное совпадение $(j+2)$-го

символа второй строки со вторым символом каждого из блоков первой строки. И так далее, для $(j+K)$-го символа второй строки войдут в сумму возможные совпадения его с K-ым символом каждого блока первой. Далее для $(j+K+1)$-го символа получаем сравнения с теми же символами, что и для $(j+1)$-го. То есть символы второй строки, находящиеся на расстоянии K друг от друга, будут сравниваться с одними и теми же соответствующими им символами первой строки.

Следовательно, достаточно перебрать в качестве начальной позицию во второй строке, начиная с 1-ой и заканчивая K-ой.

Получаем, что нам необходимо выполнить в сумме $K \cdot \frac{N \cdot M}{K} = N \cdot M$ сравнений, что в данных ограничениях выполяется достаточно быстро, чтобы уложиться в отведённое время.

День шестой (23.02.2012 г.)
Контест Гольдштейна Виталия Борисовича

Об авторе...

Гольдштейн Виталий Борисович, ассистент и аспирант Московского Физико-Технического Института. Член научного комитета всероссийской олимпиады школьников по информатике и сборов по подготовке к международной олимпиаде школьников. Стажер компании Google (зима-весна 2008). Завуч Летней Компьютерной Школы (август, 2009) и Летней Компьютерной Школы (зима, 2009-2010). Член жюри Московского и Саратовского четвертьфинала ACM ICPC, член жюри полуфинала NEERC ACM ICPC. Разработчик компании Яндекс. Лектор курса “Алгоритмы и структуры данных” базовой кафедры Яндекса “Анализ данных” на факультете Инноваций и высоких технологий в МФТИ. Член тренерской группы проекта “Яндекс-тренировки” в Москве.

Основные достижения:

- серебряная медаль чемпионата мира по программированию среди студентов (ACM ICPC Tokyo 2007);
- участник финалов TopCoder Open, TopCoder Collegiate Open, Global Google Code Jam, Google Code Jam Europe;
- тренер призеров (4-е место, золотая медаль) чемпионата мира по спортивному программированию ACM-ICPC World Finals 2009, Швеция, Стокгольм.

Теоретический материал. Объединяемые структуры данных

На этой лекции мы рассмотрим как общий подход к объединяемым структурам данных, так и некоторые конкретные структуры.

Одно из первых, что начали объединять — это кучи. Обычная куча поддерживает три операции: добавление элемента, удаление минимума и нахождение минимума. Объединяемая куча приобретает еще одну операцию — объединение двух куч.

Левая куча

Левая куча является двоичным деревом, в котором выполнено свойство кучи, а именно для каждой вершины ее ключ меньше обоих детей.

Рангом вершины назовем кратчайшее расстояние до потомка у которого нет хотя бы одного сына

Куча называется левой или левацкой, если для каждой вершины у левого сына ранг не превосходит ранга правого сына

Не трудно заметить, что левая куча с рангом r содержит не менее 2^r вершин, а так же правый путь у такой кучи порядка $O(\log N)$, где N — количество элементов в куче.

Все операции в объединяемых структурах происходят через объединение. Так, чтобы добавить элемент нужно: создать кучу из одного элемента и объединить с текущей. Чтобы удалить минимум, нужно объединить кучи двух его сыновей.

Само же объединение происходит рекурсивно следующим образом:

- Если одна из куч пуста, то вернем вторую.
- Пусть ключ корня h_1 меньше ключа корня h_2, тогда подвесим вместо правого сына h_1 объединение ее правого сына и всей кучи h_2. Если при этой операции локально нарушилось свойство левой кучи, то поменяем двух детей.

Заметим, что в таком алгоритме мы на каждой итерации спускаемся к правому сыну одной из куч. Поэтому время работы будет равно сумме длин правых путей двух куч, а следовательно $O(\log N)$, где N — суммарное количество вершин в деревьях.

Общий подход к объединению

Пусть у нас есть изначально много отдельных элементов, а мы хотим делать множество операций по их объединению и отвечать на некоторые дополнительные запросы. Для многих таких операций подойдет система

непересекающихся множеств, но лишь до тех пор, пока не потребуется удалить элемент из одного из таких множеств.

Самый простой вариант — это при объединении двух структур данных просто перекладывать элементы одной структуры данных в другую. Однако асимптотика такого подхода будет большой, например, в случае, когда нам добавляют структуру из одного элемента и мы все время перекладываем данные в нее. Применим не сложную эвристику: будем перекладывать из меньшей структуры в большую. Казалось бы простая идея, однако дает существенное асимптотическое ускорение.

Каждый элемент при перекладывании будет попадать в множество вдвое большее, поэтому если он будет переложен K раз, то окажется в множестве размером 2^K. А следовательно количество перекладываний будет порядка $O(\log N)$, где N — общее количество задействованных элементов. С помощью удаления элементов из структуры можно добиться, чтобы один элемент перекладывался существенно большее число раз, однако амортизационный анализ покажет, что учетная стоимость объединения будет равна $O(\log N)$ операций добавления. Так для самых обыкновенных куч такой подход даст время работы объединения $O(\log^2 N)$.

Рассмотрим теперь вариант, когда структура данных не поддерживает ни удаления, ни добавления. Точнее добавление одного элемента равносильно полному перестроению структуры данных. Предыдущий подход окажется очень неэффективным. Разложим все наши N элементов на кучки по 2^K элементов так, чтобы не было кучек одинакового размера. Мы получим $O(\log N)$ кучек. Вместо хранения структуры данных над всеми данными построим по структуре данных в каждой кучке. Это немного накладывает ограничения на запросы, так как они должны быть объединяемы из разных подмножеств данных. Но скажем количество, минимум или максимум вполне поддерживаются. Объединение таких структур будет проходить по аналогии с сложением двоичных чисел. Если образовалось две кучи с одинаковым количеством элементов, то создадим из них кучку вдвое большую. И так пока присутствуют кучи одинакового размера. Основное отличие этого подхода состоит в том, что объединение кучи происходит за время построение структуры данных для их суммарного размера. Однако все элементы поучаствовавшие в объединении двух куч попали вдвое большую кучу. А значит каждый элемент поучаствует в объединении не более $O(\log N)$ раз.

Одним из хороших примеров внутренней структуры данных — это отсортированный массив. Добавление элементов в него — дорогая операция, а объединение — очень простая и работает за линейное время. Отсортированный массив поддерживает огромное количество операций, которые

можно объединять (количество элементов в отрезке).

Биномиальная куча

Более частным случаем такого подхода является биномиальная куча. Преимуществами над общим подходом является то, что вложенная структура (биномиальное дерево) умеет объединяться за $O(1)$ и поддерживает (и при этом очень удачно поддерживает) удаление элемента.

Биномиальное дерево

Биномиальное дерево порядка 0 — дерево из одной вершины (C_0). Биномиальное дерево порядка K — дерево C_{K-1} к корню которого в качестве еще одного сына подвешено еще одно дерево C_{K-1}.

В C_K ровно 2^K элементов, а детьми корня являются деревья $C_0, \ldots, C_{K-1}$.

Поэтому объединение любых двух биномиальных куч работает за $O(\log N + M)$. Удаление минимума заключается в объединении всей кучи с кучей состоящей из детей минимального элемента и работает соответственно за $O(\log N)$.

Декартово дерево

Еще один пример объединяемых структур данных — это декартовы деревья. Это структура данных у которой есть сразу два ключа x и y. По ключу x оно является деревом поиска, а по ключу y кучей.

Эти деревья не являются объединяемыми в общем смысле, так как могут быть объединены только два дерева, отрезки ключей x которых не пересекаются. Объединение происходит очень похоже на объединение левых куч. Пусть y корня дерева t_1 меньше корня дерева t_2. Тогда корнем итогового дерева будет корень дерева t_1. Тогда вместо правого сына t_1 подвесим объединение правого сына и всего дерева t_2. В противном случае будем делать симметрично.

Задачи и разборы

Задача A. Перекладываем камешки

Имя входного файла:	`a.in`
Имя выходного файла:	`a.out`
Ограничение по времени:	2 с
Ограничение по памяти:	256 Мб

Соня любит перекладывать камешки между кучками. Изначально у нее N кучек, а она иногда выбирает некоторую пару кучек и меньшую из

них перекладывает в большую. При равенстве Соня спрашивает какую кучку переложить у своего брата. Подскажите Соне какое наибольшее количество камней ей придется перенести в некотором варианте.

Формат входного файла

Задано единственное число N ($1 \le N \le 10000$).

Формат выходного файла

Выведите единственное число — ответ на задачу.

Примеры

a.in	a.out
1	0
5	5
7	9

Разбор задачи A. Перекладываем камешки

Данная задача может быть решена динамическим программированием. Пусть D_i — количество перекладываний, которое необходимо сделать при i кучках. Рассмотрим самое последнее объединение. Пусть в первой кучке j, а во второй кучке $i - j$ камней. Тогда количество операций будет равно $D_j + D_{i-j} + \min(j, i-j)$. Перебирая j получим ответ. Время работы $O(N^2)$.

Задача B. Это левая куча?

Имя входного файла:	`b.in`
Имя выходного файла:	`b.out`
Ограничение по времени:	2 с
Ограничение по памяти:	256 Мб

Потенциалом вершины в подвешенном двоичном дереве назовем кратчайшее расстояние до вершины у которой меньше двух детей. Дерево называется левым, если левый сын каждой вершины имеет не меньший потенциал, чем правый. Так же не должно существовать вершины, у которой есть правый, но нет левого сына.

По заданному двоичному дереву найдите наименьшую вершину, для которой нарушается свойство.

Формат входного файла

В первой строке задано количество вершин дерева N ($1 \le N \le 10^5$). Последующие N строк описывают индексы левого и правого сына соответственно l_i, r_i ($i < l_i \le n$, $i < r_i \le n$). Если l_i или r_i равно -1 — это означает, что такого сына нет.

Формат выходного файла

Выведите наименьший номер вершины, для которой нарушено свойство, или -1, если такой вершины не существует.

Примеры

b.in	b.out
3 2 3 -1 -1 -1 -1	-1
3 2 -1 3 -1 -1 -1	-1
3 2 -1 -1 3 -1 -1	2

Разбор задачи B. Это левая куча?

Для решения этой задачи необходимо посчитать потенциал каждой вершины и проверить выполнены ли описанные условия. Задача позволяет разобраться в том, что же такое левая куча.

Задача C. Высота левого дерева

Имя входного файла:	`c.in`
Имя выходного файла:	`c.out`
Ограничение по времени:	2 с
Ограничение по памяти:	256 Мб

Потенциалом вершины в подвешенном двоичном дереве назовем кратчайшее расстояние до вершины у которой меньше двух детей. Дерево на-

зывается левым, если левый сын каждой вершины имеет не меньший потенциал, чем правый. Так же не должно существовать вершины, у которой есть правый, но нет левого сына.

Высотой дерева называют количество вершин на самом длинном пути от корня до листа.

Для заданного количества вершин N найдите высоту самого высокого и самого низкого левого дерева.

Формат входного файла

Задано единственное число — количество вершин дерева N ($1 \le N \le 10^9$).

Формат выходного файла

Выведите два числа — высоту самого высокого и самого низкого левого дерева.

Пример

c.in	c.out
1	1 1

Разбор задачи C. Высота левого дерева

Левое дерево может полностью вытягиваться в цепочку, поэтому самое высокое дерево имеет высоту N. А самое низкое левое дерево как и любое другое двоичное дерево имеет высоту $\lceil \log_2 N \rceil$

Задача D. Правый путь левого дерева

Имя входного файла:	`d.in`
Имя выходного файла:	`d.out`
Ограничение по времени:	2 с
Ограничение по памяти:	256 Мб

Потенциалом вершины в подвешенном двоичном дереве назовем кратчайшее расстояние до вершины у которой меньше двух детей. Дерево называется левым, если левый сын каждой вершины имеет не меньший потенциал, чем правый. Так же не должно существовать вершины, у которой есть правый, но нет левого сына.

Правым путем дерева называют путь от корня, двигаясь только вправо. Длиной пути назовем количество его вершин.

Для заданного количества вершин N найдите длину самого длинного и самого короткого правого пути левого дерева.

Формат входного файла

Задано единственное число — количество вершин дерева N ($1 \le N \le 10^9$).

Формат выходного файла

Выведите два числа — длину самого длинного и самого короткого правого пути левого дерева.

Пример

d.in	d.out
1	1 1

Разбор задачи D. Правый путь левого дерева

Самая маленькая высота правого пути — это 1, а самая большая $\lfloor \log_2 N \rfloor$.

Задача E. Объединение прямоугольников 2

Имя входного файла:	`e.in`
Имя выходного файла:	`e.out`
Ограничение по времени:	2 с
Ограничение по памяти:	256 Мб

Объединение прямоугольников куда проще, если их стороны параллельны осям координат. Найдите площадь объединения заданного множества таких прямоугольников.

Формат входного файла

В первой строке задано количество прямоугольников N ($2 \le N \le 15$). На следующих N строках записаны прямоугольники координатами двух противоположных углов в формате: x_1 y_1 x_2 y_2. Координаты разделены пробелами и находятся в отрезке $[-10^9, 10^9]$, а так же являются целыми числами.

Допускаются вырожденные прямоугольники.

Формат выходного файла

Выведите площадь объединения прямоугольников округленную до ближайшего целого.

Примеры

e.in	e.out
2 0 0 10 10 20 5 5 20	300
2 0 0 10 10 20 15 15 20	125
2 0 0 10 10 3 5 5 3	100
3 0 0 10 10 13 5 5 13 12 12 15 15	147

Разбор задачи E. Объединение прямоугольников 2

Данную задачу проще всего решить по формуле включения-исключения. Нужно перебрать все подмножества и площадь пересечения прямоугольников, входящих в подмножество, прибавить к ответу с соответствующим знаком. Если количество прямоугольников в подмножестве нечетно, то площадь нужно прибавить, иначе вычесть.

Задача F. Дерево

Имя входного файла:	`f.in`
Имя выходного файла:	`f.out`
Ограничение по времени:	2 с
Ограничение по памяти:	256 Мб

Задано подвешенное дерево, содержащее n ($1 \le n \le 1\ 000\ 000$) вершин. Каждая вершина покрашена в один из n цветов. Требуется для каждой вершины v вычислить количество различных цветов, встречающихся в поддереве с корнем v.

Формат входного файла

В первой строке входного файла задано число n. Последующие n строк описывают вершины по одной в строке. Описание очередной вершины i

имеет вид p_i c_i, где p_i — номер родителя вершины i, а c_i — цвет вершины i ($1 \le c_i \le n$). Для корня дерева $p_i = 0$.

Формат выходного файла

Выведите n чисел, обозначающих количества различных цветов в поддеревьях с корнями в вершинах $1, \ldots, n$.

Примеры

f.in	f.out
5 2 1 3 2 0 3 3 3 2 1	1 2 3 1 1
1 0 1	1
2 0 1 1 1	1 1
2 0 1 1 2	2 1

Разбор задачи F. Дерево

Для решения этой задачи можно использовать объединяемый set, перекладывая из меньшего в большее. Время работы будет $O(N * \log^2 N)$ на объединение и $O(N * \log N)$ на поиск. Обрабатывать нужно от листьев к корню, а для определения ответа для очередной вершины нужно объединить множества всех ее детей.

Задача G. Дерево в отрезке

Имя входного файла:	`g.in`
Имя выходного файла:	`g.out`
Ограничение по времени:	2 с
Ограничение по памяти:	256 Мб

Задано подвешенное дерево, содержащее n ($1 \le n \le 300\ 000$) вершин. Каждая вершина покрашена в один из n цветов. Требуется для каждой

вершины v вычислить количество вершин с цветами в m ($1 \le m \le 10$) отрезках-запросах l_i, r_i, встречающихся в поддереве с корнем v. Вершина лежит в отрезке, если номер ее цвета c ($l_i \le c \le r_i$).

Формат входного файла

В первой строке входного файла задано число n и m. Последующие n строк описывают вершины, по одной в строке. Описание очередной вершины i имеет вид p_i c_i, где p_i — номер родителя вершины i, а c_i — цвет вершины i ($1 \le c_i \le n$). Для корня дерева $p_i = 0$.

Следующие m строк содержат запросы в формате двух чисел l_i, r_i ($1 \le l_i \le r_i \le n$).

Формат выходного файла

Выведите n строк по m чисел, обозначающих количества цветов в поддеревьях с корнями в вершинах $1, \ldots, n$ в соответствующих отрезках.

Примеры

g.in	g.out
5 2 2 1 3 2 0 3 3 3 2 1 1 5 2 3	1 0 3 1 5 3 1 1 1 0
1 1 0 1 1 1	1
2 1 0 1 1 1 1 1	2 1
2 3 0 1 1 2 1 1 2 2 1 2	1 1 2 0 1 1

Разбор задачи G. Дерево в отрезке

Для решения этой задачи проще всего реализовать объединяющийся отсортированный массив по аналогии с рассказанным на лекции. Тогда при подсчете ответа для очередной вершины нужно будет объединить всех потомков и поискать ответ в объединенном множестве.

Суммарное время на объединения будет потрачено $O(N * \log N)$, а на запросы $O(N * M * \log^2 n)$. Однако операции бинарного поиска очень быстрые, поэтому работает все существенно быстрее сложных деревьев.

Задача H. Строки в дереве

Имя входного файла:	`h.in`
Имя выходного файла:	`h.out`
Ограничение по времени:	2 с
Ограничение по памяти:	256 Мб

Дано дерево. Дерево — это связный граф без циклов. На каждом ребре дерева написана строчная латинская буква. Между каждыми двумя вершинами существует ровно один простой путь, то есть путь по рёбрам дерева, проходящий через каждую вершину не более одного раза. Каждому пути соответствует строка, которая получается, если идти по этому пути и читать буквы на рёбрах в порядке следования. Путь можно проходить, начиная с любого его конца.

Также дана строка S. Соответствует ли она какому-либо простому пути в данном дереве?

Длина строки и размер дерева не превышают $3 \cdot 10^5$.

Формат входного файла

Первая строка содержит заданную строку s. Следующая строка содержит количество вершин в дереве n. Следующие $n - 1$ строк описывают ребра дерева в виде u, v, c, где u и v — вершины дерева, а c — символ, написанный на ребре.

Формат выходного файла

В первой строке выведите `YES`, если такой путь существует, и `NO` в противном случае.

Если путь существует, то выведите пару вершин, путь между которыми образует заданную строку.

Примеры

h.in	h.out
abc 4 1 2 c 4 3 a 2 4 b	YES 1 3
abc 1	NO
zy 6 1 2 x 1 3 y 1 4 z 3 5 a 4 6 b	YES 4 3

Разбор задачи H. Строки в дереве

Путь в дереве всегда имеет вид — подъем вверх по дереву до некоторой вершины v, после чего спуск. Будем предполагать, что самая высокая вершина — это v. Будем рассматривать все вершины. Чтобы определить существует ли путь с самой высокой вершиной v нужно найти все строки, которые идут вниз от этой вершины. Сами строки искать слишком долго, поэтому будем искать hash от этих строк. Если известны множества для детей, то необходимо дописать к множествам соответствующие буквы и объединить. А так же добавить строки из одного символа, идущего непосредственно в детей.

При использовании полиномиальной хэш-функции дописывание буквы эквивалентно применению ко всем элементам линейной функцию $p * x + c$, где p — коэффициент функции, а c — код символа. Мы не можем пройтись по всему множеству, поэтому рядом с множеством будем хранить два коэффициента линейной функции, которые нужно применить ко всему множеству (a и b).

При объединении двух множеств будем перекладывать меньшее множество в большее. При перекладывании добьемся, чтобы во второе множество попала правильная хэш-функция. Если у меньшего множества линейная функция $a_1x + b_1$, а у второго $a_2x + b_2$, то при перекладывании из множества числа z во второе множество добавим $(a_1z + b_1 - b_2) * a_2^{-1}$, чтобы фактически добавленное число совпадало с ожидаемым.

Так же при объединении мы будем следить за ответом. Если мы перекладываем число, которое является суффиксом или префиксом искомой строки (это можно определить храня множество хэш-функций префиксов и суффиксов), то проверим нет ли в большем множестве соответствующего (префикса-суффикса).

При проверке есть ли в множестве конкретная хэш-функция нужно не забывать так же применять обратное преобразование линейной функции $a^{-1}x - b$.

Задача I. Объединение прямоугольников

Имя входного файла:	`i.in`
Имя выходного файла:	`i.out`
Ограничение по времени:	2 с
Ограничение по памяти:	256 Мб

Объединение прямоугольников куда проще, если их стороны параллельны осям координат. Найдите площадь объединения заданного множества таких прямоугольников.

Формат входного файла

В первой строке задано количество прямоугольников N ($2 \leq N \leq 3$). На следующих N строках записаны прямоугольники координатами двух противоположных углов в формате: $x_1\ y_1\ x_2\ y_2$. Координаты разделены пробелами и находятся в отрезке $[-10^9, 10^9]$, а так же являются целыми числами.

Допускаются вырожденные прямоугольники.

Формат выходного файла

Выведите площадь объединения прямоугольников округленную до ближайшего целого.

Примеры

i.in	i.out
2 0 0 10 10 20 5 5 20	300
2 0 0 10 10 20 15 15 20	125
2 0 0 10 10 3 5 5 3	100
3 0 0 10 10 13 5 5 13 12 12 15 15	147

Разбор задачи I. Объединение прямоугольников

Данную задачу проще всего решить по формуле включения-исключения. Для двух прямоугольников ответ равен сумме площадей минус площадь пересечения. Для трех — сумма площадей минус сумма площадей попарных пересечений и плюс пересечение всех трех прямоугольников.

Задача J. Объединяем множество

Имя входного файла:	`j.in`
Имя выходного файла:	`j.out`
Ограничение по времени:	2 с
Ограничение по памяти:	256 Мб

Объединение множеств — это очень сложно. Поэтому найдите количество чисел, которые входят хотя бы в одно из заданных множеств.

Формат входного файла

В первой строке задано количество множеств N ($1 \le N \le 10$). На следующих N строках записаны множество в формате: $k\ a_1 \ldots a_k$ ($1 \le k \le 1000$, $-10^9 \le a_i \le 10^9$). Числа в одной строке разделены пробелами.

Формат выходного файла

Выведите единственно число — ответ на задачу.

Примеры

j.in	j.out
1 1 -606037695	1
2 1 -822866455 1 503976630	2
2 3 1 2 3 3 2 4 3	4

Разбор задачи J. Объединяем множество

Самый простой способ решить эту задачу воспользовавшись стандартными реализациями Set. Однако ограничения небольшие, поэтому можно каждый раз искать элементы множества в массиве за суммарную сложность $O(N^2)$

Задача K. Непрефиксные коды

Имя входного файла:	`k.in`
Имя выходного файла:	`k.out`
Ограничение по времени:	2 с
Ограничение по памяти:	256 Мб

Рассмотрим множество из n символов $\{1, 2, \ldots, n\}$. Пусть каждому из этих символов сопоставлен некоторый вектор b_i из 0 и 1. Тогда каждая строка из исходных символов $c = c_1 c_2 \ldots c_k$ определяет вектор из 0 и 1, получаемый конкатенацией $b_{c1}, b_{c2}, \ldots, b_{ck}$, обозначим его за $B(c)$. Найдите самый короткий вектор X, состоящий из 0 и 1, такой, что $X = B(c)$ и $X = B(d)$ для двух различных упорядоченных наборов c и d. Если таких последовательностей несколько, то выведите наименьшую в лексикографическом порядке. Гарантируется, что хотя бы одна такая последовательность будет существовать.

Формат входного файла

Первая строка входного файла содержит число N ($2 \leq N \leq 20$). Следующие N строк содержат вектора b_i, длиной не более 20.

Формат выходного файла

В выходной файл выведите на первой строке длину найденной последовательности. На второй строке выведите саму последовательность.

Пример

k.in	k.out
5 0110 00 111 001100 110	9 001100110

Разбор задачи K. Непрефиксные коды

В данной задаче необходимо найти две последовательности слов, которые при конкатенации дают одну и туже строку. Пусть мы выбрали некоторое количество строк в первой последовательности и второй. Будем действовать так, чтобы первая последовательность была длиннее второй, однако не опережала ее более, чем на одно слово. Тогда состоянием будет опережение первой последовательности относительно второй. Такие опережения возможно только на некоторый суффикс одной из строк. Переходом между состояниями будет дописывание ко второй последовательности одного из слов, которое не противоречит совпадению.

Если при этом последовательность опередила первую, то поменяем их местами. Весом перехода будет длина дописываемого слова. Таким образом нам необходимо найти кратчайший путь в описанном графе из стартовых позиций. Стартовыми позициями являются опережения образуемые парой начальных слов.

Задача L. Дождик

Имя входного файла:	`l.in`
Имя выходного файла:	`l.out`
Ограничение по времени:	2 с
Ограничение по памяти:	256 Мб

В НИИ метеорологии решили изучить процесс образования водоемов на различных рельефах местности во время дождя. Ввиду сложности реальной задачи была создана двумерная модель, в которой мест-

ность имеет только два измерения — высоту и длину. В этой модели рельеф местности можно представить как N-звенную ломаную с вершинами $(x_0, y_0), \ldots, (x_N, y_N)$, где $x_0 < x_1 < \ldots < x_N$ и $y_i \neq y_j$, для любых $i < j$. Слева в точке x_0 и справа в точке x_N рельеф ограничен вертикальными горами огромной высоты. Если бы рельеф был горизонтальным, то после дождя вся местность покрылась бы слоем воды глубины H. Но поскольку рельеф — это ломаная, то вода стекает и скапливается в углублениях, образуя водоемы. Требуется найти максимальную глубину в образовавшихся после дождя водоемах.

Формат входного файла

В первой строке задано натуральное число N ($1 \leq N \leq 100$) и H — действительное число, заданное с тремя цифрами после десятичной точки ($0 \leq H \leq 10^9$). В последующих $N + 1$ строках — по два целых числа x_i, y_i ($-10000 \leq xi, yi \leq 10000$, $0 \leq i \leq N$).

Числа в строках разделены пробелами.

Формат выходного файла

Выведите единственное число — искомую глубину с точностью до 4-х знаков после десятичной точки.

Пример

l.in	l.out
7 7.000 -5 10 -3 4 -1 6 1 -4 4 17 5 3 9 5 12 15	15.844619

Разбор задачи L. Дождик

Данная задача решается моделированием процесса подъема воды. Область ломанной в каждый момент времени разбивается на отрезки. В каждом отрезке вода поднимается с некоторой фиксированной скоростью, пропорциональной длине этого отрезка и возможно длине нескольких соседних отрезков в каждую сторону, с которых вода сливается.

Изначально такими отрезками являются $[x_i, x_{i+1}]$. В последствии некоторые из отрезков могут быть объединены, когда уровень воды поглатит границу между ними. В фиксированный момент времени необходимо определить скорость подъема воды на каждом отрезке (площадь в единицу времени). Тут нужно учесть, что вода может стекать со многих отрезков слева и справа. После чего необходимо вычислить отрезок, который быстрее других заполнится, то есть дойдет до границы по y с каким-либо другим.

После этого необходимо смоделировать подъем воды и объединить отрезки. На последующем шаге нужно проконтролировать, что объем воды ограничен.

Задача на самый короткий код

Имя входного файла:	`square.in`
Имя выходного файла:	`square.out`
Ограничение по времени:	5 с
Ограничение по памяти:	256 Мб

Современные программисты уже почти разучились работать с длинными числами самостоятельно. Во многих языках уже есть встроенная работа с длинными числами. Найдите квадрат числа, состоящего из N единиц.

Формат входного файла

В первой строке задано единственное число N ($1 \le N \le 50000000$).

Формат выходного файла

Выведите квадрат числа, состоящего из N единиц, без лидирующих нулей.

Примеры

square.in	square.out
2	121
3	12321
4	1234321
5	123454321

День шестой (23.02.2012 г.)
Контест Неспирного Виталия и Лунева Антона

Задачи и разборы

Задача A. Minimal Prime Divisor

Имя входного файла:	`a.in`
Имя выходного файла:	`a.out`
Ограничение по времени:	0.5 s
Ограничение по памяти:	256 Mb

Let $\mathrm{minp}(n)$ denotes the minimal prime divisor of integer $n > 1$. Denote further $f(n) = \min\{\mathrm{minp}(1 + n + n^2 + n^3 + n^4), K\}$, where K is the given positive integer. For given L and R you need to find the value of the following expression

$$\left(\sum_{n=L}^{R} n \cdot f(n)\right) \pmod{10^9 + 7}.$$

Формат входного файла

The only line of input file contains three space separated positive integers K, L and R. Here $1 \le K \le 10^7$, $1 \le L \le R \le 10^{18}$ and $R - L \le 2000000$.

Формат выходного файла

The only line of output file should contain the answer to the problem.

Примеры

a.in	a.out
20 1 6	207
6 13 42	4779

Разбор задачи A. Минимальный простой делитель

В задаче необходимо для всех чисел n от L до R найти минимальный простой делитель числа $1 + n + n^2 + n^3 + n^4$, если он не больше K.

Сначала найдём, какие простые числа могут быть делителями чисел вида $1 + n + n^2 + n^3 + n^4$. Ясно, что $p = 5$ делит некоторые такие числа. А именно, те и только те, для которых $n \equiv 1 \pmod 5$. Пусть теперь $p \neq 5$.

Чтобы идти дальше, введём одно важнейшее понятие из теории чисел.

Определение. Пусть m – натуральное число a взаимно простое с m. Наименьшее натуральное k, для которого $a^k \equiv 1 \pmod m$, называется порядком a по модулю m и обозначается $r_m(x)$.

Одним из важнейших свойств порядка является следующее утверждение.

Лемма. Если $a^k \equiv 1 \pmod m$, то k делится на $r_m(a)$.

Вернемся к нашей ситуации. Ясно, что $n \not\equiv 1 \pmod p$ (иначе p кратно 5). Ясно также, что $n^5 \equiv 1 \pmod p$. Но тогда по лемме порядок числа n по модулю p равен 5. Так как $x^{p-1} \equiv 1 \pmod p$ по малой теореме Ферма, то опять по лемме $p-1$ делится на 5. А, значит, $p-1$ делится на 10. Итак, мы установили, что только 5 и простые числа вида $10k+1$ могут быть делителями чисел вида $1+n+n^2+n^3+n^4$.

Убедимся, что любое простое $p = 10k+1$ делит одно из чисел вида $1+n+n^2+n^3+n^4$ и научимся быстро описывать все такие n. Другими словами, нам надо решить сравнение

$$1+n+n^2+n^3+n^4 \equiv 0 \pmod p.$$

Ясно, что это сравнение равносильно

$$n^5 \equiv 1 \pmod p, \quad n \not\equiv 1 \pmod p.$$

Заметим, что в силу малой теоремы Ферма для любого x, не кратного p, число $n = x^{(p-1)/5} = x^{2k}$ будет решением сравнения $n^5 \equiv 1 \pmod p$. Если при этом $n-1$ не делится на p, то n является решением исходного сравнения $1+n+n^2+n^3+n^4 \equiv 0 \pmod p$. По теореме Лагранжа (см. статью "Lagrange's theorem (number theory)" в Википедии) сравнение $x^{2k} \equiv 1 \pmod p$ имеет не больше $2k$ решений. На самом деле оно имеет ровно $2k$ решений, но в данной задаче это не важно. Главное, что из $p-1$ ненулевых остатков по модулю p для 80%, то есть $4(p-1)/5 = 8k$ штук, это сравнение не выполнено. Поэтому каждый из них дает решение сравнения $1+n+n^2+n^3+n^4 \equiv 0 \pmod p$ способом, описанным выше. Таким образом, мы можем либо перебирать случайные числа от 1 до $p-1$, либо идти подряд от 1, и быть уверенными, что искомое n найдется быстро. Но это только одно n, а могут быть и другие. Оказывается теперь их легко описать. Если n – одно из решений данного сравнения, то множество всех решений по модулю p имеет вид $\{n, n^2, n^3, n^4\}$. Легко проверить, что все эти числа являются решениями сравнения $n^5 \equiv 1 \pmod p$. Но так как $r_p(n) = 5$, то легко убедиться что все эти числа различны по модулю p и не равны 1. Значит, это 4 различных решения данного сравнения. Но по теореме Лагранжа это сравнение имеет не более четырех решений. Значит, это действительно все решения и других нет.

Теперь для данного p мы легко можем описать все n для которых $1 + n + n^2 + n^3 + n^4$ делится на p.

Решать исходную задачу будем аналогом решета Эратосфена. так как $K <= 10^7$, то можно найти все простые числа вида $10k + 1$ с помощью обычного решета, а затем в специальном массиве f$[n]$ размера $R - L + 1$ заполнить f$[n - L] = \text{minp}(1 + n + n^2 + n^3 + n^4)$ для $L \leq n \leq R$. Изначально все элементы массива установлены в K. На каждом шаге мы берем очередное простое число p, находим 4 арифметические прогресси тех n, для которых $1+n+n^2+n^3+n^4$ делится на n и присваиваем f$[n-L] := p$, если f$[n - L]$ было равно K.

Теперь обсудим некоторые приемы, которые позволяют существенно ускорить программу.

1. Чтобы найти все простые вида $10k + 1$, которые не превосходят K, найдем сначала все простые, не превосходящие $\sqrt{K}$, а затем будем делать решето Эратосфена, но только по числам вида $10k + 1$. Для этого нужно находить первое число вида $10k+1$ кратное p. Это можно сделать в простом цикле.

2. Выгоднее всего искать первое n для данного $p = 10k + 1$, проходя по первым простым x. Общее число возведений в степень по модулю, которые надо сделать в худшем случае, будет около $200,000$. Но главное, что можно при правильной реализации алгоритма быстрого возведения в степень экономить bit_cnt(2k) операций умножения и взятия остатка в long long.

Задача B. Isosceles Triangulation

Имя входного файла:	`b.in`
Имя выходного файла:	`b.out`
Ограничение по времени:	0.5 s
Ограничение по памяти:	256 Mb

Consider regular polygon with $N \geq 3$ sides. Let's cut it into $N-2$ triangles by $N - 3$ non-intersecting diagonals. Is it possible that all these triangles are isosceles? If yes then we say that regular N-gon admits an isosceles triangulation. Write the program that answers this question.

Формат входного файла

The first line contains a positive integer $T \leq 1000$, the number of test cases. Each of the next T lines contains a positive integer N, the number of sides of regular polygon. It is guaranteed that $3 \leq N \leq 10^9$.

Формат выходного файла

For each value of N in the input file output "`YES`" (without quotes) if regular N-gon admits an isosceles triangulation and "`NO`" otherwise.

Примеры

b.in	b.out
5 3 4 5 6 7	YES YES YES YES NO

Разбор задачи B. Равнобедренная триангуляция

Ответ в задаче простой. Правильный N-угольник допускает равнобедренную триангуляцию тогда и только тогда, когда сумма цифр N в двоичной записи не превосходит 2. Приведем более-менее строгое доказательство этого факта.

Пусть правильный N-угольник допускает равнобедренную триангуляцию (будем называть такое N хорошим). Любая его сторона является стороной одной из треугольников разбиения. Если она является боковой стороной своего треугольника, то ее треугольник образован ей и одной из соседних сторон. Если же она является основанием, то третья вершина ее треугольника лежит на серединном перпендикуляре к стороне. Это возможно только при нечетном N. Значит, при четном N любая равнобедренная триангуляция обязана содержать $N/2$ треугольников, образованных парами соседних сторон. Поэтому четное N хорошее тогда и только тогда, когда $N/2$ хорошее.

Осталось разобраться с нечетным N. В этом случае стороны нельзя разбить на пары, поэтому одна из сторон будет основанием своего треугольника. По замечанию выше этот треугольник разобьет наш N-угольник на два равных. В каждом из них есть сторона, которая больше всех диагоналей и сторон в нем. Поэтому она должна быть основанием. Значит, $(N-1)/2$ должно быть четным, а вновь построенные треугольники разбивают наши два многоугольника на 4, для которых выполнены те же свойства. Поэтому продолжая так дальше мы получим, что N должно иметь вид 2^k+1. А это (с учетом возможности умножения на любую степень двойки) те и только те числа, в которых ровно два бита равны 1.

Задача C. XOR pairs

Имя входного файла:	`c.in`
Имя выходного файла:	`c.out`
Ограничение по времени:	0.5 s
Ограничение по памяти:	256 Mb

For given non-negative integers X, A, B consider all pairs of non-negative integers (a, b) such that a xor $b = X$, $a \geq A$ and $b \geq B$. Here a xor b is the xor-operation ("xor" in Pascal, "^" in C/C++/Java). Let's sort these pairs by a in increasing order and pairs with equal value of a by b in increasing order. You need to find N-th pair among these pairs. Numeration starts from 1. But be ready to answer for thousands of such question for given X, A, B.

Формат входного файла

The first line contains four integers X, A, B and T, the number of test cases. Each of the next T lines contains a positive integer N. It is guaranteed that $0 \leq X, A, B \leq 10^{18}$, $1 \leq T \leq 10000$ and $1 \leq N \leq 10^{18}$.

Формат выходного файла

For each value of N in the input file output two space separated integers a and b, such that pair (a, b) is N-th pair among described sequence of pairs.

Примеры

c.in	c.out
6 1 4 10 1 2 3 4 5 6 7 8 9 10	1 7 2 4 3 5 8 14 9 15 10 12 11 13 12 10 13 11 14 8

Разбор задачи C. XOR-пары

Ясно, что все такие пары имеют вид $(a, a \text{ xor } X)$, где $a \geq A$ и a xor $X \geq B$. Опишем те $a \geq 0$, для которых a xor $X < B$. Множество таких a совпадает

с $S = \{b \text{ xor } X : 0 \geq b < B\}$. Опишем S. Запишем B в двоичной записи

$$B = 2^{r_h} + 2^{r_{h-1}} + \ldots + 2^{r_1} + 2^{r_0}, \quad r_h > r_{h-1} > \ldots > r_1 > r_0 \geq 0.$$

Рассмотрим числа в промежутке $[L_j, R_j)$, где $R_j = 2^{r_h} + 2^{r_{h-1}} + \ldots + 2^{r_j}$, а $L_j = R_j - 2^{r_j}$. Здесь $j \in \{0, 1, \ldots, h\}$. Тогда легко проверить, что $S_j = \{b \text{ xor } X : L_j \geq b < R_j\}$ совпадает с промежутком $[X_j, X_j + 2^{r_j})$, где $X_j = ([X/2^{r_j}] \cdot 2^{r^j}) \text{ xor } L_j$. Действительно, промежуток $[L_j, R_j)$ это все числа, у которых r_j младших битов принимают произвольные значения, а старшие биты фиксированы и совпадают с соответствующими битами L_j. Поэтому у b xor X при $L_j \leq b < R_j$ старшие биты, начиная с r_j-ого, совпадают с соответствующими битами L_j xor X, а младшие также пробегают все возможные значения (последнее утверждение — несложное упражнение). Таким образом, S это объединение $h + 1$ промежутка $[X_j, X_j + 2^{r_j})$, так как промежутки $[L_j, R_j), j \in \{0, 1, \ldots, h\}$ покрывают промежуток $[0, B)$ и попарно не пересекаются. Тогда множество тех a, для которых $a \geq A$ i a xor $X \geq B$ это тоже объединение малого ($\leq h + 1$) числа отрезков.

Таким образом, искомое множество значений a может быть найдено за $O(\log B * \log \log B)$ (не забываем о сортировке :)).

Теперь на запрос можно отвечать за один проход по этому множеству отрезков, если поддерживать сумму длин пройденных уже отрезков и сравнивать ее с N из запроса. Можно также найти эти суммы заранее и отвечать на каждый запрос за $O(\log \log B)$ двоичным поиском. Но это было бы критично, если бы ограничения на X, A, B, N были порядка 2^{500}.

Задача D. Win of the looser

Имя входного файла:	`d.in`
Имя выходного файла:	`d.out`
Ограничение по времени:	1 s
Ограничение по памяти:	256 Mb

Two players staged a darts tournament. Tournament is a series consisting of N games. In each game participants make some number of shots, gaining points.For one game each player can get any number of points from 0 to K. Of course a game is won by one of two players who gains more points in it than the opponent. If players gain equal points in the game, then its outcome is a draw.

But determining the winner of the tournament is not so straightforward. By some rules the winner is the one who's won more games in the series. On the other the one who's gained more points over all games.

You are required to determine the number of different variants of the tournament's scenario, where the loser of one version is the winner of the other. Two variants of the tournament's scenario are considered as different if at least one of the participants in any game has gained a different number of points.

Формат входного файла

In a single line you are given 3 numbers N, K and p.

Ограничения

$1 \le N \le 50$, $0 \le K \le 100$, $1 \le p \le 10^9$.

Формат выходного файла

In a single line output the number of possible scenario of the tournament modulo p.

Примеры

d.in	d.out
3 3 100	54
2 4 200	0

Разбор задачи D. Задача Победа проигравшего

Пусть $F(i, dw, dp)$ – количество вариантов исхода первых i игр ($0 \le i \le N$), после которых разность между количеством одержаных игроками побед равна dw ($-i \le dw \le i$), а разность набранных очков – dp ($-iK \le dw \le iK$). Тогда ответ на задачу выражается формулой

$$\sum_{dw<0,dp>0} F(N, dw, dp) + \sum_{dw>0,dp<0} F(N, dw, dp).$$

Для получения рекуррентной формулы рассмотрим все варианты счета $k_1 : k_2$, с которым могла закончиться игра с номером i. Тогда для того, чтобы после этой игры, мы оказались в состоянии (i, dw, dp), до нее мы должны были находиться в состоянии $(i-1, dw-\mathrm{sign}(k_1-k_2), dp-(k_1-k_2))$. Отсюда следует формула

$$F(i, dw, dp) = \sum_{0 \le k_1, k_2 \le K} F(i-1, dw - \mathrm{sign}(k_1 - k_2), dp - (k_1 - k_2)).$$

Следует отметить, что если величины (i, dw, dp) не удовлетворяют ограничениям, определяющими область определения функции, то мы считаем F

в таких состояниях равным 0. В качестве начального значения может быть выбрано $F(0,0,0)=1$. Все другие состояния при $i=0$ лежат вне области определения.

Общее количество состояний у нас имеет порядок $O(N^3K)$, а предложенная формула позволяет вычислять функцию в любом из состояний (при условии, что уже просчитаны необходимые предыдущие) за $O(K^2)$. Таким образом весь алгоритм имел бы сложность $O(N^3K^3)$, что довольно много при заданных ограничениях.

Заметим, что на самом деле предыдущие состояния определяются лишь разностью k_1-k_2. Обозначим ее s. Остается лишь узнать количество пар k_1, k_2, которые дают определенную разность. Нетрудно увидеть, что таких пар будет $K+1-|s|$. Тогда мы можем записать рекуррентную формулу сложности $O(K)$:

$$F(i,dw,dp)=(K+1)F(i-1,dw,dp)+\sum_{0<s\le K}(K+1-s)F(i-1,dw-1,dp-s)+$$

$$+\sum_{0<s\le K}(K+1-s)F(i-1,dw+1,dp+s).$$

Но возможно еще одно улучшение, основанное на идеологии конечных разностей, которое позволит вычислять очередное значение за $O(1)$. Для этого рассмотрим, насколько будет отличаться $\sum\limits_{0<s\le K}(K+1-s)F(i-1,dw+1,dp+s)$ от такого же значения для $dp-1$. Простые вычисления показывают, что

$$\sum_{0<s\le K}(K+1-s)F(i-1,dw+1,dp+s)-\sum_{0<s\le K}(K+1-s)F(i-1,dw+1,dp-1+s)=$$

$$=-KF(i-1,dw+1,dp)+\sum_{0<s\le K}F(i-1,dw+1,dp+s)=$$

$$=-KF(i-1,dw+1,dp)+S(i-1,dw+1,dp+K)-S(i-1,dw+1,dp).$$

Здесь

$$S(i,dw,dp)=\sum_{r\le dp}F(i,dw,r).$$

Аналогично можно записать рекуррентность и для другой суммы:

$$\sum_{0<s\le K}(K+1-s)F(i-1,dw-1,dp-s)-\sum_{0<s\le K}(K+1-s)F(i-1,dw-1,dp-1-s)=$$

$$=KF(i-1,dw-1,dp-1)-\sum_{1<s\le K+1}F(i-1,dw-1,dp-s)=$$

$= KF(i-1, dw-1, dp-1) - S(i-1, dw-1, dp-2) + S(i-1, dw-1, dp-K-2)$.

Эти формулы позволят пересчитывать суммы за $O(1)$ правда за счет сохранения еще и величин S, каждая из которых очевидно также вычисляется за $O(1)$:

$$S(i, dw, -iK-1) = 0, S(i, dw, dp) = S(i, dw, dp-1) + F(i, dw, dp).$$

Тем самым мы приходим к алгоритму, который будет работать за время $O(N^3K)$ и требует памяти порядка $O(N^2K)$ (поскольку для перехода к слою i нам нужен лишь слой $i-1$, остальные мы можем уже не хранить).

Возможны еще некоторые неасимптотические улучшения. Во-первых, заметим, что $F(i, dw, dp) = F(i, -dw, -dp)$. Это сократить вычисления в 2 раза, сохраняя лишь те элементы $F(i, dw, dp)$, для которых $dw \geq 0$. А во-вторых, можем уточнить границы диапазона dp, учитывая значение dw. Если общее количество сыгранных уже игр равно i, а разность между количеством одержанных игроками побед dw, то первый игрок мог победить не более, чем в $w_1 = \lfloor (i+dw)/2 \rfloor$ играх, а проиграть – не более, чем в $w_2 = \lfloor (i-dw)/2 \rfloor$ играх. Разность между количеством очков тогда не может превзойти $Kw_1 - w_2$ (во всех выигранных встречах мы выиграем с максимальным счетом, а во всех проигранных – проиграем с минимальным счетом; добавлять ничьи невыгодно, поскольку их нужно добавлять парами, забирая не только поражения, но и победы, чтобы сохранить баланс dw). Аналогично можно найти и ограничение снизу. Тогда имеем $-Kw_2 + w_1 \leq p \leq Kw_1 - w_2$. Указанные два улучшения сохраняют ту же самую асимптотику $O(N^3K)$ для вычислительной сложности, но уменьшают константу.

Задача E. Longest Geometric Progression

Имя входного файла:	`e.in`
Имя выходного файла:	`e.out`
Ограничение по времени:	1 sec
Ограничение по памяти:	256 Мб

The sequence $\{a_0, a_1, \ldots, a_n\}$ is called *geometric progression* if $a_i^2 = a_{i-1}a_{i+1}$ for all $i \in \{1, 2, \ldots, n-1\}$. Note that the sequence composed of one or two numbers is always a geometric progression. You are given positive integers L and R such that $L < R$. Find the longest geometric progression $\{a_0, a_1, \ldots, a_n\}$ composed of positive integers such that

$$L \leq a_0 < a_1 < \ldots < a_n \leq R.$$

Формат входного файла

The first line contains a positive integer $T \leq 10000$, the number of test cases. Each of the next T lines contains two space separated positive integer L and R. It is guaranteed that $1 \leq L < R \leq 10^{18}$.

Формат выходного файла

For each pair (L, R) from the input file output the maximal possible n such that there exists a sequence $\{a_0, a_1, \ldots, a_n\}$ composed of positive integers such that $a_i^2 = a_{i-1}a_{i+1}$ for all $i \in \{1, 2, \ldots, n-1\}$ and $L \leq a_0 < a_1 < \ldots < a_n \leq R$, followed by numbers a_0 and a_1. (Note that $L < R$. Hence the sequence $\{L, R\}$ always satisfies these conditions. So n will be always positive). If there several such sequences for given n you can output any.

Примеры

e.in	e.out
7 1 4 4 9 10 20 100 101 100 1000 999 1333 1 1000000	2 1 2 2 4 6 1 10 11 1 100 101 5 128 192 3 1000 1100 19 1 2

Разбор задачи E. Самая длинная геометрическая прогрессия

Пусть $\{a_0, a_1, \ldots, a_k\}$ – некоторая строго возрастающая прогрессия из чисел от L до R. Число $b = a_1/a_0$ называется знаменателем прогрессии. Запишем b в виде несократимой дроби $b = p/q$. Здесь $p > q \geq 1$. Ясно, что $a_j = a_0 \cdot b^j = a_0/q^j \cdot p^j$. Так как p и q взаимно просты, то a_0 должно делиться на все степени q вплоть до k-ой. Значит, $a_0 = a \cdot q^k$. Поэтому наша прогрессия примет вид

$$a \cdot q^k, a \cdot q^{k-1} \cdot p, \ldots, a \cdot p^k.$$

Если мы заменим p на $q + 1$, то только выиграем (последовательность будет удовлетворять ограничениям задачи и иметь ту же длину, но при этом может быть продолжена).

Таким образом, существование прогрессии длины k равносильно существованию натуральных чисел a и q таких, что

$$L \le a \cdot q^k, a \cdot (q+1)^k \le R. \tag{1}$$

Разрешим неравенство (1) относительно a:

$$\frac{L}{q^k} \le a \le \frac{R}{(q+1)^k}. \tag{2}$$

Существование a, удовлетворяющего этому неравенству, равносильно такому неравенству:

$$\left\lceil \frac{L}{q^k} \right\rceil \le \left\lfloor \frac{R}{(q+1)^k} \right\rfloor. \tag{3}$$

Далее $a_k = a(q+1)^k \le R$. Поэтому $(q+1)^k \le R$ или $q \le \sqrt[k]{R} - 1$. Поэтому, если k не слишком маленькое (скажем больше 6 или 7), мы можем просто перебрать все q до $\lfloor \sqrt[k]{R} \rfloor - 1$ и проверить неравенство (3). Но что делать при маленьких k? Если пытаться перебирать a, то не ясно до каких пределов. Это основная хитрость задачи. Оказывается, если (a, q) удовлетворяет неравенству (1) и $a \ge 2^k$, то найдется такое Q, что при $a = 1$ и $q = Q$ это неравенство снова будет выполнено. Действительно, это Q должно удовлетворять неравенствам

$$a \cdot q^k \le Q^k, (Q+1)^k \le a \cdot (q+1)^k$$

или

$$\sqrt[k]{a} \cdot q \le Q \le \sqrt[k]{a} \cdot (q+1) - 1$$

или

$$\sqrt[k]{a} \cdot q \le Q \le \sqrt[k]{a} \cdot q + \sqrt[k]{a} - 1.$$

Но $\sqrt[k]{a} - 1 \ge 1$. Поэтому на отрезке $[\sqrt[k]{a}, \sqrt[k]{a} \cdot q + \sqrt[k]{a} - 1]$ действительно найдется натуральное число.

Теперь разрешим неравенства (1) относительно q:

$$\sqrt[k]{\frac{L}{a}} \le q \le \sqrt[k]{\frac{R}{a}} - 1. \tag{4}$$

Зная такую границу a, мы можем легко перебирать a до $2^k - 1$ при маленьких k, и если ни при одном a неравенство (4) не выполнено, мы можем быть уверены, что прогрессии длины k не существует.

Теперь если искать k двоичным поиском и правильно выбирать по чем перебирать для данного k (по q или по a), то можно получить AC, если предподсчитать все степени $q^k \le 10^{18}$ такие, что $k \ge 4$ и $q > 1$.

Но можно ускорить решение в 6 раз, если заметить такой момент. Из неравенства (2) следует, что

$$\frac{L}{q^k} \le \frac{R}{(q+1)^k}$$

или

$$1 + 1/q \le \sqrt[k]{R/L}$$

или

$$q \ge \left\lceil \frac{1}{\sqrt[k]{R/L} - 1} \right\rceil .$$

Значит, нам незачем рассматривать q, меньшие этого значения!

Обозначим $maxa = 2^k - 1$, $maxq = \lfloor \sqrt[k]{R} \rfloor - 1$ и $minq = \left\lceil \frac{1}{\sqrt[k]{R/L}-1} \right\rceil$. Если для данного k выполнено $maxa < maxq - minq$, то будем перебирать a от 1 до $maxa$, иначе перебираем q от $minq$ до $maxq$.

Но с вычислением $minq$ надо быть очень осторожным. Если отношение R/L очень близко к единице, то в знаменателе произойдет вычитание очень близких чисел, что, как известно, является главным источником большой погрешности. Если вычисленное значение больше настоящего, то вполне можно пропустить q, для которого есть ответ. Поэтому лучше занизить $minq$, например в 0.9999 раз, чтобы точно не упустить подходящего q.

День седьмой (24.02.2012 г.) Контест Куликова Егора Юрьевича

Об авторе...

Куликов Егор Юрьевич, старший инженер–программист в компании "Эксперт-система". Закончил "Московский государственный университет"

Основные достижения:

- Победитель Google CodeJam 2010;

  В составе команды МГУ x13 победитель Чемпионата России 2006;

- На январь 2012 года — 8е место в рейтинге TopCoder и 3е в рейтинге Codeforces.

Теоретический материал. Дерево отрезков

Постановка задачи

Пусть есть некоторый массив данных и операции, часть которых работает с отрезками этого массива. Мы хотим довольно быстро (быстрее, чем за размер отрезка) выполнять эти операции. Очень часто нам поможет структура под названием дерево отрезков. Рассмотрим эту структуру на простейшем примере: у нас есть массив и две операции — прибавление к данному элементу заданного числа и запрос суммы элементов на отрезке.

Изменение одного элемента, запрос на отрезке

Простейшее решение: изменяем элементы просто прибавлением, а возвращаем ответ на запрос, суммируя элементы на отрезке итерацией. Оно требует $O(1)$ подготовки, $O(n)$ на запрос и $O(1)$ на изменение (здесь и далее n — длина массива). Покажем, как организовать работу за $O(n)$ подготовки, $O(\log n)$ на запрос и изменение. Определим дерево отрезков следующим образом:

Дерево отрезков — рекурсивная структура над массивом. Для массива единичной длины состоит из одного элемента, соответствующего всему

массиву. Для массива большей длины состоит из корня, соответствующего всему массиву, и двух сыновей этого корня — поддерева, соответствующего подмассиву из первой половины элементов, и поддерева, соответствующего подмассиву из остальных элементов массива.

Что нам это дает? Давайте хранить в каждом узле дерева сумму элементов подмассива, ему соответствующего. Тогда для поддержания этой структуры при изменениях достаточно спускаться по дереву вниз вплоть до вершины, которая соответствует подмассиву из одного изменяемого элемента, и в каждом узле добавлять к сумме значение изменения. Чтобы ответить на запрос, будем также спускаться вниз. При этом, если отрезок запроса не пересекается с отрезком узла, возвращаем 0. Если отрезок узла содержится в вершине запроса, возвращаем сумму, хранимую в данном узле. Если же они пересекаются, то возвращаем сумму результатов вызова от детей. Заметим, что в ответ мы посчитаем сумму на отрезках, которые между собой не пересекаются, а в объединении дают ровно отрезок запроса. Кроме того, на каждом уровне рекурсивный вызов мы будет делать не более, чем из двух узлов, так как один из концов отрезка запроса должен лежать строго внутри отрезка узла. Так как уровней $O(\log n)$, то мы получили требуемую асимптотику и для изменения, и для запроса.

Реализовать данную структуру легче всего на массиве. Листинг реализации приведен ниже. Заметим, что если мы не будем хранить границы отрезка отдельно, а будем вычислять их каждый раз, то это приведет к значительному ускорению — мы будем делать очень “дешевые” вычисления вместо “дорогой” работы с памятью.

Листинг 1: MassQuerySegmentTree

```
1 class MassQuerySegmentTree {
2
3     int length;
4     int[] sum;
5
6     MassQuerySegmentTree(int[] array) {
7         length = array.length;
8         int nodeCount = Integer.highestOneBit(length) << 1;
9         sum = new int[nodeCount];
10        init(0, 0, length - 1, array);
11    }
12
13    void init(int root, int left, int right, int[] array) {
14        if (left == right) {
15            sum[root] = array[left];
16        } else {
17            init(2 * root + 1, left, (left + right) >> 1, array);
18            init(2 * root + 2, ((left + right) >> 1) + 1, right, array);
19            sum[root] = sum[2 * root + 1] + sum[2 * root + 2];
20        }
```

```
21      }
22
23      void add(int position, int value) {
24          add(0, position, value, 0, length - 1);
25      }
26
27      private void add(int root, int position, int value, int left, int right)
28      {
29          if (position > right || position < left) {
30              //Элемент лежит вне отрезка вершины
31              return;
32          }
33          sum[root] += value;
34          if (left != right) {
35              add(2 * root + 1, position, value, left, (left + right) >> 1);
36              add(2 * root + 2, position, value, ((left + right) >> 1) + 1,
37              right);
38          }
39      }
40
41      int get(int from, int to) {
42          return get(0, from, to, 0, length - 1);
43      }
44
45      private int get(int root, int from, int to, int left, int right) {
46          if (from > right || to < left) {
47              //Отрезки запроса и вершины не пересекаются
48              return 0;
49          }
50          if (from <= left && right <= to) {
51              //Отрезок вершины лежит внутри отрезка запроса
52              return sum[root];
53          }
54          return get(2 * root + 1, from, to, left, (left + right) >> 1)
55                  + get(2 * root + 2, from, to, ((left + right) >> 1) + 1,
56                 right);
57      }
58  }
```

Изменение на отрезке, запрос конкретного элемента

В этой задаче мы хотим прибавлять данное число ко всем элементам на отрезке и отвечать на вопрос, что на данный момент находится в заданной позиции Чтобы добиться тех же асимптотических оценок, изменим нашу структуру следующем образом: будем хранить в каждом узле дерева сумму всех изменений, отрезки которых содержат отрезок данного узла, но не содержат отрезок его родителя. Тогда запрос будет выглядеть аналогично изменению в предыдущем случае, а изменение будет выглядеть аналогично запросу.

Листинг 2: MassChangeSegmentTree

```
1 class MassChangeSegmentTree {
2
3     int length;
4     int[] delta;
5
6     MassChangeSegmentTree(int[] array) {
7         length = array.length;
8         int nodeCount = Integer.highestOneBit(length) << 2;
9         delta = new int[nodeCount];
10        init(0, 0, length - 1, array);
11    }
12
13    void init(int root, int left, int right, int[] array) {
14        if (left == right) {
15            delta[root] = array[left];
16        } else {
17            init(2 * root + 1, left, (left + right) >> 1, array);
18            init(2 * root + 2, ((left + right) >> 1) + 1, right, array);
19        }
20    }
21
22    int get(int position) {
23        return get(0, position, 0, 0, length - 1);
24    }
25
26    private int get(int root, int position, int totalDelta, int left,
27    int right)
28    {
29        if (position > right || position < left) {
30            //Элемент лежит вне отрезка вершины
31            return 0;
32        }
33        totalDelta += delta[root];
34        if (left != right) {
35            return get(2 * root + 1, position, totalDelta, left, (left +
36                   right) >> 1) + get(2 * root + 2, position, totalDelta,
37                   ((left + right) >> 1) + 1, right);
38        } else {
39            return totalDelta;
40        }
41    }
42
43    void add(int from, int to, int change) {
44        add(0, from, to, change, 0, length - 1);
45    }
46
47    private void add(int root, int from, int to, int change, int left,
48    int right)
49    {
50        if (from > right || to < left) {
51            //Отрезки запроса и вершины не пересекаются
52            return;
53        }
```

```
54          if (from <= left && right <= to) {
55              //Отрезок вершины лежит внутри отрезка запроса
56              delta[root] += change;
57              return;
58          }
59          add(2 * root + 1, from, to, change, left, (left + right) >> 1);
60          add(2 * root + 2, from, to, change, ((left + right) >> 1)+ 1,right);
61      }
62  }
```

Изменение на отрезке, запрос на отрезке

А теперь объединим использованные техники для того, чтобы обрабатывать изменения на отрезке и отвечать, опять же, сумму элементов на отрезке. В каждом узле будем хранить 2 числа. Во-первых, сумму всех элементов на соответствующем ему подмассиве без учета изменений которые содержат отрезок его родителя. Во-вторых, сумму всех изменений, отрезки которых содержат отрезок данного узла, но не содержат отрезок его родителя.

Листинг 3: SegmentTree

```
 1 class SegmentTree {
 2
 3     int length;
 4     int[] sum;
 5     int[] delta;
 6
 7
 8     SegmentTree(int[] array) {
 9         length = array.length;
10         int nodeCount = Integer.highestOneBit(length) << 1;
11         sum = new int[nodeCount];
12         delta = new int[nodeCount];
13         init(0, 0, length - 1, array);
14     }
15
16
17     void init(int root, int left, int right, int[] array) {
18         if (left == right) {
19             sum[root] = array[left];
20         } else {
21             init(2 * root + 1, left, (left + right) >> 1, array);
22             init(2 * root + 2, ((left + right) >> 1) + 1, right, array);
23             sum[root] = sum[2 * root + 1] + sum[2 * root + 2];
24         }
25     }
26
27
28     int get(int from, int to) {
29         return get(0, from, to, 0, 0, length - 1);
30     }
31
```

```
32      private int get(int root, int from, int to, int totalDelta, int left,
33      int right)
34      {
35          if (from > right || to < left) {
36              //Отрезки запроса и вершины не пересекаются
37              return 0;
38          }
39          if (from <= left && right <= to) {
40              //Отрезок вершины лежит внутри отрезка запроса
41              return sum[root] + totalDelta * (right - left + 1);
42          }
43          totalDelta += delta[root];
44          return get(2 * root + 1, from, to, totalDelta, left,
45                 (left + right) >> 1)+ get(2 * root + 2, from, to,
46                 totalDelta, ((left + right) >> 1) + 1, right);
47      }
48
49      void add(int from, int to, int change) {
50          add(0, from, to, change, 0, length - 1);
51      }
52
53      private void add(int root, int from, int to, int change, int left,
54      int right)
55      {
56          if (from > right || to < left) {
57              //Отрезки запроса и вершины не пересекаются
58              return;
59          }
60          if (from <= left && right <= to) {
61              //Отрезок вершины лежит внутри отрезка запроса
62              delta[root] += change;
63              sum[root] += change * (right - left + 1);
64              return;
65          }
66          add(2 * root + 1, from, to, change, left, (left + right) >> 1);
67          add(2 * root + 2, from, to, change, ((left + right) >> 1) + 1,
68          right);
69          sum[root] = sum[2 * root + 1] + sum[2 * root + 2] + delta[root] *
70          (right - left + 1);
71      }
72  }
```

Бинарный поиск

Пусть теперь у нас в дереве в любой момент времени находятся только положительные числа и нам надо найти первую такую позицию, что сумма всех элементов до нее превосходит заданное число. Используя текущую структуру, мы можем сделать это за $O(\log^2 n)$. Действительно, двоичным поиском будем искать позицию, а затем делать запросы к дереву. Но несложным изменением можно добиться того, чтобы на такой запрос мы отвечали за $O(\log n)$. Действительно, если сумма элементов в левом поддереве не меньше, чем заданное число, то и искомая позиция находит-

ся в левом поддереве, а иначе вычтем эту сумму и будем искать ответ в правом поддереве.

Листинг 4: BinarySearchSegmentTree

```
1 class BinarySearchSegmentTree extends SegmentTree {
2
3     BinarySearchSegmentTree(int[] array) {
4         super(array);
5     }
6
7     int lowerBound(int value) {
8         if (value > sum[0]) {
9             return length;
10        }
11        return lowerBound(0, value, 0, 0, length - 1);
12    }
13
14    private int lowerBound(int root, int value, int totalDelta, int left,
15    int right)
16    {
17        if (left == right) {
18            return left;
19        }
20        totalDelta += delta[root];
21        int leftSum = sum[2 * root + 1] + totalDelta *
22        (((right - left) >> 1) + 1);
23        if (value <= leftSum) {
24            return lowerBound(2 * root + 1, value, totalDelta, left,
25                    (left + right) >> 1);
26        } else {
27            return lowerBound(2 * root + 2, value - leftSum, totalDelta,
28                    ((left + right) >> 1) + 1, right);
29        }
30    }
31 }
```

Пример задачи

Условие. Имеется массив целых чисел размера n. Необходимо за $O(\log n)$ уметь обрабатывать следующие запросы (a и b могут быть различны для разных запросов):

1. Добавить к заданному элементу 1;
2. Найти сумму элементов массива между a и b;
3. Уменьшить все элементы массива между a и b на 1.

При первом взгляде на задачу кажется, что дерево отрезков тут неприменимо, ведь операции производятся не на отрезках. Но если мы сможем поддерживать массив в отсортированном порядке, то вторая и третья операции на самом деле будут на отрезках.

Для того, чтобы было возможно выполнить первую операцию, будем поддерживать порядок. А именно для каждого элемента будем знать его индекс в отсортированном массиве, а для каждого элемента отсортированного массива — его индекс в изначальном массиве. Тогда для исполнения первого запроса достаточно найти в отсортированном массиве последний элемент, равный данному, увеличить его на 1 (при этом массив сохранит свойство отсортированности), и поменять местами этот последний элемент и тот, который указан в запросе в обоих порядках (если это не один и тот же элемент).

Заметим также, что третья операция не меняет нестрогий порядок (то есть если $a_i \leq a_j$ до выполнения этой операции, то это верно и после выполнения этой операции). Таким образом третья операция сводится к вычитанию на отрезке.

Ну, а вторая операция — это просто сумма на отрезке.

Для обработки запросов надо также быстро находить первый и последний элемент с данным значением в отсортированном массиве. Это делается с помощью описанного выше бинарного поиска.

Листинг 5: SampleTask

```
1 import java.util.Arrays;
2 import java.util.Comparator;
3
4 class SampleTask {
5
6     Integer[] order;
7     int[] reverseOrder;
8     int[] first;
9     int[] last;
10    int length;
11    int[] sum;
12    int[] delta;
13
14    SampleTask(final int[] array) {
15        order = new Integer[array.length];
16        for (int i = 0; i < array.length; i++) {
17            order[i] = i;
18        }
19        Arrays.sort(order, new Comparator<Integer>() {
20
21            public int compare(Integer o1, Integer o2) {
22                return array[o1] - array[o2];
23            }
24        });
25        reverseOrder = new int[array.length];
26        for (int i = 0; i < array.length; i++) {
27            reverseOrder[order[i]] = i;
28        }
29        length = array.length;
30        int nodeCount = Integer.highestOneBit(length) << 1;
```

```
31          sum = new int[nodeCount];
32          delta = new int[nodeCount];
33          first = new int[nodeCount];
34          last = new int[nodeCount];
35          int[] sortedArray = new int[array.length];
36          for (int i = 0; i < array.length; i++) {
37              sortedArray[i] = array[order[i]];
38          }
39          init(0, 0, array.length - 1, sortedArray);
40      }
41
42      void init(int root, int left, int right, int[] array) {
43          if (left == right) {
44              sum[root] = array[left];
45              first[root] = array[left];
46              last[root] = array[left];
47          } else {
48              init(2 * root + 1, left, (left + right) >> 1, array);
49              init(2 * root + 2, ((left + right) >> 1) + 1, right, array);
50              sum[root] = sum[2 * root + 1] + sum[2 * root + 2];
51              first[root] = first[2 * root + 1];
52              last[root] = last[2 * root + 2];
53          }
54      }
55
56      private int get(int from, int to) {
57          return get(0, from, to, 0, 0, length - 1);
58      }
59
60      private int get(int root, int from, int to, int totalDelta, int left,
61      int right)
62      {
63          if (from > right || to < left) {
64              return 0;
65          }
66          if (from <= left && right <= to) {
67              return sum[root] + totalDelta * (right - left + 1);
68          }
69          totalDelta += delta[root];
70          return get(2 * root + 1, from, to, totalDelta, left,
71                 (left + right) >> 1)+ get(2 * root + 2, from, to,
72                 totalDelta, ((left + right) >> 1) + 1, right);
73      }
74
75      private void add(int from, int to, int change) {
76          add(0, from, to, change, 0, length - 1);
77      }
78
79      private void add(int root, int from, int to, int change, int left,
80      int right)
81      {
82          if (from > right || to < left) {
83              return;
84          }
```

```
85          if (from <= left && right <= to) {
86              delta[root] += change;
87              sum[root] += change * (right - left + 1);
88              first[root] += change;
89              last[root] += change;
90              return;
91          }
92          add(2 * root + 1, from, to, change, left, (left + right) >> 1);
93          add(2 * root + 2, from, to, change, ((left + right) >> 1) + 1,
94          right);
95          sum[root] = sum[2 * root + 1] + sum[2 * root + 2] + delta[root] *
96          (right - left + 1);
97          first[root] = first[2 * root + 1] + delta[root];
98          last[root] = last[2 * root + 1] + delta[root];
99      }
100
101     private static int[] sortedCopy(int[] array) {
102         int[] result = array.clone();
103         Arrays.sort(result);
104         return result;
105     }
106
107     private int lowerBound(int value) {
108         return lowerBound(0, value, 0, length - 1);
109     }
110
111     private int lowerBound(int root, int value, int left, int right) {
112         if (left == right) {
113             return left;
114         }
115         if (last[2 * root + 1] >= value) {
116             return lowerBound(2 * root + 1, value - delta[root], left,
117                     (left + right) << 1);
118         } else {
119             return lowerBound(2 * root + 2, value - delta[root],
120                     ((left + right) << 1) + 1, right);
121         }
122     }
123
124     private int upperBound(int value) {
125         return upperBound(0, value, 0, length - 1);
126     }
127
128     private int upperBound(int root, int value, int left, int right) {
129         if (left == right) {
130             return left;
131         }
132         if (first[2 * root + 2] > value) {
133             return upperBound(2 * root + 1, value - delta[root], left,
134                     (left + right) << 1);
135         } else {
136             return upperBound(2 * root + 2, value - delta[root],
137                     ((left + right) << 1) + 1, right);
138         }
```

```
139     }
140
141     int query(int min, int max) {
142         int from = lowerBound(min);
143         int to = upperBound(max);
144         return get(from, to);
145     }
146
147     void add(int position) {
148         int sortedPosition = reverseOrder[position];
149         int lastSortedPosition = upperBound(get(sortedPosition,
150         sortedPosition));
151         int lastPosition = order[lastSortedPosition];
152         add(lastSortedPosition, lastSortedPosition, 1);
153         order[sortedPosition] = lastPosition;
154         order[lastSortedPosition] = position;
155         reverseOrder[position] = lastSortedPosition;
156         reverseOrder[lastPosition] = sortedPosition;
157     }
158
159     void decrease(int min, int max) {
160         int from = lowerBound(min);
161         int to = upperBound(max);
162         add(from, to, -1);
163     }
164 }
```

Двумерное дерево отрезков

Предположим, что у нас есть двумерный массив $n \times m$ и мы хотим за $O(\log n \log m)$ выполнять следующие запросы:

1. прибавить к данному элементу данное число;
2. подсчет всех элементов в прямоугольнике.

Заведем дерево отрезков (длина корня — число строк в массиве), в каждом узле которого будем хранить обычное дерево отрезков (длина корня — число столбцов в массиве), позволяющее менять один элемент и спрашивать сумму на отрезке.

Полученная структура будет выглядеть так:

Листинг 6: DoubleSegmentTree

```
1 class DoubleSegmentTree {
2
3     int rowCount, columnCount;
4     MassQuerySegmentTree[] data;
5
6     DoubleSegmentTree(int[][] array) {
7         rowCount = array.length;
8         columnCount = array[0].length;
9         int nodeCount = Integer.highestOneBit(rowCount) << 1;
```

```
10          data = new MassQuerySegmentTree[nodeCount];
11          init(0, 0, rowCount, array);
12      }
13
14      int[] init(int root, int left, int right, int[][] array) {
15          if (left == right) {
16              data[root] = new MassQuerySegmentTree(array[left]);
17              return array[left];
18          } else {
19              int[] leftSum = init(2 * root + 1, left,
20              (left + right) >> 1, array);
21              int[] rightSum = init(2 * root + 2,
22              ((left + right) >> 1) + 1, right, array);
23              int[] sum = new int[columnCount];
24              for (int i = 0; i < columnCount; i++) {
25                  sum[i] = leftSum[i] + rightSum[i];
26              }
27              data[root] = new MassQuerySegmentTree(sum);
28              return sum;
29          }
30      }
31
32      void add(int row, int column, int value) {
33          add(0, row, column, value, 0, rowCount - 1);
34      }
35
36      private void add(int root, int row, int column, int value, int left,
37      int right)
38      {
39          if (row > right || row < left) {
40              return;
41          }
42          data[root].add(column, value);
43          if (left != right) {
44              add(2 * root + 1, row, column, value, left, (left + right) >>1);
45              add(2 * root + 2, row, column, value, ((left + right) >> 1) + 1,
46              right);
47          }
48      }
49
50      int get(int fromRow, int toRow, int fromColumn, int toColumn) {
51          return get(0, fromRow, toRow, fromColumn, toColumn, 0, rowCount - 1)
            ;
52      }
53
54      private int get(int root, int fromRow, int toRow, int fromColumn,
55      int toColumn, int left, int right)
56      {
57          if (fromRow > right || toRow < left) {
58              return 0;
59          }
60          if (fromRow <= left && right <= toRow) {
61              return data[root].get(fromColumn, toColumn);
62          }
```

```
63          return get(2 * root + 1, fromRow, toRow, fromColumn, toColumn, left,
64                  (left + right) >> 1)+ get(2 * root + 2, fromRow, toRow,
65                  fromColumn, toColumn, ((left + right) >> 1) + 1, right);
66      }
67  }
```

Каждая из операций потребует $O(\log n)$ запросов к внутренним деревьям, то есть общая сложность будет $O(\log n \log m)$.

Обратная задача (изменение на прямоугольнике, запрос конкретного элемента) остается в качестве упражнения читателю. Автор не знает структуры, работающей за $O(\log n \log m)$, которая позволяла бы отвечать на запросы на прямоугольнике и осуществлять изменения на прямоугольнике.

Персистентное дерево отрезков

Предположим теперь, что мы хотим иметь возможность сделать запрос к любой из версий дерева отрезков в прошлом. Этого не очень сложно добиться, но нам придется отказаться от стандартной практики, что сыновья узла с индексом i — это узлы с индексами $2i+1$ и $2i+2$. Вместо этого мы будем для каждого узла хранить индексы его сыновей. При любой операции изменения у нас изменяются данные в $O(\log n)$ узлах. Вместо них надо создать новые узлы, все остальные узлы остаются без изменения. Каждый раз у нас изменяется индекс корня, именно его мы возвращаем в качестве результата операции изменения, в последствии этот индекс можно использовать для доступа к версии непосредственно после этого изменения. Для доступа к самой начальной версии следует использовать 0.

Листинг 7: PersistentSegmentTree

```
 1
 2 import java.util.Arrays;
 3
 4 public class PersistentSegmentTree {
 5
 6     int currentRoot;
 7     int length;
 8     int[] sum;
 9     int[] delta;
10     int[] leftChild;
11     int[] rightChild;
12     int nodeCount;
13
14     PersistentSegmentTree(int[] array) {
15         length = array.length;
16         sum = new int[4 * array.length];
17         delta = new int[4 * array.length];
18         leftChild = new int[4 * array.length];
19         rightChild = new int[4 * array.length];
20         nodeCount = 1;
21         init(0, 0, array.length - 1, array);
22     }
```

```
23
24      void init(int root, int left, int right, int[] array) {
25          if (left == right) {
26              sum[root] = array[left];
27          } else {
28              init(leftChild[root] = nodeCount++, left, (left + right) >> 1,
29              array);
30              init(rightChild[root] = nodeCount++, ((left + right) >> 1) + 1,
31              right, array);
32              sum[root] = sum[leftChild[root]] + sum[rightChild[root]];
33          }
34      }
35
36      int get(int from, int to) {
37          return get(currentRoot, from, to, 0, 0, length - 1);
38      }
39
40      int get(int from, int to, int timestamp) {
41          return get(timestamp, from, to, 0, 0, length - 1);
42      }
43
44      private int get(int root, int from, int to, int totalDelta, int left,
45      int right)
46      {
47          if (from > right || to < left) {
48              return 0;
49          }
50          if (from <= left && right <= to) {
51              return sum[root] + totalDelta * (right - left + 1);
52          }
53          totalDelta += delta[root];
54          return get(leftChild[root], from, to, totalDelta, left,
55                 (left + right) >> 1)+ get(rightChild[root], from, to,
56                 totalDelta, ((left + right) >> 1) + 1, right);
57      }
58
59      int add(int from, int to, int change) {
60          return currentRoot = add(0, from, to, change, 0, length - 1);
61      }
62
63      private int add(int root, int from, int to, int change, int left,
64      int right)
65      {
66          if (from > right || to < left) {
67              return root;
68          }
69          root = cloneNode(root);
70          if (from <= left && right <= to) {
71              delta[root] += change;
72              sum[root] += change * (right - left + 1);
73              return root;
74          }
75          leftChild[root] = add(leftChild[root], from, to, change, left,
76          (left + right) >> 1);
```

```
 77          rightChild[root] = add(rightChild[root], from, to, change,
 78          ((left + right) >> 1) + 1, right);
 79          sum[root] = sum[leftChild[root]] + sum[rightChild[root]] +
 80          delta[root] * (right - left + 1);
 81          return root;
 82      }
 83
 84      private int cloneNode(int root) {
 85          ensureCapacity();
 86          leftChild[nodeCount] = leftChild[root];
 87          rightChild[nodeCount] = rightChild[root];
 88          sum[nodeCount] = sum[root];
 89          delta[nodeCount] = delta[root];
 90          return nodeCount++;
 91      }
 92
 93      private void ensureCapacity() {
 94          if (nodeCount == sum.length) {
 95              sum = Arrays.copyOf(sum, nodeCount * 2);
 96              delta = Arrays.copyOf(delta, nodeCount * 2);
 97              leftChild = Arrays.copyOf(leftChild, nodeCount * 2);
 98              rightChild = Arrays.copyOf(rightChild, nodeCount * 2);
 99          }
100      }
101 }
```

Задачи и разборы

Задача A. Дима и знаменитый турист

Имя входного файла:	`a.in`
Имя выходного файла:	`a.out`
Ограничение по времени:	1 с
Ограничение по памяти:	256 Мб

Знаменитый турист Геннадий всегда использует кратчайшие пути в своих путешествиях. Мальчик Дима является поклонником Геннадия и собирает всю информацию, которую он может найти о нем — вырезки из газет, новости из Интернета и т.п.

Недавно Геннадий совершил путешествие. В некоторых городах по дороге его видели поклонники и оставляли об этом запись в своем блоге. Дима нашел все эти упоминания, но так как свой поиск он проводил уже по окончании путешествия, он не смог восстановить хронологический порядок записей — он знает лишь набор городов, в которых Геннадий точно побывал. Ему точно известно, что Геннадий путешествовал из какого-то города в какой-то другой по кратчайшему пути. Помогите Диме построить один из возможных путей Геннадия.

Дима пользуется только проверенными источниками, так что путь гарантированно существует.

Формат входного файла

Первая строка содержит 2 целых числа n и m ($1 \le n, m \le 10^5$) — количество городов и дорог, соответственно. Каждая из последующих m строк содержит описание одной дороги a_i, b_i, t_i ($a_i \ne b_i$, $1 \le a_i, b_i \le n$, $1 \le t_i \le 10^4$) — города на концах дороги и время путешествия по ней. В следующей строке содержится k — количество городов, которые точно посетил Геннадий. В последней строке содержится k чисел — номера этих городов. Каждый город в этом списке встречается не более одного раза.

Формат выходного файла

В первой строке выведите количество дорог, в которых побывал Геннадий, а во второй — эти дороги в порядке посещения Геннадием. В случае, если существует несколько решений — выведите любое.

Примеры

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

Разбор задачи A. Дима и знаменитый турист

Возьмем одну из вершин, которая точно лежит на пути, и запустим из нее алгоритм Дейкстры. Тогда самая удаленная вершина (назовем ее v) является одним из концов пути. Запустим из нее алгоритм Дейкстры со следующей модификацией — если мы достаем из очереди вершину, которая точно входит в путь, то все остальные вершины, которые лежат в очереди, из очереди удалим и присвоим расстоянию до них бесконечность. Тогда получившийся путь до самой удаленной от v вершины, которая точно лежит на пути, и есть ответ на задачу.

Задача B. Дима и строки

Имя входного файла:	`b.in`
Имя выходного файла:	`b.out`
Ограничение по времени:	1 с
Ограничение по памяти:	256 Мб

Мальчик Дима изучает алгоритмы поиска вхождения одной строки в другую. Более формально — он хочет найти пары индексов (i, j) такие, что подстрока строки t, начинающаяся с символа с индексом i и заканчивающаяся символом с индексом j, совпадала со строкой s.

Дима пытается придумать новый быстрый алгоритм, решающий данную задачу. Основная идея алгоритма — провести сравнение лишь некоторых символов, при этом значительно уменьшив количество возможных подходящих пар индексов. Он уже провел несколько сравнений и теперь хочет узнать — сколько еще осталось пар индексов, являющихся ответом на задачу и не противоречащих полученным им данным.

Помните, Дима еще маленький мальчик, так что мог ошибиться в измерениях. Если входные данные противоречивы выведите 0.

Алфавит, которым пользуется Дима, содержит ровно 10^{100} букв.

Формат входного файла

Первая строка содержит три целых числа n, l_s, l_t, $0 \le n \le 100$, $1 \le l_s \le l_t \le 10^9$. n — это количество проведенных сравнений, l_s — длина строки s и l_t — длина строки t. Следующие n строк содержат информацию об одном сравнении. Каждая строка содержит число i, $1 \le i \le l_s$, пробел, символ "=" или "!", пробел и число j, $1 \le j \le l_t$. Если использован символ "=", то $s_i = t_j$, а если символ "!", то $s_i \ne t_j$.

Формат выходного файла

Одно число — ответ на вопрос Димы

Примеры

b.in	b.out
6 3 10 1 ! 1 1 = 10 2 = 10 3 = 10 1 ! 5 1 ! 8	1

Разбор задачи B. Дима и строки

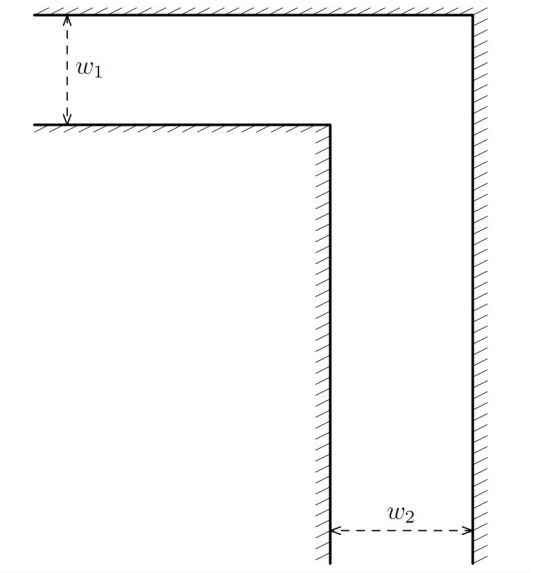
Заведем систему непересекающихся множеств, элементы которой соответствуют тем индексам в строках s и t, которые упоминаются в сравнениях. Для всех равенств во вводе объединим соответствующие данным индексам множества. Проверим все неравенства на непротиворечивость, а именно, что соответствующие индексы не лежат в одном и том же множестве, если нашли противоречие — выведем 0 и выйдем. Теперь заметим, что j однозначно восстанавливается по i (так как $j - i = length(s) - 1$) и i может быть от 1 до $length(t) - length(s) + 1$ включительно. Если ни для какого индекса строки s в сравнениях нету этого индекса, увеличенного на $i - 1$, в строке t в сравнениях, то данный сдвиг i точно является возможным. Рассмотрим i, для которых такая пара индексов найдется. Их не более n^2. Тогда скопируем нашу систему множеств и объединим в копии все пары индексов, которые встречаются в сравнениях и при этом должны быть равны между собой, если i — подходящее начало. Проверим полученную систему на противоречивость так же, как мы делали в начале. Если она не противоречива, то i — подходящее начало. Таким образом ответ — это количество начал минус число неподходящих.

Задача C. Дима и компьютерная игра

Имя входного файла:	`c.in`
Имя выходного файла:	`c.out`
Ограничение по времени:	1 с
Ограничение по памяти:	256 Мб

Мальчик Дима играет в компьютерную игру. На одном из уровней он едет на машине и ему надо проехать участок дороги, который представляет из себя поворот на 90 градусов.

Более формально этот участок состоит из двух дорог, пересекающихся под прямым углом как на рисунке:



Машина может в любой момент времени либо ехать по прямой, либо по дуге окружности. При этом направление ее движения изменяется непрерывно.

Нам надо найти путь от точки очень далеко слева до точки очень далеко внизу на дороге. Определим *радиус поворота* пути как наименьший радиус всех дуг окружностей в этом пути. С каким наибольшим *радиусом поворота* Дима сможет преодолеть этот участок дороги?

Формат входного файла

Единственная строка содержит два целых числа w_1 и w_2 ($1 \le w_1, w_2 \le 100$) — ширина левой и нижней частей дороги соответственно.

Формат выходного файла

Выведите единственно число — максимальный радиус поворота. Ответ будет засчитан в случае, если его относительная погрешность не более $1e-9$.

Примеры

c.in	c.out
10 10	34.14213562373095

Разбор задачи C. Дима и компьютерная игра

Нетрудно заметить, что нам нужно начать поворот из какой-то точки на верхнем крае дороги, имея при этом направление направо, проехать по дуге окружности, которая касается верхнего и правого краев дороги, и продолжить движение по правому краю вниз. Максимальный радиус, который при этом достижим, определяется уравнением $(r - w_1)^2 + (r - w_2)^2 = r^2$. Решив это уравнение, получим $r = w_1 + w_2 + \sqrt{2w_1w_2}$.

Задача D. Дима и перестановка

Имя входного файла:	`d.in`
Имя выходного файла:	`d.out`
Ограничение по времени:	2 с
Ограничение по памяти:	256 Мб

Мама подарила мальчику Диме перестановку $p_1, p_2, \ldots, p_n$ целых чисел от 1 до n. А еще Дима очень любит графы. Дима хочет построить ориентированный граф, в котором не менее n вершин, в котором выполняется следующее свойство — из вершины с номером i в вершину с номером j $(1 \le i, j \le n)$ путь существует тогда и только тогда, когда $i < j$ и $p_i > p_j$. Дима еще маленький и не любит очень большие графы. Он хочет, чтобы в нем было не более $30n$ вершин и $30n$ ребер. Помогите мальчику Диме.

Формат входного файла

Первая строка содержит число n $(1 \le n \le 1000)$. Вторая строка содержит перестановку.

Формат выходного файла

В первой строке выведите числа v и e — число вершин и ребер соответственно ($n \le v \le 30n$, $0 \le e \le 30n$). В следующих e строках выведите описания ребер графа — два числа от 1 до v, откуда и куда идет ребро соответственно.

Примеры

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

Разбор задачи D. Дима и перестановка

Будем строить граф постепенно, в некотором роде выполняя алгоритм сортировки слиянием. Пусть мы построили граф инверсий в котором выполняются описанное неравенство для первых n_0 вершин и граф, в котором неравенство выполняется для остальных n_1 вершин (эти графы мы создадим рекурсивным вызовом нашего алгоритма). Тогда покажем, как добавить n вершин и не более $2n$ ребер, чтобы условия выполнялось для всех вершин. Будем каждый раз брать вершину с самым большим значением из непомеченных. Создадим новую вершину, если вершина находится среди первых n_0, то проведем ребро из вновь созданной вершины в нее и ребро из предыдущей созданной в этой части вершины в нее (если она существует). Если же она соответствует вершине из остальных n_1, то проведем ребро из вершины во вновь созданную и из вновь созданной вершины в последнюю созданную для первой части (если она существует).

Задача E. Дима и проценты

Имя входного файла:	`e.in`
Имя выходного файла:	`e.out`
Ограничение по времени:	1 с
Ограничение по памяти:	256 Мб

Мама подарила мальчику Диме строку s, состоящую не менее чем из 4 букв. Дима начал с ней играть следующим образом — он берет 4 различные случайные позиции в этой строке и выписывает буквы, которые стоят на этих позициях, в том порядке, в котором они встречаются в строке. Любую последовательность из 4 различных позиций он выберет с одинаковой вероятностью. Дима недавно изучил в школе проценты и теперь его интересует, какие строки получатся с наибольшей вероятностью. Его интересуют только строки, получающиеся с вероятностью не менее 1%. Для каждой из них он хочет так же знать ее частоту, округленную вниз до целого процента.

Формат входного файла

В единственной строке содержится s, длиной от 4 до 10000.

Формат выходного файла

Выведите все строки, по одной на линию, которые получатся с вероятностью не менее 1% и их округленную частоту (со знаком процента, см. пример). Строки следует выводить в порядке убывания округленной частоты, а в случае равенства — в лексикографическом порядке. В случае,

если есть строки, вероятность появления которых менее 1%, надо в конце вывести “`Others less than 1%`”

Примеры

e.in
tests
e.out
ests 20% tess 20% test 20% tets 20% tsts 20%

e.in
aabbccdd
e.out
abcd 22% aabc 5% aabd 5% aacd 5% abbc 5% abbd 5% abcc 5% abdd 5% accd 5% acdd 5% bbcd 5% bccd 5% bcdd 5% aabb 1% aacc 1% aadd 1% bbcc 1% bbdd 1% ccdd 1%

e.in
testaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
e.out
aaaa 59% taaa 17% eaaa 8% saaa 8% Others less than 1%

Разбор задачи E. Дима и проценты

Запустим перебор следующего вида — у нас есть какой-то префикс строки, и мы хотим найти все строки длины 4 с этим префиксом, частота которых не менее одного процента. Тогда запустим динамику, которая посчитает для каждой следующей буквы число строк длины 4, которые начинаются с конкатенации заданного нам префикса и этой буквы. Далее достаточно сделать рекурсивный вызов только для тех новых префиксов, в которых число строк не менее сотой части их общего количества. Каждый рекурсивный вызов выполняется за $O(n)$, а так как для каждой длины префикса вызовов будет не более 100, проблем со временем быть не должно. Надо так же аккуратно понять, есть ли хоть одна строка, которую мы не учли.

Задача F. Дима & модуль

Имя входного файла:	`f.in`
Имя выходного файла:	`f.out`
Ограничение по времени:	1 с
Ограничение по памяти:	256 Мб

Мальчик Дима любит битовые операции с числами, потому что они быстрые, и не любит операцию взятия по модулю, потому что она медленная. Он знает, что если целое число N — степень двойки, то $x\%N = x\&(N-1)$ для любого натурального x, где $\&$ — побитовое И, а $\%$ — взятие по модулю. Он хочет распространить это на другие числа N и теперь хочет узнать, для какой доли чисел x ($1 \le x \le 10^{100} * N!$) это равенство будет верно для данного N.

Формат входного файла

В единственной строке — целое число N ($2 \le N \le 10^{18}$).

Формат выходного файла

В единственной строке — ответ на задачу в виде несократимой дроби p/q.

Примеры

f.in	f.out
4	1/1
3	1/3

Разбор задачи F. Дима & модуль

Пусть $N = 2^k N_0$, где N_0 — нечетное число. Тогда $(x\&(N-1)) \mod 2^k = x mod 2^k$. То есть $x\&(N-1) = x \mod N$ тогда и только тогда, когда $(x/2^k)\&(N_0 - 1) = (x/2^k) \mod N_0$ (деление целое). Рассмотрим пары $((x/2^k)\&(N_0-1), (x/2^k) \mod N_0)$. Несложно заметить, что каждая такая возможная пара встречается одинаковое число раз. Таким образом ответ на задачу — $\frac{1}{N_0}$.

Задача G. Дима и машина времени

Имя входного файла:	g.in
Имя выходного файла:	g.out
Ограничение по времени:	3 с
Ограничение по памяти:	256 Мб

Мама подарила мальчику Диме машину времени. К сожалению, эта машина работает только со встроенным в нее массивом длины n. Она может привести массив к состоянию на любой момент времени. Массив в машине тоже не простой, а особенный. Дима может выбрать три числа — i, j и d ($1 \le i \le j \le n$, $-1000 \le d \le 1000$), и ко всем элементам массива с индексами от i до j магически прибавится d. Дима играет со своим массивом, а мама время от времени задает ему вопросы — какова сумма всех чисел в массиве с индексами от f до t? Дима легко справился с этими вопросами, сможете ли вы?

Формат входного файла

В первой строке находятся два целых числа n и q ($1 \le n, q \le 10^5$) — количество элементов в массиве и суммарное количество операций и запросов соответственно. В следующей строке дано n чисел $a_1, a_2, \ldots, a_n$

($-1000 \le a_i \le 1000$) — начальное состояние массива. В следующих q строках заданы операции и запросы. Первый символ в строке может быть **t**, **+** или **?**. Если строка начинается с **t**, то это описание операции с машиной времени. Тогда в строке содержится еще одно число i ($1 \le i$) — номер операции или запроса, к состоянию перед выполнением которой возвращается массив. i всегда не более номера текущей операции. Если строка начинается с **+**, то это операция прибавления. Далее следуют i, j и d, ограничения на которые описаны в условии. Если строка начинается с **?**, то это запрос. Далее следуют числа f и t ($1 \le f, t \le n$).

Формат выходного файла

Для каждого запроса выведите сумму чисел в массиве с индексами от f до t, по одному результату в строке.

Примеры

g.in	g.out
3 5 1 2 3 ? 1 3 + 2 3 1 ? 1 3 t 1 ? 1 3	6 8 6

Разбор задачи G. Дима и машина времени

В данной задаче можно использовать персистентное дерево отрезков, описанное в лекции с небольшой модификацией — операция изменения тоже должна принимать номер корня, соответствующий моменту времени. Кроме того нам надо будет хранить в массиве корни дерева перед проведением соответствующей операции.

Задача H. Дима и машина времени-2

Имя входного файла:	h.in
Имя выходного файла:	h.out
Ограничение по времени:	2 с
Ограничение по памяти:	256 Мб

Мама подарила мальчику Диме машину времени. К сожалению, эта машина работает только со встроенным в нее массивом длины n. Она

может привести массив к состоянию на любой момент времени. Массив в машине тоже не простой, а особенный. Дима может выбрать два числа — i и d ($1 \le i \le n$, $-1000 \le d \le 1000$), и к элементу массива с индексом i магически прибавится d. Дима играет со своим массивом, а мама время от времени задает ему вопросы — какова сумма всех чисел в массиве с индексами от f до t? Дима легко справился с этими вопросами, сможете ли вы?

Формат входного файла

В первой строке находятся два целых числа n и q ($1 \le n, q \le 10^5$) — количество элементов в массиве и суммарное количество операций и запросов соответственно.

В следующей строке дано n чисел $a_1, a_2, \ldots, a_n$ ($-1000 \le a_i \le 1000$) — начальное состояние массива.

В следующих q строках заданы операции и запросы. Первый символ в строке может быть **t**, **+** или **?**.

Если строка начинается с **t**, то это описание операции с машиной времени. Тогда в строке содержится еще одно число i ($1 \le i$) — номер операции или запроса, к состоянию перед выполнением которой возвращается массив. i всегда не более номера текущей операции.

Если строка начинается с **+**, то это операция прибавления. Далее следуют i и d, ограничения на которые описаны в условии.

Если строка начинается с **?**, то это запрос. Далее следуют числа f и t ($1 \le f, t \le n$).

Формат выходного файла

Для каждого запроса выведите сумму чисел в массиве с индексами от f до t, по одному результату в строке.

Примеры

h.in	h.out
3 5 1 2 3 ? 1 3 + 2 2 ? 1 3 t 1 ? 1 3	6 8 6

Разбор задачи H. Дима и машина времени

В данной задаче можно использовать персистентное дерево отрезков, описанное в лекции с небольшой модификацией — операция изменения тоже должна принимать номер корня, соответствующий моменту времени. Кроме того нам надо будет хранить в массиве корни дерева перед проведением соответствующей операции.

Задача I. Дима и массив

Имя входного файла:	`i.in`
Имя выходного файла:	`i.out`
Ограничение по времени:	2 с
Ограничение по памяти:	256 Мб

Мама подарила мальчику Диме массив длины n. Массив этот не простой, а особенный. Дима может выбрать три числа — i, j и d ($1 \le i \le j \le n$, $-1000 \le d \le 1000$), и все элементы с индексами от i до j магически становятся равными d. Дима играет со своим массивом, а мама время от времени задает ему вопросы — какова сумма всех чисел в массиве с индексами от f до t? Дима легко справился с этими вопросами, сможете ли вы?

Формат входного файла

В первой строке находятся два целых числа n и q ($1 \le n \le 5 \times 10^5$, $1 \le q \le 10^5$) — количество элементов в массиве и суммарное количество операций и запросов соответственно. В следующей строке дано n чисел $a_1, a_2, \ldots, a_n$ ($-1000 \le a_i \le 1000$) — начальное состояние массива. В следующих q строках заданы операции и запросы. Первый символ в строке может быть **=** или **?**. Если строка начинается с **=**, то это операция присваивания. Далее следуют i, j и d, ограничения на которые описаны в условии. Если строка начинается с **?**, то это запрос. Далее следуют числа f и t ($1 \le f, t \le n$).

Формат выходного файла

Для каждого запроса выведите сумму чисел в массиве с индексами от f до t, по одному результату в строке.

Примеры

i.in	i.out
3 3 1 2 3 ? 1 3 = 2 3 2 ? 1 3	6 5

Разбор задачи I. Дима и массив

Создадим дерево отрезков на данном массиве и будем хранить в каждой вершине сумму всех элементов, равны ли все элементы на этом отрезке и, если да, чему они равны. Тогда при любом входе в вершину в том случае, если мы будем делать рекурсивный вызов в поддеревья, и если в данной вершине мы задали, что все элементы равны — то мы для детей данной вершины устанавливаем флаг, что все элементы равны, переносим значение, чему они равны, и высчитываем сумму, а для самой вершины этот флаг снимаем. Кроме того в операции изменения после рекурсивного вызова мы пересчитываем значение суммы в этой вершине как сумма значений в детях. Тогда в любой момент времени у нас значение суммы верно, если только в каком-либо предке данной вершины не стоит пометка, что все элементы из подмассива, соответствующего этой вершине, равны.

Задача J. Дима и большой массив

Имя входного файла:	`j.in`
Имя выходного файла:	`j.out`
Ограничение по времени:	3 с
Ограничение по памяти:	256 Мб

Мама подарила мальчику Диме массив длины n. Массив этот не простой, а особенный. Дима может выбрать два числа — i и d ($1 \le i \le n$, $-1000 \le d \le 1000$), и к элементу с индексом i магически прибавляется d. Дима играет со своим массивом, а мама время от времени задает ему вопросы — какова сумма всех чисел в массиве с индексами от f до t? Однако мама очень занята и не может задавать вопросы так часто, как обычно — всего она задала не более 1000 вопросов. Дима легко справился с этими вопросами, сможете ли вы?

Формат входного файла

В первой строке находятся два целых числа n и q ($1 \le n \le 10^6, 1 \le$

$q \leq 5 * 10^5$) — количество элементов в массиве и суммарное количество операций и запросов соответственно. В следующей строке дано n чисел $a_1, a_2, \ldots, a_n$ ($-1000 \leq a_i \leq 1000$) — начальное состояние массива. В следующих q строках заданы операции и запросы. Первый символ в строке может быть **+** или **?**. Если строка начинается с **+**, то это операция присваивания. Далее следуют i и d, ограничения на которые описаны в условии. Если строка начинается с **?**, то это запрос. Далее следуют числа f и t ($1 \leq f, t \leq n$). Гарантируется, что строк, начинающихся с **?** не более 1000.

Формат выходного файла

Для каждого запроса выведите сумму чисел в массиве с индексами от f до t, по одному результату в строке.

Примеры

j.in	j.out
3 3 1 2 3 ? 1 3 + 3 -1 ? 1 3	6 5

Разбор задачи J. Дима и большой массив

Если мы довольно оптимально напишем дерево с изменением конкретного элемента и запросом на отрезке, то это решение должно пройти. Однако, так как запросов относительно немного, мы можем использовать sqrt-декомпозицию. Тогда операция изменения будет выполняться за $O(1)$ (2 операции), а запрос — за $O(\sqrt{n})$.

Задача K. Дима и массив-2

Имя входного файла:	`k.in`
Имя выходного файла:	`k.out`
Ограничение по времени:	2 с
Ограничение по памяти:	256 Мб

Мама подарила мальчику Диме массив длины n. Массив этот не простой, а особенный. Дима может выбрать два числа — i и d ($1 \leq i \leq n$, $-1000 \leq d \leq 1000$), и элемент с индексом i магически становится равными d. Дима играет со своим массивом, а мама время от времени задает ему

вопросы — какова сумма всех чисел в массиве с индексами от f до t? Дима легко справился с этими вопросами, сможете ли вы?

Формат входного файла

В первой строке находятся два целых числа n и q ($1 \le n \le 5 \times 10^5$, $1 \le q \le 10^5$) — количество элементов в массиве и суммарное количество операций и запросов, соответственно. В следующей строке дано n чисел $a_1, a_2, \ldots, a_n$ ($-1000 \le a_i \le 1000$) — начальное состояние массива. В следующих q строках заданы операции и запросы. Первый символ в строке может быть **=** или **?**. Если строка начинается с **=**, то это операция присваивания. Далее следуют i и d, ограничения на которые описаны в условии. Если строка начинается с **?**, то это запрос. Далее следуют числа f и t ($1 \le f, t \le n$).

Формат выходного файла

Для каждого запроса выведите сумму чисел в массиве с индексами от f до t, по одному результату в строке.

Примеры

k.in	k.out
3 3 1 2 3 ? 1 3 = 3 2 ? 1 3	6 5

Разбор задачи K. Дима и массив

Данную задачу можно легко свести к задаче с прибавлением к конкретному элементу и запросу на отрезке. Достаточно перед тем, как вызвать операцию прибавления, сделать запрос об этом конкретном элементе и затем в операцию обновления передать разность того, что мы должны присвоить, и текущего значения.

Задача L. Дима и таблица

Имя входного файла:	`l.in`
Имя выходного файла:	`l.out`
Ограничение по времени:	2 с
Ограничение по памяти:	256 Мб

Мама подарила мальчику Диме таблицу размера $n \times m$. Таблица эта

не простая, а особенная. Дима может выбрать три числа — i, j и d ($1 \le i \le n, 1 \le j \le m$, $-1000 \le d \le 1000$), и элемент с индексом (i, j) магически становится равным d. Дима играет со своим массивом, а мама время от времени задает ему вопросы — какой максимальный элемент, индексы которого удовлетворяют неравенствам $from_{row} \le row \le to_{row}$ и $from_{column} \le column \le to_{column}$? Дима легко справился с этими вопросами, сможете ли вы?

Формат входного файла

В первой строке находятся три целых числа n, m и q ($1 \le n, m \le 1000$, $1 \le q \le 10^5$) — количество строк и столбцов в таблице и суммарное количество операций и запросов соответственно. В следующих n строках дано по m чисел по модулю не более 1000 — начальное состояние таблицы. В следующих q строках заданы операции и запросы. Первый символ в строке может быть **=** или **?**. Если строка начинается с **=**, то это операция присваивания. Далее следуют i, j и d, ограничения на которые описаны в условии. Если строка начинается с **?**, то это запрос. Далее следуют числа $from_{row}, to_{row}, from_{column}$ и to_{column} ($1 \le from_{row} \le to_{row} \le n, 1 \le from_{column} \le to_{column} \le n$).

Формат выходного файла

Ответы на запросы по одному в строке

Примеры

l.in	l.out
3 3 5 1 2 3 4 5 6 7 8 9 ? 1 3 1 3 = 3 3 2 ? 1 3 1 3 ? 1 3 3 3 ? 1 2 1 2	9 8 6 5

Разбор задачи L. Дима и таблица

Заведем дерево отрезков деревьев отрезков. На внутренних деревьях будем в каждой вершине хранить максимум на соответствующем отрезке. В деревьях, соответствующих одной строке, операция обновления будет просто

присваиванием в листе и присваиванием значению вершины максимума из значений в ее детях для остальных вершин после рекурсивного вызова. В деревьях, соответствующих более чем одному ряду, операция обновления будет присваиванием в каждой вершине, отрезок которой содержит колонку запроса, максимума значений в соответствующих вершинах в деревьях, являющихся потомками данного. Операция запроса ничем не отличается от стандартной операции, описанной в лекции.

Задача M. Дима и таблица-2

Имя входного файла:	`m.in`
Имя выходного файла:	`m.out`
Ограничение по времени:	2 с
Ограничение по памяти:	256 Мб

Мама подарила мальчику Диме таблицу размера $n \times m$. Таблица эта не простая, а особенная. Дима может выбрать три числа — i, j и d ($1 \le i \le n, 1 \le j \le m$, $-1000 \le d \le 1000$), и к элементу с индексом (i, j) магически прибавится d. Дима играет со своим массивом, а мама время от времени задает ему вопросы — какова сумма всех элементов, индексы которых удовлетворяют неравенствам $from_{row} \le row \le to_{row}$ и $from_{column} \le column \le to_{column}$? Дима легко справился с этими вопросами, сможете ли вы?

Формат входного файла

В первой строке находятся три целых числа n, m и q ($1 \le n, m \le 1000$, $1 \le q \le 10^5$) — количество строк и столбцов в таблице и суммарное количество операций и запросов, соответственно. В следующих n строках дано по m чисел по модулю не более 1000 — начальное состояние таблицы. В следующих q строках заданы операции и запросы. Первый символ в строке может быть **+** или **?**. Если строка начинается с **+**, то это операция прибавления. Далее следуют i, j и d, ограничения на которые описаны в условии. Если строка начинается с **?**, то это запрос. Далее следуют числа $from_{row}, to_{row}, from_{column}$ и to_{column} ($1 \le from_{row} \le to_{row} \le n, 1 \le from_{column} \le to_{column} \le n$).

Формат выходного файла

Ответы на запросы по одному в строке

Примеры

m.in	m.out
3 3 5 1 2 3 4 5 6 7 8 9 ? 1 3 1 3 + 3 3 2 ? 1 3 1 3 ? 1 3 3 3 ? 1 2 1 2	45 47 20 12

Разбор задачи M. Дима и таблица

Заметим, что для решения этой задачи достаточно применить дерево отрезков деревьев отрезков, описанное в лекции.

День седьмой (24.02.2012 г.) Контест Мистряну Ивана Леонидовича и Щепина Алексея Юрьевича

Об авторах. . .

Мистряну Иван Леонидович, родился в 1984 году в Севастополе. Окончил Филиал Московского Государственного Университета им. М. В. Ломоносова в г. Севастополе в 2006 году по специальности “математик/системный программист”. Работал преподавателем в Филиале МГУ в Севастополе. Является тренером команд по спортивному программированию Филиала МГУ в Севастополе. Участвует в организации олимпиад в Севастополе.

Щепин Алексей Юрьевич, родился в 1981 году. Участвовал в ACM-олимпиадах с 2001 года. Закончил Севастопольский национальный технический университет в 2003 и аспирантуру Морского гидрофизического института НАНУ в 2006 году. С 2005 года работает ведущим программистом в компании ProcessOne. С 2008 года готовит школьников г. Севастополя к олимпиадам по информатике.

Основные достижения:

- в составе команды СевНТУ занял 3-е место на II этапе Всеукраинской олимпиады по информатике (Запорожье, 2003);
- обладатель награды “Erlang User of the Year 2006”;
- победитель блиц-раунда ICFPC 2008;
- победитель блиц-раунда ICFPC 2009.

Задачи и разборы

Задача A. Полёт хомяка

Имя входного файла:	`a.in`
Имя выходного файла:	`a.out`
Ограничение по времени:	1 с
Ограничение по памяти:	64 Мб

В Хомякбурге проводится ежегодное соревнование летающих хомяков. В этом году правила следующие. Хомяка запускают из рогатки из некоторой точки на земле с начальной скоростью V. В воздухе находятся несколько контрольных точек. Если траектория полета хомяка пересекается с некоторой контрольной точкой, он останавливается в этой точке и снова может быть запущен из этой точки под любым углом с начальной скоростью V. Все контрольные точки находятся в одной плоскости, перпендикулярной земле. В той же плоскости находится стартовая точка. Также в воздухе расположены препятствия в виде кругов в той же плоскости. Траектория

полета хомяка не может проходить через внутреннюю часть какого-либо круга (хотя может касаться круга). Сами круги не касаются и не покрывают контрольных точек и стартовой точки. Но могут пересекаться между собой и частично уходить под землю. Хомяк должен за определенное время T достичь заданной контрольной точки, прыгая по другим контрольным точкам. Если хомяк может это сделать, он получит Q очков. Чем меньше очков, тем лучше полет. Пусть хомяк прыгал по точкам p_0, p_1, ..., p_k, где p_0 — стартовая точка, а p_k — целевая. Пусть он прилетел в точку p_i под углом a_i градусов, а вылетел под углом b_i, и пусть минимальный поворот от a_i до b_i равен по модулю c_i градусов. Тогда, для такого полета $Q = \max\{c_i\}$. Какое минимальное количество очков может получить хомяк? Размерами хомяка и рогаток, а также сопротивлением воздуха можно пренебречь. Ускорение свободного падения можно принять за $10\,\text{м/с}^2$. По правилам хомяк не может начинать и заканчивать прыжок в одной и той же контрольной точке.

Формат входного файла

Первая строка содержит числа n — количество контрольных точек, m — количество кругов, V — скорость (м/с), T — время (с). Далее в n строках идут координаты каждой контрольной точки x, y (м). Хомяк должен достичь контрольной точки, указанной последней. В следующих m строках идут координаты и радиусы каждого круга x, y, r (м). Стартовая точка имеет координаты $(0, 0)$.

Ограничения

$1 \leqslant n \leqslant 100$
$0 \leqslant m \leqslant 100$
$1 \leqslant V \leqslant 100$
$1 \leqslant T \leqslant 100$
$-100 \leqslant x \leqslant 100$
$0 \leqslant y \leqslant 100$
$1 \leqslant r \leqslant 100$

Формат выходного файла

Выведите минимальное количество очков, которое может получить хомяк с тремя знаками после запятой или '`-1`', если хомяк не сможет достичь контрольной точки за время T.

Пример

a.in	a.out
3 1 10 5 1 3 -1 2 4 0 1 1 1	44.310

Разбор задачи A. Полёт хомяка

Решение можно разбить на две части: построение графа, а затем поиск решения задачи на этом графе.

Итак, в первой части построим ориентированный граф, вершинами которого являются контрольные точки, а дугами возможные прыжки между точками. Рассмотрим прыжок из точки $(0, 0)$ в точку (x_0, y_0). Всего может быть не более двух параболических траекторий, соединяющих эти две точки. Значит, будет не более двух дуг между каждой парой точек. Для каждой дуги будем дополнительно сохранять информацию об исходящем угле, о входящем угле и о времени полета. Все это пригодится во второй части решения. Для построения дуг составим стандартные уравнения:

$$\begin{cases} x = V\cos\alpha\, t \\ y = V\sin\alpha\, t - gt^2/2 \end{cases},$$

где α — исходящий угол. Отсюда можно получить квадратное уравнение для $\tan\alpha$ (пусть $\tan\alpha = z$):

$$\frac{1}{2}gxz^2 - V^2 z + \frac{V^2 y}{x} + \frac{gx}{2} = 0.$$

И найти возможные углы, под которыми может быть совершен прыжок. Сама парабола будет иметь вид:

$$y = zx - \frac{gx^2(z^2+1)}{2v^2} = Ax^2 + Bx.$$

Далее надо проверить, не пересекает ли траектория для каждого угла препятствия в виде кругов. Для этого решим задачу пересечения параболы с окружностью. Находить точки пересечения нам не надо, можно просто проверить, что часть параболы проходит на достаточном расстоянии от центра круга (x_c, y_c). Функция расстояния от параболы до точки будет иметь вид:

$$D(x) = \sqrt{(x - x_c)^2 + (Ax^2 + Bx - y_c)^2}.$$

Тогда чтобы найти экстремумы этой функции берем производную и решаем полученное кубическое уравнение:

$$2A^2x^3 + 3ABx^2 + (1 - 2Ay_c + B^2)x - x_c - y_cB = 0$$

Для решения такого уравнения можно, например, найти один корень численно, а два других по формулам для квадратного уравнения, вынеся предварительно множитель для первого корня за скобки. Таким образом, мы отбросим все дуги, которые проходят через препятствия. Остальные добавим в граф, сохраняя необходимую дополнительную информацию.

Теперь на построенном графе решим поставленную задачу двоичным поиском по ответу. То есть, зададим максимальный возможный поворот β, который хомяк может совершить в любой контрольной точке. Таким образом, некоторые прыжки станут запрещенными. В таких условиях решим задачи поиска кратчайшего пути в графе от стартовой точки до заданной. При этом понятно, что если при данном β мы можем дойти до заданной точки за время T, то при большем β, также сможем. Найдем минимум β, что и будет ответом на задачу. Для поиска кратчайшего пути, конечно же используем алгоритм Дейкстры. В очередь с приоритетами будем сохранять дуги и релаксировать тоже будем дуги.

Асимптотика решения $O((n^2k + m\log m + mn)\log(1/\varepsilon))$, где n — количество контрольных точек, k — количество препятствий, m — количество дуг в построенном графе.

Задача B. Интервалы

Имя входного файла:	`b.in`
Имя выходного файла:	`b.out`
Ограничение по времени:	2 с
Ограничение по памяти:	64 Мб

На вещественной прямой отмечено n интервалов. Концы каждого интервала попадают в целые точки. При этом концы интервалов не принадлежат им. Известно, какие отрезки пересекаются, а какие нет. По этой информации необходимо определить, какая может быть минимальная суммарная длина всех отрезков. Также интервалы не могут иметь нулевую длину (т.е. длина каждого из них должна быть хотя бы 1).

Формат входного файла

В первой строке входа даны числа n количество интервалов, и m количество пересечений между интервалами. Далее в m строках указаны пары пересекающихся интервалов. Интервалы нумеруются числами от 0

до $n-1$. Если какая-то пара интервалов не указана, значит они не пересекаются. Гарантируется, что по данной информации можно построить набор интервалов, соответствующий ей.

Ограничения

$1 \leqslant n \leqslant 1234$
$0 \leqslant m \leqslant n(n-1)/2$

Формат выходного файла

Выведите минимально возможную суммарную длину всех интервалов. Интервалы не могут иметь нулевую длину.

Пример

b.in	b.out
3 2 0 1 1 2	4

Разбор задачи B. Интервалы

Рассмотрим граф $G = (V, E)$, который получается из условия задачи. Вершинами графа будут интервалы. Две вершины соединены ребром, только если интервалы пересекаются. Назовем клику в графе тупиковой, если её нельзя увеличить. Рассмотрим произвольный интервал и соответствующую ему вершину v. Пусть v принадлежит некоторой тупиковой клике K_1. Так как интервалы могут пересекаться только внутри единичных отрезков вида $(a, a+1)$, то этой клике отвечает единичный кусочек интервала v, на котором он пересекает другие интервалы из той же клики. Так как K_1 тупиковая, то интервалы из K_1 не пересекают вместе никаких других интервалов. Если рассмотреть другую тупиковую клику K_2, которой принадлежит v, то ей будет отвечать другой единичный кусочек v, на котором v должен пересекать другое множество интервалов. И так для каждой тупиковой клики, которой принадлежит v. То есть, v должен иметь длину не меньше количества тупиковых клик, в которые он входит. С другой стороны этого достаточно, потому что таких единичных кусочков достаточно, чтобы удовлетворить подграфу, состоящему из v и его соседей в графе G. Значит, минимальная суммарная длина каждого интервала равна количеству тупиковых клик, которым он принадлежит. Получаем, что ответом на нашу задачу будет сумма размеров тупиковых клик в графе G.

В общем случае задача нахождения всех тупиков клик в графе сложна. Но граф G особенный. Фактически он является интервальным графом.

Интервальные графы являются подмножеством хордовых графов. Хордовый граф — это граф, в котором любой цикл длины больше трех имеет хорду (то есть ребро между двумя вершинами цикла не являющимися соседними в цикле). Для хордовых графов известно, что они имеют линейное относительно количества вершин количество тупиковых клик. Получается, что можно эффективно перечислить все тупиковые клики нашего графа G.

Для этого можно воспользоваться, например, следующим алгоритмом, который называется “лексикографический поиск в ширину”. Это алгоритм обхода графа, в котором на каждом шаге мы фактически предпочитаем вершины, для которых обработано наибольшее количество соседей.

Алгоритм 1

Пусть Σ *— последовательность множеств вершин*
Добавим V *в* Σ
Result $\leftarrow$ *пустая последовательность*
while Σ *не пусто* **do**
  Выберем и удалим вершину v *из первого множества из* Σ.
  Если после этого первое множество стало пустым, удалим его из Σ
  Добавим v *в конец* Result
  Для каждого множества S *принадлежащего* Σ*, заменим* S *на* S_0*, содержащее соседей* v *из* S *и* S_1*, содержащее другие вершины из* S*. При этом поставим* S_0 *в* Σ *раньше, чем* S_1.
  Удалить пустые множества из Σ
end
return Result

Для хордового графа этот алгоритм выдает такой порядок вершин, что если его обратить, то для любой вершины v, эта вершина и все её соседи, идущие за ней, образуют клику. Исходя из этого, можно воспользоваться следующим алгоритмом для перечисления всех тупиковых клик хордового графа.

Алгоритм 2

```
Order ← Результат работы алгоритма 1
CurrentClique ← ∅
foreach вершины v из Order do
    if {v} ∪ CurrentClique образует клику в G then
        добавить v в CurrentClique
    else
        вывести CurrentClique
        CurrentClique ← {v} ∪ {все соседи v, слева от v в Order}
    end
end
вывести CurrentClique
```

Данный алгоритм перечислит все тупиковые клики графа. Просуммировав их размеры, получим ответ на нашу задачу.

Оба эти алгоритма могут быть реализованы со сложностью $O(n + m)$.

Вероятно, для решения данной задачи можно воспользоваться и другими алгоритмами поиска всех тупиковых клик, если поверить, что их будет не много.

Задача C. Иерархия

Имя входного файла:	`c.in`
Имя выходного файла:	`c.out`
Ограничение по времени:	3 с
Ограничение по памяти:	64 Мб

В некоторой организации работает n сотрудников. Будем считать, что сотрудник A является подчиненным сотрудника B, если:

1. A — непосредственный подчиненный B.

   или

2. Существует сотрудник C, такой что A непосредственный подчиненный C, и C подчиненный B.

Ни один сотрудник не является собственным подчиненным. Различные сообщения в организации передаются от A к B только в том случае, если A подчиненный B, или наоборот. При этом отношения в организации устроены так, что сообщения могут быть переданы от любого сотрудника к любому другому. Необходимо для данных сотрудников A и B определять,

является ли A подчиненным B, или B подчиненным A, или отношение подчиненности между этими сотрудниками не установлено.

Формат входного файла

В первой строке входа дано количество сотрудников n и количество запросов q. Во второй и третьей строках даны параметры запросов X, a, b и Y, c, d. Далее в n строках идет описание структуры организации. Для i-того сотрудника в $(i+3)$-й строке дана информация о его непосредственных подчиненных в отдельной строке: k_i — количество непосредственных подчиненных, а затем перечислены номера этих k непосредственных подчиненных. Сотрудники пронумерованы числами от 0 до $n-1$. Запросы имеют вид (x_j, y_j), ответ на каждый запрос должен быть 1, если y_j является подчиненным x_j; -1, если x_j является подчиненным y_j; 0 — иначе. Здесь $x_0 = X$, $y_0 = Y$, $x_j = (ax_{j-1}+b+sum_{j-1}) \mod n$, $y_j = (cy_{j-1}+d+sum_{j-1}) \mod n$, $j = 1 \ldots q-1$, а sum_{j-1} равно сумме модулей ответов на первые $j-1$ запросов.

Ограничения

$1 \leqslant n \leqslant 10^5$
$1 \leqslant q \leqslant 5 \cdot 10^5$
$0 \leqslant X, a, b, Y, c, d < n$
$n - 1 \leqslant \sum_{i=0}^{n-1} k_i < n + 100$

Формат выходного файла

Для каждого запроса выведите ответ в отдельной строке как описано выше.

c.in	c.out
6 3 1 3 5 4 1 4 1 5 3 0 2 3 2 0 4 1 4 0 2 3 4	1 0 -1

Разбор задачи C. Иерархия

Фактически в задаче дано некое отношение, и необходимо быстро отвечать на запросы о транзитивном замыкании этого отношения. Транзитив-

ное замыкание в свою очередь является отношением строгого частичного порядка. В терминах теории графов это сводится к определению достижимости между вершинами на ориентированном ациклическом графе. Для произвольных ориентированных графов это достаточно трудно, но в нашей задаче дан сильно разреженный граф, имеющий структуру почти дерева. Также количество запросов относительно больше количества вершин (ребер), поэтому мы будем стремиться быстро отвечать на запросы с, быть может, более медленным препроцессингом. Предлагаем следующую схему решения задачи.

Для начала решим задачу для ориентированного ациклического графа G без неориентированных циклов. Если забыть об ориентации дуг, то мы получим дерево T. Подвесим это дерево за любую вершину. Пометим каждое ребро в получившемся корневом дереве единицей, если соответственная дуга ориентирована от корня, или минус единицей, если к корню. Теперь пусть вершина u является предком вершины v в нашем дереве. Тогда, если сумма пометок ребер на пути от u к v равна длине этого пути, получаем, что из u достижима v в графе G. Если сумма равна минус длине, то из v достижима u. Для произвольных различных вершин u и v, пусть $w = lca(u, v)$, и $w \neq u$ и $w \neq v$. Тогда вопрос о достижимости для (u, v) сводится к вопросам о достижимости для (u, w) и для (v, w). То есть, например, из u достижима v, если из u достижима w и из w достижима v. Как отвечать на такие запросы, было рассмотрено выше. Отметим, что находить длину пути, сумму пометок ребер и lca можно стандартными алгоритмами за $O(1)$ с препроцессингом за $O(n)$ или $O(n \log n)$ для нашего дерева T.

Теперь рассмотрим ориентированный ациклический граф G, в котором могут быть неориентированные циклы. Пусть из него достаточно удалить не более чем k дуг, чтобы неориентированных циклов не осталось. По условию нашей задачи $k \leqslant 100$. Удалим из G произвольные k дуг так, чтобы не осталось неориентированных циклов. Получим граф G', для которого мы можем отвечать на запросы о достижимости, как описано выше. Пусть множество вершин $W = \{w_1, w_2, \ldots, w_m\}$ являются концами удаленных дуг ($m \leqslant 2k$). Определим отношение достижимости на G для каждой пары этих вершин: это можно сделать, определив отношение достижимости между всеми парами на G' и добавив информацию, которую дают удаленные из G дуги, а затем проведя транзитивное замыкание этого отношения (за $O(m^3)$). То есть, для каждой вершины w_i из W мы знаем множество вершин $R_i \subset W$, которые достижимы из w_i. Теперь для двух произвольных различных вершин u и v из G, будем отвечать на вопрос о достижимости следующим образом. Проверим достижимость для u и v на графе G'. Если достижимость определена, то это и будет ответом. Иначе определим для u,

какие вершины из W достижимы из u на G. Для этого объединим все R_i для тех w_i, которые достижимы из u на G'. Получим множество R_u. Для v определим все вершины из W, из которых достижима v на G. Для этого объединим все R_i для тех w_i, из которых достижима v на G'. Получим R_v. Тогда из u достижима v на G, если пересечение R_u и R_v не пусто. Достижимость u из v проверяется аналогично. Таким образом, на каждый запрос в данном случае уходит $O(m)$ времени.

Суммарно получаем сложность предложенного решения: $O(n \log n + k^3 + k^2 q)$. При этом, если реализовать операции с множествами с помощью упакованных битовых масок, то k^2 в последнем слагаемом будет иметь очень малую константу ($k^2/64$). Также можно сохранять уже вычисленные множества для каждой вершины, тогда k^2 на самом деле умножается на n, а не на q.

Задача D. Двоичные произведения

Имя входного файла:	`d.in`
Имя выходного файла:	`d.out`
Ограничение по времени:	1 с
Ограничение по памяти:	64 Мб

Есть последовательность из n единиц. Рассмотрим все различные способы вставить между ними одну и более операций умножения так, чтобы получившееся выражение имело смысл (т. е. операции умножения не должны стоять подряд, в начале и в конце последовательности). Например, для $n = 4$ есть 7 способов: 1×111, 11×11, 111×1, $1 \times 1 \times 11$, $1 \times 11 \times 1$, $11 \times 1 \times 1$ и $1 \times 1 \times 1 \times 1$. Посчитаем сумму получившихся произведений, считая, что числа записаны в двоичной системе счисления: $1_2 \times 111_2 + 11_2 \times 11_2 + 111_2 \times 1_2 + 1_2 \times 1_2 \times 11_2 + 1_2 \times 11_2 \times 1_2 + 11_2 \times 1_2 \times 1_2 + 1_2 \times 1_2 \times 1_2 \times 1_2 = 33$.

Вы должны посчитать эту сумму для произвольного n.

Ограничения

$2 \leqslant n \leqslant 10^9$

Формат входного файла

Первая строка содержит T ($1 \leqslant T \leqslant 1000$) — количество тестов. Следующие T строк содержат n.

Формат выходного файла

Для каждого n из исходных данных программа должна выдать искомую сумму по модулю 1000000007.

Пример

d.in	d.out
2 2 4	1 33

Разбор задачи D. Двоичные произведения

Обозначим искомую сумму как b_n. Если будем суммировать также исходную последовательность (т. е. добавим вариант, когда не добавляется ни одно умножение), то сумма будет равна $a_n = b_n + 2^n - 1$.

Способ 1 Рассмотрим все расстановки умножений соответствующие сумме a_n. Среди них есть одна, не содержащая умножений, которая добавляет к сумме $2^n - 1$. Остальные можно разбить на группы, содержащие в конце i единиц, $1 \leqslant i \leqslant n-1$. Каждая такая группа добавляет к сумме $(2^i - 1)a_{n-i}$.

Таким образом получаем:

$$
\begin{aligned}
a_0 &= 0 \\
a_1 &= 1 \\
a_n &= 2^n - 1 + \sum_{i=1}^{n-1}(2^i - 1)a_{n-i} \\
a_n - a_{n-1} &= 2^n - 1 + \sum_{i=1}^{n-1}(2^i - 1)a_{n-i} - \left(2^{n-1} - 1 + \sum_{i=1}^{n-2}(2^i - 1)a_{n-i-1}\right) \\
&= 2^{n-1} + a_{n-1} + \sum_{i=2}^{n-1}(2^i - 1)a_{n-i} - \sum_{i=2}^{n-1}(2^{i-1} - 1)a_{n-i} \\
&= 2^{n-1} + a_{n-1} + \sum_{i=2}^{n-1} 2^{i-1}a_{n-i} \\
a_n &= 2^{n-1} + a_{n-1} + \sum_{i=1}^{n-1} 2^{i-1}a_{n-i} \qquad (1) \\
a_n - a_{n-1} &= 2^{n-1} + a_{n-1} + \sum_{i=1}^{n-1} 2^{i-1}a_{n-i} - \left(2^{n-2} + a_{n-2} + \sum_{i=1}^{n-2} 2^{i-1}a_{n-i-1}\right) \\
&= 2^{n-2} + 2a_{n-1} - a_{n-2} + \sum_{i=2}^{n-1} 2^{i-1}a_{n-i} - \sum_{i=2}^{n-1} 2^{i-2}a_{n-i} \\
a_n &= 2^{n-2} + 3a_{n-1} - a_{n-2} + \sum_{i=2}^{n-1} 2^{i-2}a_{n-i} \qquad (2)
\end{aligned}
$$

Удвоим (2) и вычтем из него (1):

$$
\begin{aligned}
2a_n - a_n &= 2\left(2^{n-2} + 3a_{n-1} - a_{n-2} + \sum_{i=2}^{n-1} 2^{i-2}a_{n-i}\right) - \\
&\quad - \left(2^{n-1} + a_{n-1} + \sum_{i=1}^{n-1} 2^{i-1}a_{n-i}\right) \\
&= 5a_{n-1} - 2a_{n-2} + \sum_{i=2}^{n-1} 2^{i-1}a_{n-i} - \sum_{i=1}^{n-1} 2^{i-1}a_{n-i} \\
a_n &= 4a_{n-1} - 2a_{n-2} \qquad (3)
\end{aligned}
$$

Формула (3) позволяет вычислить a_n и $b_n = a_n - 2^n + 1$ с помощью быстрого возведения в степень за время $\Theta(\log n)$.

Способ 2 Аналогично первому способу можно вывести рекуррентную формулу сразу для b_n:

$$b_n = \sum_{i=1}^{n-1} (2^i - 1)(b_{n-i} + 2^{n-i} - 1)$$
$$b_n = 7b_{n-1} - 16b_{n-2} + 14b_{n-3} - 4b_{n-4}$$

Способ 3 Найдём производящую функцию для a_n:

$$\begin{aligned}
\sum_{n \geqslant 0} a_n x^n &= \sum_{k \geqslant 1} (x + 3x^2 + 7x^3 + \cdots)^k \\
&= \sum_{k \geqslant 1} \left(\frac{1}{1-2x} - \frac{1}{1-x} \right)^k \\
&= \sum_{k \geqslant 1} \left(\frac{x}{(x-1)(2x-1)} \right)^k \\
&= \frac{1}{1 - \frac{x}{(x-1)(2x-1)}} - 1 \\
&= \frac{x}{1 - 4x + 2x^2}
\end{aligned}$$

Функция рациональная, поэтому коэффициенты многочлена в знаменателе дают коэффициенты в рекуррентной зависимости:

$$a_n = 4a_{n-1} - 2a_{n-2}$$

Задача E. Упаковочная машина

Имя входного файла:	`e.in`
Имя выходного файла:	`e.out`
Ограничение по времени:	1 с
Ограничение по памяти:	64 Мб

В упаковочную машину загрузили n игрушечных Дедов Морозов и n Снегурочек. Машина может упаковать:

1. одного Деда Мороза в упаковку одного из двух видов;

2. одну Снегурочку;

3. Деда Мороза и Снегурочку в одну упаковку.

Таким образом, после того, как все игрушки будут упакованы, на выходе машины будет последовательность из упаковок 4-х видов. Сколько таких различных последовательностей может сделать машина?

Ограничения

$1 \leqslant n \leqslant 10^6$

Формат входного файла

Первая строка входного файла содержит целое число n.

Формат выходного файла

Выведите единственное целое число — количество различных последовательностей упаковок по модулю 1000003.

Пример

e.in	e.out
1	5
2	37

Разбор задачи E. Упаковочная машина

Пусть машина k раз упаковала Деда Мороза в отдельную упаковку, тогда Дед Мороз вместе со Снегурочкой был упакован $n-k$ раз, а Снегурочка отдельно k раз. Тогда ответ на задачу можно представить в виде суммы мультиномиальных коэффициентов:

$$f(n) = \sum_k 2^k \binom{n+k}{n-k,k,k} = \sum_k 2^k \binom{n+k}{n-k}\binom{2k}{k}$$
$$= \sum_k 2^k \binom{n+k}{2k}\binom{2k}{k}$$

Найдём производящую функцию для $f(n)$:

$$
\begin{aligned}
F(n) &= \sum_{n\geqslant 0} f(n)x^n \\
&= \sum_{n\geqslant 0}\sum_{k} 2^k \binom{n+k}{2k}\binom{2k}{k} x^n \\
&= \sum_{k} 2^k \binom{2k}{k} \sum_{n\geqslant 0} \binom{n+k}{2k} x^n \\
&= \sum_{k} 2^k \binom{2k}{k} x^k \sum_{n\geqslant k} \binom{n+k}{2k} x^{n-k} \\
&= \sum_{k} 2^k \binom{2k}{k} x^k \sum_{n\geqslant 0} \binom{n+2k}{n} x^n \\
&= \sum_{k} 2^k \binom{2k}{k} x^k \sum_{n\geqslant 0} \binom{-2k-1}{n} (-x)^n \\
&= \sum_{k} 2^k \binom{2k}{k} \frac{x^k}{(1-x)^{2k+1}} \\
&= \frac{1}{1-x} \sum_{k} \binom{2k}{k} \left(\frac{2x}{(1-x)^2}\right)^k \\
&= \frac{1}{1-x} \left(1 - \frac{8x}{(1-x)^2}\right)^{-1/2} \\
&= \frac{1}{\sqrt{1-10x+x^2}}
\end{aligned}
$$

Продифференцируем $F(x)$ и получим рекуррентную зависимость для $f(n)$:

$$F'(x) = \frac{5-x}{(1-10x+x^2)^{3/2}}$$

$$(1-10x+x^2)F'(x) = (5-x)F(x)$$

$$(1-10x+x^2)\sum_{n\geqslant 1} nf(n)x^{n-1} = (5-x)\sum_{n\geqslant 0} f(n)x^n$$

$$\sum_{n\geqslant 0}\big((n+1)f(n+1) - 10nf(n) + (n-1)f(n-1)\big)x^n =$$

$$\sum_{n\geqslant 0}\big(5f(n) - f(n-1)\big)x^n$$

$$(n+1)f(n+1) - (10n+5)f(n) + nf(n-1) = 0$$

Получаем рекуррентную зависимость, которую можно вычислить по заданному модулю за линейное время:

$$nf(n) = 5(2n-1)f(n-1) - (n-1)f(n-2)$$

День восьмой (25.02.2012 г.)
Контест Копелиовича Сергея Владимировича

Теоретический материал. Мощные структуры данных

• *Offline* структура данных — все K запросов даны заранее. Можно обработать их одновременно.

• *Online* структура данных — следующий запрос можно узнать только после того, как структура данных ответит на предыдущий.

• *Real время* работы — максимальное время обработки одного запроса (например, *Real время* работы операции `get` в `Системе непересекающихся множеств` = $O(\log N)$.

• *Амортизационное время* работы — максимальное время обработки **первых** K запросов, деленное на K (например, *амортизационное время* работы операции `get` в `Системе непересекающихся множеств` = $O($`обратной функции Аккермана`$)$.

Persistent Data Structures

Основной принцип: никакой уже созданный объект (например, вершина дерева) никогда в будущем не поменяется.

То, что мы хотим достигнуть: теперь, если у нас есть, например, *Persistent Array*, то операция `a[i] := x` создаст новый массив. Т.е. в результате у нас будет два массива, отличающихся только в i-м элементе.

• *Persistent Search Tree* — Вопрос: как обычно добавляются вершины в дерево поиска (здесь и далее я говорю о самом простом, несбалансированном дереве поиска)? Ответ: спускаемся вниз по дереву и в конце создаем новый лист. Чтобы лист был частью дерева, нужно сказать, что у вершины, за которую крепится лист, поменялась ссылка на левого или на правого сына. В нашем случае так делать нельзя. Нельзя менять уже существующие вершины, в частности — ссылки на левого и правого сына. Сделаем так: создадим копии всех вершин, на пути от корня до листа. Свежесозданные вершины, естественно, будут иметь левых и правых сыновей из старых вершин. Корень свежесозданного дерева — копия корня старого дерева.
Теперь мы можем написать код добавления в *Persistent Search Tree*:

```
1 Add(t, x)} {
2 if (t is empty) return new tree(x, null, null);
3 else
4   if (t.x < x) return new tree(t.x, t.l, Add(t.r, x));
```

```
5    else return new tree(t.x, Add(t.l, x), t.r);
6 }
```

• *Persistent Treap* (treap = декартово дерево) — декартово дерево основано на двух основных операциях `Split` и `Merge`. Чтобы декартово дерево стало персистентным, достаточно по ходу рекурсии создавать новые вершины и не менять старые.

• *Persistent Дерево Отрезков* — если все операции с деревом отрезков реализованы спуском сверху вниз, то опять же нужно просто создавать новые вершины и не менять старые. Тут стоит заметить, что классическая реализация дерева отрезков считает, что вершина i имеет детей $2i$ и $2i+1$. Теперь так делать нельзя. У каждой вершины придется хранить две ссылки — на левого сына и на правого.

• *Persistent Array* — можно хранить массив в дереве отрезков (или в декартовом дереве по неявному ключу :-)). Чтобы массив стал персистентным, достаточно дерево отрезков, в котором мы его храним, сделать персистентным.

• *ScanLine* (сканирующая прямая) — рассмотрим решение классической задачи: *даны N точек на плоскости и M прямоугольников, нужно для каждого прямоугольника посчитать количество точек в внутри*. Будем отвечать на запросы в *Offline*. Ответ для прямоугольника $[x_1..x_2] \times [y_1..y_2]$ = $F(x_2, [y_1..y_2]) - F(x_1 - 1, [y_1..y_2])$, где F — количество точек на полоске $[y_1..y_2]$, у которых x не больше, чем текущий. Чтобы посчитать F, отсортируем точки по x и будем в таком порядке для каждой точки (x, y) делать `sum[y]++`. В тот момент, когда мы добавили все точки с x не больше некоторого X, значение функции $F(X, [y_1..y_2])$ — это сумма на отрезке $y_1..y_2$. Чтобы посчитать эту сумму, мы храним массив `sum`, как дерево отрезков с операцией “сумма на отрезке”. Это было решение классической задачи. Теперь перейдем к вопросу “причем здесь персистентность?”. А при том, что теперь после перебора всех точек нам доступны все промежуточные деревья отрезков, и на запрос-прямоугольник мы можем отвечать в `Online`.

• **Время и память** — Сбалансированные деревья поиска и дерево отрезков все еще работают за $O(\log N)$ на запрос. При этом K запросов выделяют $K \log N$ памяти. Операции с массивом (`get(i)` и `set(i, x)`) теперь работают за $O(\log N)$.

• **Все структуры могут стать персистентными** — мир состоит из массивов, а персистентный массив — это персистентное дерево отрезков. Т.е. **любую** структуру данных вы уже можете сделать персистентной.

• *Garbage Collection* (сборка мусора) — Сейчас на примере персистентного дерева мы научимся оптимизировать память. После выполнения K запросов у нас появилась $K+1$ копия дерева, которые вместе занимают целых $O(N + K \log N)$ ячеек памяти, что печально. Печально это в том случае, если из всех $K+1$ созданных деревьев нужны нам только 3 (т.е. среди $O(N + K \log N)$ вершин только $O(N)$ являются нужными). Итак, **сборка мусора**: для каждой вершины будем хранить `count` — количество ссылок вида "родитель $\rightarrow$ ребенок" на эту вершину. Также для каждой версии увеличим `count` корня дерева на 1. Теперь, если какое то дерево с корнем `root` нам больше не нужно, то достаточно вызвать следующую функцию:

```
1 Del(t){
2   t.count--;
3   if (t.count == 0) {
4     Del(t.l);
5     Del(t.r);
6     delete t;
7   }
8 }
```

• **Замечание** — все, что сказано выше, относится к *Online* задачам и структурам. В *Offline* задачи на персистентные структуры решаются гораздо проще. Для этого достаточно представить себе все дерево версий. Например, чтобы обработать все запросы к *Persistent Array* достаточно обойти дерево версий `dfs`-ом.

SQRT-decomposition

`SQRT-decomposition` = **`корневая эвристика`**. Этот подход многолик. Сейчас вы увидите несколько его проявлений в теории алгоритмов.

• **Простая** — Рассмотрим задачу "нужно с массивом делать 2 операции `a[i] := x` и `getSum(L..R)`". Пусть мы не знаем структуру данных дерева отрезков. Давайте хранить массив `a` и суммы `a[0..K-1]`, `a[K..2K-1]` и т.д. Чтобы сделать операцию `a[i] := x`, достаточно двух действий — поменять массив и поменять одну из сумм. Чтобы посчитать `getSum(L..R)`, нам достаточно $O(K + \frac{N}{K})$ времени (отрезок $[L..R]$ делится на два хвоста и несколько уже посчитанных сумм `a[i·K..i·K+K-1]`. Если выбрать K равным $\sqrt{N}$, то время обработки запроса `getSum(L..R)` будет $\sqrt{N}$. Конец :-)

• **Почти дерево отрезков** — Если вы знаете дерево отрезков, структуру, описанную выше, можно рассматривать как двухуровневое дерево, в котором у каждой вершины $\sqrt{N}$ детей. Давайте добавим третий уровень,

тогда у каждой вершины будет $\sqrt[3]{N}$ детей, и у нас получится структура данных, которая умеет запрос `getSum(L..R)` обрабатывать за $O(\sqrt[3]{N})$, а запрос `a[i] := x` за $O(1)$. Это *Real Time*.

• **Split & Rebuild** — Научимся решать новую, более сложную задачу. Запросы: `reverse(L..R)`, `getSum(L..R)`. Сперва для исходного массива предподсчитаем суммы на префиксах (массив `sum`). Теперь, если мы не пользуемся операцией `reverse(L..R)`, то `getSum(L..R) = sum[R+1] - sum[L]`.

Опишем структуру данных, которая умеет делать `reverse`. В каждый момент времени наш новый массив — это последовательность кусков $[L_1..R_1], [L_2..R_2], \cdots, [L_k..R_k]$ старого массива. Каждый из кусков может быть перевернут, т.е. мы помним флажки $isReversed_i$. На запрос `getSum(L..R)` можно отвечать за время $O(k)$. Все, что нам нужно — поддерживать массив в таком виде. Изначально пусть у нас есть один кусок $[1..N]$ и $isReversed_1 = 0$. Вопрос: что происходит при операции `reverse`? Ответ: какие-то два куска $[L_i..R_i]$ и $[L_j..R_j]$ разобьются на два более маленьких. Также некоторый отрезок кусков теперь перевернется. Каждый кусок на этом отрезке сам по себе тоже перевернется, т.е. $isReversed_i$ поменяется на противоположный. При этом количество кусков k каждый раз увеличивается на 2. Чтобы k не росло бесконечно, когда k станет больше $\sqrt{N}$, мы перестроим всю структуру данных за $O(N)$ и количество кусков опять станет равно 1. *Амортизационное время* на один запрос получилось $\sqrt{N}$.

• **Split & Merge** — а еще для решения той же задачи можно поддерживать инвариант, что размер каждого куска от K до $2K$. Чтобы его поддерживать, нам нужны операции `Split` и `Merge` для кусков. При $K = \sqrt{N}$ мы получаем ту же асимптотику. Константа же получится заметно хуже.

• **As Treap** — Если вы помните, что декартово дерево удобно тем, что, чтобы посчитать любую $F(L..R)$, нам достаточно высплитить отрезок $[L..R]$, вернуть F, хранящуюся теперь прямо в корне, и вмерджить $[L..R]$ обратно, то вам будет приятно узнать следующее. Если для изучаемой нами сейчас структуры реализовать такие же операции `GlobalSplit` и `GlobalMerge` (для этого нам понадобится реализовать корневую именно так, как предлагается в предыдущем пункте). то появится тот же эффект, что и в декартовом дереве.

• **2D корневая** — Пусть у нас есть N точек на плоскости с координатами от 1 до N. Все x-ы и y-ы различны. Плоскость можно делить на $\sqrt{N} \times \sqrt{N}$ ячеек. Для каждой ячейки посчитать количество точек в ней и на матрице ячеек посчитать двумерные частичные суммы (т.е. научить-

ся отвечать на запрос “сколько точек в прямоугольнике ячеек” за $O(1)$). Тогда, мы за $O(\sqrt{N})$ умеем отвечать на запрос “сколько точек в прямоугольнике?”. Для этого нужно хвосты (не более $\sqrt{N}$ точек слева, справа, сверху, снизу) обработать отдельно и получить прямоугольник-запрос из цельных ячеек, ответ на который можно узнать с помощью предподсчитанных частичных сумм за $O(1)$.

• **Отложенные операции** — Еще раз научимся обрабатывать все те же операции `a[i] := x` и `getSum(L..R)`. Сперва для исходного массива предподсчитаем суммы на префиксах (массив `sum`). Теперь, если мы не пользуемся операцией `a[i] := x`, то `getSum(L..R) = sum[R+1] - sum[L]`. Пусть произошло k операций вида $a[i_j] := x_j$. Сперва преобразуем их к виду $a[i_j]$+=y_j. Для этого на момент выполнения операции нужно помнить содержимое ячейки $a[i_j]$. Теперь запрос `getSum(L..R)` обрабатывается почти так же: изначально результат = `sum[R+1] - sum[L]`. Затем мы перебираем k изменений исходного массива, и, если изменение относится к нужному нам отрезку $[L..R]$, меняем ответ. Время работы $O(k)$. k постоянно растет. Применим уже известный нам прием: если k стало больше $\sqrt{N}$, построим частичные суммы для нового массива, k станет равным 0.

• **Длины строк всегда малы** — Пусть есть задача вида: *даны N строк суммарной длины L*, тогда иногда полезно использовать тот факт, что различных длин будет $O(\sqrt{L})$ (пусть у нас есть строки длин $1, 2, \cdots, L$, тогда их суммарная длина $\frac{L(L+1)}{2}$. Пример. Пусть мы умеем с помощью хэшей искать в тексте T строку S за время $O(|T| + |S|)$, вернее $O(|T|)$ т.к. если текст короче строки, то правильный ответ “не найдена”. Решение такое: посчитаем хэш строки и хэши всех подстрок текста нужной длины. Теперь пусть нам дали более сложную задачу: *дан текст T и N строк S_i, нужно найти эти строки в тексте*. Если мы применим уже известный алгоритм, то получим решение за $O(|T|N)$. Теперь сперва научимся решать за $O(|T|)$ задачу в случае, если все длины строк одинаковы: сложим хэши всех строк в хэш-таблицу и пройдемся один раз по хэшам подстрок текста. А, если решение этой задачи понятно, то общий случай мы умеем решать за $O(|T|\sqrt{N})$ т.к. различных длин, как мы и говорили в самом начале, мало.

• **Обобщение метода двух указателей** — Научимся в *Offline* решать такую задачу: нужно для отрезков $[L_i..R_i]$ массива a посчитать $\sum\limits_x x \cdot count(x)$. Сумма берется по всем числам на отрезке, а $count(x)$ — сколько раз x встречается на отрезке. Предположим, что числа в массиве целые от 1 до 10^6. Тогда для одного отрезка мы умеем вычислять нужную сумму

за $O(R-L)$. Пусть мы знаем сумму для `[L..R]`, получим сумму для `[L..R+1]`. Мы знаем `a[R+1]`, мы знаем `count(a[R+1])`, мы умеем менять `count(a[R+1])`, этого нам должно хватить :-) Таким же образом можно делать не только операцию `R++`, но и `L--`, `L++`, `R--`. Теперь вернемся к исходной задаче. Если мы знаем, что $L_i \le L_{i+1}, R_i \le R_{i+1}$, то ответить на все запросы мы можем за суммарное время $O(N)$. Это и есть классический метод двух указателей. Вопрос: что же делать, если отрезки произвольные? Ответ: давайте все отрезки разобьем на группы: i-я группа — все отрезки, левый конец которых лежит в диапазоне `[iK..(i+1)K-1]`. Если в i-й группе m_i отрезков, то ее можно обработать за время $O(N+Km_i)$. Для этого нужно отсортировать по возрастанию R все отрезки, посчитать сумму для первого, а к следующему переходить с помощью операций `L--`, `L++`, `R++`. Операция `R++` за все время случится не более N раз. Операции `L--` и `L++` вызываются не более K раз для перехода к следующему отрезку (т.к. все левые концы "близки"). Просуммируем полученное время по всем i при $K=\sqrt{N}$. Получим $O(N\sqrt{N})$.

2D-деревья

- **КД-дерево**
  - **Квадро-дерево**
- **Сколько точек в полуплоскости?** — Даны N точек на плоскости. Сейчас мы научимся в *Online* отвечать на запрос "сколько точек в полуплоскости" за время лучшее, чем $O(N)$. Давайте предположим для простоты, что никакие три точки не лежат на одной прямой. Сперва за $O(N\log N)$ научимся искать две перпендикулярных друг другу прямых, которые делят плоскость на четыре части, в каждой из которых ровно $\frac{N}{4}$ точек (естественно, с точностью до округления). Утверждается, что если зафиксировать угол α, то можно построить две такие перпендикулярные прямые, соответственно с углами нормалей α и $\alpha+\frac{\Pi}{4}$, что каждая из них делит множество точек на две равные части. Проблема в том, что в соседних четвертях может оказаться A и B точек ($A \ne B$). Рассмотрим $F(\alpha)=A-B$. Заметим, что $F(\alpha)=-F(\alpha+\frac{\Pi}{4})$. Значит, есть корень, найдем его бинпоиском. Все :-) Теперь самое простое. Если есть разбиение плоскости таким образом, то, какой бы запрос-полуплоскость нам не пришла, прямая, образующая полуплоскость, пересекает не более трех из четырех частей плоскости. А четвертая или целиком внутри, или целиком снаружи. Осталось построить квадро дерево. Время обработки одной полуплокости будет $3^{log_4 N}=N^{\frac{log_2 3}{2}}$, что примерно равно $N^{0.792}$.

• **Простая реализация 2D-дерева** — Когда говорят страшные слова *двумерное дерево отрезков*, большинство людей представляют себе длинный и сложный код. Это не всегда правда. Рассмотрим такую задачу: дан массив, нужно в *Online* отвечать на запрос "сколько точек на отрезке `[i..j]` имеют значения от L до R?". Решение: построим дерево отрезков, в каждой вершине которого будем хранить отсортированный массив (т.е. все числа этого отрезка в отсортированном порядке). Для того, чтобы код был покороче, реализуем дерево отрезков снизу.
Самая простая процедура построения нашей структуры такова:

1. каждое число x массива положим циклом `for` в нужные вершины дерева отрезков;
2. отсортируем массивы всех вершин дерева отрезков.

Самая простая процедура ответа на запрос теперь такова:

1. поднимаемся по дереву снизу, какие-то вершины дерева отрезков покрывают наш отрезок `[i..j]`;
2. для каждой такой вершины бинпоиском найдем R и $L-1$, разность полученных индексов нужно прибавить к ответу.

Для тех, кто пишет на C++: вам достаточно функций `sort` и `lower_bound`, но для более быстрого построения вместо `sort` можно также использовать функцию `merge` от уже посчитанных детей.

• **Чуть более сложная задача** — пусть даны произвольные N точек на плоскости, опять же нужно научиться считать количество точек в прямоугольнике в *Online*. Давайте отсортируем наши точки по x-ам. i-я точка имеет теперь координаты x_i, y_i и $x_i \le x_{i+1}$. Тогда прямоугольный запрос $[x_1..x_2] \times [y_1..y_2]$ превращается в запрос на отрезке отсортированного массива `[i..j]` "сколько элементов имеют y от y_1 до y_2?". Числа i и j предлагается найти бинпоиском.

• **Замена координат** — пусть даны произвольные N точек на плоскости, нужно научиться считать что-нибудь на прямоугольнике в *Online*. Если вас смущают равные x или y, можно сделать замену координат $x, y \to z = (x, y)$ и $t = (y, x)$ (z и t — это пары чисел, сравниваются на больше-меньше они именно как пары). Тогда прямоугольный запрос $[x_1..x_2]\times[y_1..y_2]$ перейдет в $[z_1..z_2] \times [t_1..t_2]$, где $z_1 = (x_1, y_1)$ и т.д.

Классификация BST

• **Не сбалансированное дерево поиска** — чтобы освоить все последующие деревья, нужно хорошо знать, как добавляется и удаляются вершины

в обычном несбалансированном дереве. **Добавление:** спуститься вниз, создать новый лист. **Удаление:** найти нужную вершину p, если у p есть только один сын — очевидно. Если у p два сына, перейти к левому, от него спуститься вправо до упора, получить вершину q, поменять местами ключи p и q и удалить вершину q (а у нее, мы точно знаем, нет правого сына). Теперь, когда мы умеем строить обычное дерево поиска, осталось изучить методы и инварианты для балансировки.

• B, B^+ — инвариант: все листья имеют одинаковую глубину, а каждая вершина имеет степень от k до $2k-1$. Корню разрешается иметь степень меньше k. В B^+ дереве ключи хранятся во всех вершинах, в B дерево только в листьях. В B^+ дереве в вершине степени d хранится $d-1$ ключ (между каждой парой ссылок вниз один ключ). Как делать операции добавления/удаления, я описывать не буду. Это можно или придумать самостоятельно, или прочитать в википедии.

• **2-3-4, 2-3** — оба дерева это вариации B^+ дерева. 2-3 — B^+ для $k = 2$, в 2-3-4 мы дополнительно разрешили вершинам иметь степень 4. Про 2-3 деревья интересно то, что быстрая реализация такого дерева работает быстрее многих других (`AVL`, красно-черное, `set` из `C++/STL`, `treap`).

• **Red-Black, AA** — дерево называется красно-черным, если все вершины раскрашены в два цвета (красный и черный), и выполняются следующие инварианты:

1. У каждой вершины количество детей или 2 или 0.
2. Все листья черные
3. На пути от корня до любого из листьев одинаковое количество черных вершин
4. Отцом красной вершины не может быть красная вершина

Теперь заметим интересный момент: если каждую красную вершину стянуть с ее отцом в одну большую вершину, получится 2-3-4 дерево. Про 2-3-4 дерево мы помним, что без числа 4 на самом деле можно обойтись. А теперь попробуем 2-3 дерево обратным преобразованием переделать в красно-черное. Мы получим особое красно-черное дерево, в котором все красные деревья являются левыми детьми. Такое дерево проще пишется (меньше случаев) и имеет особое название — `AA`-дерево.

Еще один момент: B^+ дерево, а значит и 2-3 дерево просты. Просты тем, что там нет случаев. Процедура добавления состоит только из одной операции вида "если степень слишком большая, делай так". Поэтому, чтобы запомнить красно-черное дерево (в реализации которого случаи как раз имеются в большом количестве), по-моему, проще представлять себе красно-черное дерево, как 2-3 дерево.

• **Вращения** — следующая группа деревьев использует для перебалансировки так называемые малые и большие вращения. Малое вращение: пусть есть дерево `(a)x((b)y(c))`, тогда из него можно сделать новое `((a)x(b))y(c)`. И обратно. Это малый поворот вокруг ребра `x-y`. Большое вращение выглядит так: `((a)x((b)y(c)))z(d)` $\rightarrow$ `((a)x(b))y((c)z(d))`, вершина y при этом прыгает на две ступени вверх и становится корнем. Для реализации полезно знать, что большое вращение выражается через малое: мы сперва повернули вокруг ребра `x-y`, затем вокруг ребра `y-z`.

• **Pre-Splay** — работает не за логарифм, но хорошо иллюстрирует общую идею. Давайте сделаем добавление также, как в несбалансированное. А теперь с помощью малых вращений переместим только что добавленную вершину в корень. Как это сделать? С помощью одного малого вращения мы можем поднять вершину на единицу вверх по дереву.

• **Splay** — Модифицируем описанную выше идею. Будем поднимать вершину не на один, а на два (возможно, из-за четности, в самом конце придется сделать один подъем на единицу). При подъеме на два вверх есть два случая: `zig-zag` (дерево устроено как при большом вращении, тогда сделаем большое вращение) и `zig-zig` (дерево устроено так `(((a)x(b))y(c))z(d)`, тогда мы дерево преобразуем так: `(a)x((b)y((c)z(d)))`). Заметим, что `zig-zig` не выражается через *малое вращение*. *Амортизированное время* обработки одного запроса в `Splay` дереве = $O(\log N)$. Чтобы это было так, нужно всегда после того, как мы спускались до вершины v и тратили на это свое драгоценное время, после этого не забывать поднимать ее в корень. Этой перебалансировкой мы амортизируем время спуска.

• **AVL** — Инвариант `AVL`: $|height_{left} - height_{right}| \leq 1$. Добавление в `AVL` дерево происходит так: сперва добавляем также, как в несбалансированное. Теперь поднимаемся вверх, если в текущей вершине нарушен инвариант, малое и большое вращения нам в руки, все получится.

• **ChinaTree** — Инвариант `ChinaTree`: $|\frac{size_{left}}{size_{right}}| \leq 2$ и наоборот. Также допускается случай, что один из размеров 1, а другой 0. Добавление в `ChinaTree` дерево происходит также, как в `AVL`, только функция балансировки проще. Утверждается, что если вращать (понятно в какую сторону) малыми вращениями все, что не сбалансированно, то, во-первых, процесс сойдется, во-вторых, амортизированное время работы будет $O(\log N)$.

• **Treap, RBST** — Здесь предполагается, что читатель уже знаком с операциями `Split` и `Merge`. Инвариант декартового дерева (`treap`): у

каждой вершины есть свой y. Это случайное число. По величине y вершины образуют кучу (в корне минимум). Из определения видно, что, если все y различны, корень определяется единственным способом, и все оставшиеся вершины однозначно делятся на левое поддерево и правое поддерево. Т.е. получается, что корень — случайная из N вершин. Тут нужно как раз сказать, что такое `RBST`. Random Balanced Search Tree. Это такое дерево, в котором корнем является случайная вершина. Разница в том, что y-ов в `RBST` нет, вместо этого в операции `Merge`, чтобы выбрать новый корень, в соответствии с инвариантом, корнем с вероятностью $\frac{L_{size}}{L_{size}+R_{size}}$ становится корень левого дерева. Иначе, корень правого дерева. Плюс `RBST` перед декартовым: не нужны y-и, благодаря этому получается персистентность. С точки зрения памяти: y-ки хранить не нужно, `size` хранить нужно. Т.е. `RBST` по памяти никогда не хуже, а иногда даже лучше. С точки зрения скорости: `RBST` в два раза медленнее. Все потому, что рандом по ходу выполнения генерировать слишком долго.

• **Выражение операций друг через друга** — Для всех деревьев есть операции `Add` и `Del`. Для декартовых деревьев мы видели операции `Split` и `Merge`, в Splay-дереве видели на самом деле операцию `MakeRoot` — сделать вершину корнем дерева. Давайте поймем, какая из выше описанных операций (какой подход) полезнее, мощнее.
Утверждения:

1. `Add` и `Del` выражаются через `Split` и `Merge`
2. `Split` и `Merge` выражаются через `MakeRoot`.
3. Операции `Split` и `Merge` позволяют использовать возможности *дерева по неявному ключу*
4. Операция `MakeRoot` позволяет сэкономить время в том случае, если к одной и той же вершине мы обращаемся много раз.

Обоснования:

1. `Add(x)` : Split T by x to L and R. Merge L, (x), R.
2. `Split T by x` : MakeRoot(x). L = T.L and (T.x), R = T.R
3. Здесь для тех, кто не знает, что такое *возможности дерева по неявному ключу*, я объясню это. Если массив хранить, как дерево по неявному ключу, то появляются новые операции: поменять два куска массива местами, вставить в середину массива, удалить из середины массива. При этом обращение по индексу все еще доступно.
4. Пусть в `Splay` дереве мы всегда обращаемся только к одному элементу. Тогда первым же запросом он поднимется в корень, далее обращения к нему будут быстрее. Если мы обращаемся только к каким-то k элементам, утверждается, что *амортизированное время* обработки одного запроса будет $O(\log k)$.

Преобразование операций

• **Add** $\rightarrow$ **Build** — чтобы построить дерево из N элементов, можно все их по очереди добавить.

• **Merge** $\rightarrow$ **Add** — если мы умеем `Merge`-ить, то, чтобы добавить, нужно с-`Merge`-ить с одноэлементным множеством. Например, так устроены биномиальные (сливаемые) кучи.

• **Build, Add** $\rightarrow$ **Merge** — изначально все структуры построены операцией `Build`. Пусть теперь мы хотим какие-то две слить в одну. Перекинем меньшую в большую поэлементно операцией `Add`. Утверждается, что, каждый элемент будет добавлен $O(\log N)$ раз, т.к. после очередного добавления размер структуры, в которой он живет, увеличился хотя бы в два раза.

• **Build** $\rightarrow$ **Merge** — что делать, если мы не можем перекидывать по одному элементу, если у нас нет операции `Add`? Придумаем новую идею на примере задачи: *даны N точек на плоскости, нужно делать какой-то запрос на прямоугольнике*. Т.е. по сути мы сейчас хотим научиться *сливать 2D-деревья*. Будем хранить текущие K точек как $\log K$ 2D-деревьев, каждое из которых состоит из 2^x точек. Все x-ы различны. Теперь есть две структуры такого вида, мы их можем сложить как числа в двоичной системе счисления. Т.е. когда мы видим два множества из одинакового количества точек — 2^x, мы порождаем новое дерево из уже 2^{x+1} точек. Как мы это делаем? Просто вызываем процедуру `Build`. Почему это быстро работает? Потому что, когда точка участвует в очередном `Build`-е, размер множества, в котором она живет, удвоился. Т.е. каждая точка будет участвовать в `Build`-ах не более $\log N$ раз. Мы получили, что суммарное время на все `Build`-ы равно $Build(N \log N)$. За сколько работает `Get` в новой структуре? Мы вызовем старый `Get` от каждой из $O(\log N)$ частей, т.е. за $O(Get \cdot \log N)$.

• **Build** $\rightarrow$ **Merge, Add** — мы уже умеем делать `Build` $\rightarrow$ `Merge` и `Merge` $\rightarrow$ `Add`. Так что все ок.

• **Add = Change + Offline** — Все, о чем мы говорили выше, работало в *Online*, теперь предположим, что все запросы добавления даны нам в *Offline*. Рассмотрим задачу "запрос = посчитать количество точек в прямоугольнике". Можно сказать, что у каждой точки есть значение, и изначально все не добавленные точки имеют значение 0, а добавление точки = поменять ее значение на 1. Запрос теперь превратился в “найти сумму значений точек в прямоугольнике”. Зато добавлять новые точки (что для 2D-дерева операция не простая) уже не нужно.

• **Del = Find** — Чтобы удалить элемент, например, в сбалансированном дереве, часто достаточно найти его и сделать специальную пометку *нет тебя!* Таким образом, если вы умеете добавлять в `AVL`, но не помните, как удалять. Ничего страшного, просто не удаляйте.

• **Del = do not Del** — На разборе вы узнаете более мощный способ, как можно не удалять, если все запросы даны в *Offline*. Тут удаление будет рассматриваться, как `отмена добавления`, все запросы будут разбиты на пары (добавить-удалить). Т.е. просто каждое добавление будет действовать на отрезке `[L..R]`. Собственно способ можно прочесть в разборе задачи `A`. А сейчас реклама: дан граф, добавляются и удаляются ребра, нужно говорить в каждый момент времени, сколько компонент связности. Запросов 10^5, `TL` = 0.5 секунд.

Задачи и разборы

Задача A. Добавление и удаление точек

Имя входного файла:	`a.in`
Имя выходного файла:	`a.out`
Ограничение по времени:	1 с
Ограничение по памяти:	256 Мб

Не все $N^2 \log N$ одинаковы полезны...

Какая-то лекция

У вас в каждый момент есть мультимножество A точек на плоскости. Нужно научиться обрабатывать запросы трех типов:

- Добавить точку в мультимножество A
- Удалить точку из мультимножества A
- Вычислить $\sum_{p \in A} \max_{q \in A} distance(p, q)$.

Формат входного файла

Число запросов N ($1 \le N \le 3000$). Далее N строк, описывающие запросы, точный формат смотрите в примере. Координаты точек — целые числа от 0 до 3000. Точки могут совпадать. Запрос *удалить точку* должен удалять ровно одну точку (гарантируется, что такая точка в мультимножестве на момент запроса есть).

Формат выходного файла

После каждой операции с множеством выводите текущую сумму максимальных расстояний.
Абсолютная погрешность не должна превышать 10^{-6}.

Примеры

a.in	a.out
6 + 0 0 + 5 5 + 5 0 - 5 5 - 5 0 - 0 0	0.00000000000000000000 14.142135623730951000000 19.142135623730951000000 10.00000000000000000000 0.00000000000000000000 0.00000000000000000000

Разбор задачи A. Если не хочется, можно не удалять

Заметим, что если нам нужно только добавлять точки, то это можно легко делать за время $O(N)$. Осталось избавиться от операций удаления. У каждого добавления есть парный запрос-удаление. Если такого нет, создадим фиктивный запрос-удаление в момент времени $K+1$, где K — число запросов. Теперь все запросы разбиты на пары, можно сказать, что i-я точка живет в отрезок времени $[l_i..r_i]$. При этом все точки времени — целые от 0 до $K+1$.

Будем обрабатывать запросы деревом отрезков по времени. Чтобы обработать запросы со временем $[L..R]$ нужно сперва обработать все запросы с временим $[L..M)$, а затем обработать все запросы на отрезке $[M..R]$. Пусть мы сейчас находимся в вершине $[L..R]$. Рассмотрим точки. Если для какой-то точки $l_i < L$ и $R < r_i$, то она будет *добавлена* на протяжении обработки поддерева $[L..R]$. Добавим ее. Если же $R < l_i$ или $r_i < L$, то, наоборот, эта точка не будет в состоянии *добавлена* ни в какой момент времени обработки поддерева $[L..R]$. Теперь поймем, что точек, которые не попали ни в один из двух случаев всего $O(L-R)$ и их можно тащить за собой в рекурсии. Т.е. в момент обработки вершины дерева отрезков $[L..R]$ перебирать всего $O(L-R)$ точек. Теперь время работы $O(K \log K + \texttt{Add} \cdot \log K)$ т.к. обход дерева отрезков занимает $O(K \log K)$ времени и каждую точку этого дерева мы будем *добавлять* $O(\log K)$ раз. Осталось понять, что делать с добавленными точками, когда мы закончим обрабатывать текущую вершину. Нужно все добавления отменить. Это сделать очень просто: нужно хранить список ячеек памяти, которые мы перезаписали в процессе

добавления (их будет не больше, чем то, сколько времени мы потратили на добавление точек). А зная, как и какую память мы перезаписали и, конечно, то, что там хранилось раньше, можно все отменить.

Асимптотика полученного решения = $O(N^2 \log N)$. Заметим, что можно было для каждой точки хранить `set` и получить такую же асимптотику... Но данная реализация работает раз в 10 дольше. Возможно, из-за того, что используется $O(N^2 \log N)$ памяти с операцией `random_access`.

Задача B. Сбалансированное дерево

Имя входного файла:	`b.in`
Имя выходного файла:	`b.out`
Ограничение по времени:	0.5 с
Ограничение по памяти:	256 Мб

Дано подвешенное за вершину с номером 1 дерево из N вершин. Вершины дерева нумеруются от 1 до N и покрашены в красный и черный цвета.

Дерево называется красно-черным, если:

- У каждой вершины количество детей или 2 или 0.
- Все листья черные
- На пути от корня до любого из листьев одинаковое количество черных вершин
- Отцом красной вершины не может быть красная вершина

Ваша задача — проверить, является ли данное вам дерево красно-черным. Если же является, то нужно построить изоморфное ему 2-3-4-дерево.

Формат входного файла

Число N ($1 \le N \le 10^5$), далее N пар чисел. i-я пара чисел — отец i-й вершины и ее цвет (буква `R` = red, или буква `B` = black). Отец корня — вершина с номером -1.

Формат выходного файла

Выведите `YES` или `NO`. Если ответ `YES`, выведите 2-3-4 дерево изоморфное исходному. Дерево выведите в следующем формате: количество ребер и ребра. Нумерация вершин должна быть такая же, как и в исходном дереве.

Примеры

b.in	b.out
1 -1 R	NO
3 -1 R 1 B 1 B	YES 2 1 2 1 3
2 -1 B 1 B	NO
5 -1 R 1 B 1 R 3 B 3 B	NO
17 -1 B 1 R 1 B 2 B 2 B 4 B 4 B 5 B 5 R 9 B 9 B 3 R 3 R 12 B 12 B 13 B 13 B	YES 12 1 3 1 4 1 5 4 6 4 7 5 8 5 10 5 11 3 14 3 15 3 16 3 17
4 -1 B 1 B 1 B 1 B	NO

Примечание

2-3-4-дерево обладает двумя свойствами — во-первых, расстояния от корня до листьев одинаковы, во-вторых, число детей любого не листа — 2, 3 или 4.

Имеется в виду изоморфизм, обсуждавшийся на лекции. Если лекция не помогла, предлагается изучить тестовые примеры.

Разбор задачи B. Про красно-черное дерево

См. лекцию (раздел про красно-черные деревья).

Задача C. Count Offline

Имя входного файла:	`c.in`
Имя выходного файла:	`c.out`
Ограничение по времени:	2 с
Ограничение по памяти:	256 Мб

Вам дано множество точек на плоскости.
Нужно уметь отвечать на два типа запросов:

- `+ x y` — добавить в множество точку (x, y).
- `?` x_1 y_1 x_2 y_2 — сказать, сколько точек лежит в прямоугольнике $[x_1..x_2] \times [y_1..y_2]$. Точки на границе и в углах тоже считаются. $x_1 \le x_2$, $y_1 \le y_2$.

Формат входного файла

Число точек N ($1 \le N \le 50\,000$). Далее N точек. Число запросов Q ($1 \le Q \le 100\,000$). Далее Q запросов. Все координаты от 0 до 10^9.

Формат выходного файла

Для каждого запроса `GET` одно целое число — количество точек внутри прямоугольника.

Пример

c.in	c.out
4 0 0 1 0 0 1 1 1 5 ? 0 1 1 2 + 1 2 + 2 2 ? 1 0 2 2 ? 0 0 0 0	2 4 1

Примечание

Автор задачи уже написал некоторый код, который вы можете скачать по адресу (http://ejudge.kture.kharkov.ua/buffer/Kopeliovich/C/lib.cpp или http://ejudge.kture.kharkov.ua/buffer/Kopeliovich/C/lib.java) и использовать. Имеется код на языках `C++`, `Java`.

Написанная часть умеет делать три вещи:

- `Build(` множество точек на плоскости и их начальные значения `)`.
- `ChangeValue(` индекс точки в множестве, ее новое значение `)`.
- `GetSum(` прямоугольник `)`.

Время работы: `Build` за $O(N \log N)$, `ChangeValue` за $O(\log^2 N)$, `Get` за $O(\log^2 N)$.

Разбор задачи C. Count-Offline

См. лекцию (как из `Build` и `Change` получить `Add` в *Offline*).

Задача D. Count Online

Имя входного файла:	`d.in`
Имя выходного файла:	`d.out`
Ограничение по времени:	3 с
Ограничение по памяти:	256 Мб

Вам дано множество точек на плоскости.

Нужно уметь отвечать на два типа запросов: Вам дано множество точек на плоскости.

Нужно уметь отвечать на два типа запросов:

- `?` x_1 y_1 x_2 y_2 — сказать, сколько точек лежит в прямоугольнике $[x_1..x_2] \times [y_1..y_2]$.Точки на границе и в углах тоже считаются. $x_1 \le x_2$, $y_1 \le y_2$.
- `+` x y — добавить в множество точку `(x + res % 100, y + res % 101)`.Где `res` — ответ на последний запрос вида `?`, а `%` — операция взятия по модулю.

Формат входного файла

Число точек N ($1 \le N \le 50\,000$). Далее N точек. Число запросов Q ($1 \le Q \le 100\,000$). Далее Q запросов. Все координаты от 0 до 10^9.

Формат выходного файла

Для каждого запроса `GET` одно целое число — количество точек внутри прямоугольника.

Пример

d.in	d.out
5 0 0 1 0 0 1 1 1 1 1 9 ? 0 1 1 2 + 1 2 + 2 2 ? 1 0 2 2 ? 0 0 0 0 + 3 3 ? 3 3 3 3 ? 4 3 4 3 ? 4 4 5 5	3 3 1 0 0 3

Примечание

На самом деле добавлялись точки `(4, 5)`, `(5, 5)`, `(4, 4)`.

Разбор задачи D. Count-Online

Предлагается эту задачу решать 2D-деревом с отложенными операциями. В условии была сделана попытка запретить все *Offline*-овые решения.

Задача E. Динамический Лес

Имя входного файла:	`e.in`
Имя выходного файла:	`e.out`
Ограничение по времени:	0.5 с
Ограничение по памяти:	256 Мб

Вам нужно научиться обрабатывать 3 типа запросов:

1. Добавить ребро в граф (`link`).
2. Удалить ребро из графа (`cut`).
3. По двум вершинам a и b вернуть длину пути между ними (или -1, если они лежат в разных компонентах связности) (`get`).

Изначально граф пустой (содержит N вершин, не содержит ребер). Гарантируется, что в любой момент времени граф является лесом. При добавлении ребра гарантируется, что его сейчас в графе нет. При удалении ребра гарантируется, что оно уже добавлено.

Формат входного файла

Числа N и M ($1 \le N \le 10^5 + 1$, $1 \le M \le 10^5$) — количество вершин в дереве и, соответственно, запросов. Далее M строк, в каждой строке команда (`link` или `cut`, или `get`) и 2 числа от 1 до N — номера вершин в запросе.

Формат выходного файла

В выходной файл для каждого запроса `get` выведите одно число — расстояние между вершинами, или -1, если они лежат в разных компонентах связности.

Примеры

e.in	e.out
3 7 get 1 2 link 1 2 get 1 2 cut 1 2 get 1 2 link 1 2 get 1 2	-1 1 -1 1
5 10 link 1 2 link 2 3 link 4 3 cut 3 4 get 1 2 get 1 3 get 1 4 get 2 3 get 2 4 get 3 4	1 2 -1 1 -1 -1

Разбор задачи E. Link-cut

Здесь приведена основная, но очень мощная, идея. Без доказательства.

У нас в каждый момент времени имеется покрытие путями. Вопрос: что такое покрытие путями? Ответ: у дерева есть корень. Каждая вершина покрыта ровно одним вертикальным путем (путем идущим от своего начала только вниз). От каждой вершины мы знаем ссылку на путь, в котором она содержится, а от каждого пути на вершину-родителя начала пути. Таким образом мы можем быстро подниматься по дереву. Вершина $\rightarrow$ путь $\rightarrow$ начало пути $\rightarrow$ прыжок к родителю, который уже содержится в другом пути.

Каждый путь будет храниться в декартовом дереве по неявному ключу (или в Splay дереве). На самом деле — мы можем воспользоваться любыми деревьями, которые поддерживают операции `Split`, `Merge` и `Reverse`.

Изначально покрытие тривиально: каждая вершина — сама по себе путь. Введем самою главную операцию — `Expose(v)`. Эта операция делает так, что вертикальный путь, начинающийся в корне, заканчивается ровно в v.

Для этого нужно подняться из v до корня, а по пути что-то померджить и посплитить.

Вторая важная операция — `MakeRoot(v)`. Эта операция подвешивает все дерево за вершину v. Утверждается, что после того, как мы научились делать `Expose(v)`, достаточно сделать `Expose(v)` и `reverse(Tree(v))`.

Теперь мы готовы написать код операций `Link`, `Cut` и `Get`.

`Link(a,b)` : `MakeRoot(a)`, `parent[a] = b`

`Cut(a,b)` : `MakeRoot(a)`, `Expose(b)`, `Split(tree[b], 1)` `parent[b] = -1`

`Get(a,b)` : `MakeRoot(a)`, `Expose(b)`, `answer = tree[b].size`

Задача F. Самая дальняя

Имя входного файла:	`f.in`
Имя выходного файла:	`f.out`
Ограничение по времени:	1 с
Ограничение по памяти:	256 Мб

Даны N точек на плоскости, нужно уметь обрабатывать следующие запросы:

- `get a b` — возвращает максимум по всем точкам величины $ax + by$.
- `add x y` — добавить точку в множество.

Формат входного файла

Число N ($1 \le N \le 10^5$) и N точек. Далее число M ($1 \le M \le 10^5$ — количество запросов и собственно запросы. Формат запросов можно посмотреть в примере. Все координаты точек и числа `a`, `b` — целые числа, по модулю не превосходящие 10^9.

Формат выходного файла

На каждый запрос вида `get` выведите одно целое число — максимум величины $ax + by$.

Примеры

f.in	f.out
3 0 0 1 0 0 1 10 get 1 1 get -1 -1 get 1 -1 get -1 1 add 2 2 add -2 -2 get 1 1 get -1 -1 get 1 -1 get -1 1	1 0 1 1 4 4 1 1

Примечание

Автор задачи уже написал некоторый код, который вы можете скачать по адресу (http://ejudge.kture.kharkov.ua/buffer/Kopeliovich/F/lib.cpp или http://ejudge.kture.kharkov.ua/buffer/Kopeliovich/F/lib.java) и использовать. Имеется код на языках `C++`, `Java`.

Написанная часть умеет делать две вещи:

- `Build(` множество точек на плоскости `)`.
- `GetMax(a, b)`.

Время работы: `Build` за $O(N \log N)$, `GetMax` за $O(\log N)$.

Разбор задачи F. The most far

См. лекцию (как из `Build` получить `Add`).

Задача G. Persistent Array

Имя входного файла:	`g.in`
Имя выходного файла:	`g.out`
Ограничение по времени:	0.5 с
Ограничение по памяти:	256 Мб

Дан массив (вернее, первая, начальная его версия).

Нужно уметь отвечать на два запроса:

- $a_i[j] = x$ — создать из i-й версии новую, в которой j-й элемент равен x, а остальные элементы такие же, как в i-й версии.
- `get` $a_i[j]$ — сказать, чему равен j-й элемент в i-й версии.

Формат входного файла

Количество чисел в массиве N ($1 \le N \le 10^5$) и N элементов массива. Далее количество запросов M ($1 \le M \le 10^5$) и M запросов. Формат описания запросов можно посмотреть в примере. Если уже существует K версий, новая версия получает номер $K + 1$. И исходные, и новые элементы массива — целые числа от 0 до 10^9. Элементы в массиве нумеруются числами от 1 до N.

Формат выходного файла

На каждый запрос типа `get` вывести соответствующий элемент нужного массива.

Примеры

g.in	g.out
6 1 2 3 4 5 6 11 create 1 6 10 create 2 5 8 create 1 5 30 get 1 6 get 1 5 get 2 6 get 2 5 get 3 6 get 3 5 get 4 6 get 4 5	6 5 10 5 10 8 6 30

Разбор задачи G. Persistent Array

Решаем в *Offline*. Строим дерево версий, в каждой вершине храним список запросов к этой версии. Теперь обходим дерево `dfs`-ом, меняя массив при проходе по ребрам (когда идем по ребру, обратно, вверх, откатываем изменение).

Задача H. Перестановки strike back

Имя входного файла:	`h.in`
Имя выходного файла:	`h.out`
Ограничение по времени:	1.5 с
Ограничение по памяти:	256 Мб

Вася выписал на доске в каком-то порядке все числа от 1 по N, каждое число ровно по одному разу. Иногда он стирает какое-то число и записывает на его место другое. Количество чисел, выписанных Васей, оказалось довольно большим, поэтому Вася не может окинуть взглядом все числа. Однако ему надо всё-таки представлять эту последовательность, поэтому он написал программу, которая в любой момент отвечает на вопрос — сколько среди чисел, стоящих на позициях с x по y, по величине лежат в интервале от k до l. Сделайте то же самое.

Формат входного файла

В первой строке лежит два натуральных числа — $1 \leqslant N \leqslant 100\,000$ — количество чисел, которые выписал Вася, и $1 \leqslant M \leqslant 100\,000$ — суммарное количество вопросов и изменений, сделанных Васей. Во второй строке дано N чисел — последовательность чисел, выписанных Васей. Далее в M строках находятся описания вопросов. Каждый запрос на изменение числа в некоторой позиции начинается со слова `SET` и имеет вид `SET a b` ($1 \leqslant a \leqslant N$, $1 \leqslant b \leqslant N$). Это означает, что Вася изменил число, записанное в позиции a на число b. Каждый Васин вопрос начинается со слова `GET` и имеет вид `GET x y k l` ($1 \leqslant x \leqslant y \leqslant N$, $1 \leqslant k \leqslant l \leqslant N$).

Формат выходного файла

Для каждого Васиного вопроса выведите единственное число — ответ на Васин вопрос.

Примеры

h.in	h.out
4 4 1 2 3 4 GET 1 2 2 3 GET 1 3 1 3 SET 1 4 GET 1 3 1 3	1 3 2

Разбор задачи H. Permutation2

Внезапно, в этой задаче корневая эвристика работает быстрее, чем 2D-дерево. А, да, собственно решение этой задачи: или отложенные операции (см. лекцию), или вариант корневой оптимизации, в которой мы храним для каждого куска его отсортированную версию.

Задача I. Persistent List

Имя входного файла:	`i.in`
Имя выходного файла:	`i.out`
Ограничение по времени:	2 с
Ограничение по памяти:	512 Мб

Даны N списков. Каждый состоит из одного элемента.
Нужно научиться совершать следующие операции:

- `merge` — взять два каких-то уже существующих списка и породить новый, равный их конкатенации.
- `head` — взять какой-то уже существующий список L и породить два новых, в одном первый элемент L, во втором весь L кроме первого элемента.
- `tail` — взять какой-то уже существующий список L и породить два новых, в одном весь L кроме последнего элемента, во втором последний элемент L.

Для свежесозданных списков нужно говорить сумму элементов в них по модулю $10^9 + 7$.

Формат входного файла

Число N ($1 \le N \le 10^5$). Далее N целых чисел от 1 до 10^9 — элементы списков. Исходные списки имеют номера — $1, 2, \cdots, N$.

Затем число M ($1 \le M \le 10^5$) — количество операций. Далее даны операции в следующем формате:

- `merge i j`
- `head i`
- `tail i`

Где `i` и `j` — номера уже существующих списков. Если в текущий момент имеется K списков, новый список получает номер $K + 1$.

Для операций `head` и `tail` считается, что сперва порождается левая часть, затем правая (см. пример). Также вам гарантируется, что никогда не будут порождаться пустые списки.

Формат выходного файла

Для каждого нового списка нужно вывести сумму элементов по модулю $10^9 + 7$.

Примеры

i.in	i.out
4 1 2 3 4 6 merge 1 2 merge 3 4 merge 6 5 head 7 tail 9 merge 2 3 merge 1 1	3 7 10 3 7 5 2 5 2

Разбор задачи I. Persistent List

Будем хранить список, как персистентное декартово дерево. Только вот после N мерджей список может иметь длину 2^N. Поскольку нам никогда не понадобится более M элементов из начала и из конца, то можно хранить в декартовом дереве не все, а только $2M$ элементов (M первых и M последних).

Задача J. Проекция в R^3

Имя входного файла:	`j.in`
Имя выходного файла:	`j.out`
Ограничение по времени:	1 с
Ограничение по памяти:	256 Мб

Даны N трехмерных точек. Нужно для каждой найти любую ближайшую точку. Расстояние между точками равно $\sqrt{(x_i - x_j)^2 + (y_i - y_j)^2 + (z_i - z_j)^2}$.

Формат входного файла

Число точек N ($2 \le N \le 3 \cdot 10^4$) и N точек. Каждая точка задается тремя координатами `x`, `y`, `z`. Все координаты — целые числа от 0 до 10^9.

Формат выходного файла

Выведите N чисел — для каждой точки номер ближайшей к ней точки (от 1 до N).

Пример

j.in	j.out
6 0 0 0 2 0 0 2 2 0 0 2 0 1 1 0 0 4 0	5 5 5 5 3 4
3 0 0 0 0 0 0 1 1 1	2 1 2

Разбор задачи J. Проецирование на прямую

Сперва решим задачу на плоскости. У каждой точки есть две координаты — x_i и y_i. Давайте, посчитаем новую, третью координату: $z_i = x_i \cdot cos(a) + y_i \cdot sin(a)$. Где a — некоторые случайный угол. На самом деле, z_i — проекция точки (x_i, y_i) на прямую. Отсортируем все точки по координате z. Теперь, чтобы найти ближайшую точку для точки i будем перебирать кандидатов в порядке увеличения расстояния по z ($|z_i - z_j| \uparrow$). Пусть текущее минимальное расстояние до точки i равно $best_i$. Если для текущего j $|z_i - z_j| \ge best_i$, значит, ответ уже точно не улучшится, т.е. $dist(i, j) \ge |z_i - z_j|$ для любых i и j.

Обобщение до трехмерного пространства: $t_i = x_i n_x + y_i n_y + z_i n_z$, где n — случайный нормированный вектор. Самая трудоемкая операция — *посчитать расстояние между двумя точками*. Если ввести еще одну координату-проекцию и перед подсчетом расстояния проверять, имеет ли это смысл, то решение ускорится еще в 2-3 раза.

Утверждается, что суммарное время обработки всех точек будет $O(N\sqrt{N})$ (в предположении, что все координаты — целые числа от 1 до

N). Оценка $O(N\sqrt{N})$ достигается на тесте вида "все целые точки внутри круга радиуса R". В предположении, что все координаты — произвольные числа, существует тест, на котором время работы будет $\Theta(N^2)$ с вероятностью $\frac{1}{3}$. Тест такой: i-я точка имеет координаты $(x_{i-1} + 2^{N-i}, 0)$, первая точка имеет координаты $(0, 0)$. Если угол a отстоит от 0 и от π хотя бы на 60 градусов, то для каждой точки мы будем просматривать все точки с большими индексами.

Задача K. Прямоугольные запросы

Имя входного файла:	`k.in`
Имя выходного файла:	`k.out`
Ограничение по времени:	4 с
Ограничение по памяти:	256 Мб

Даны N точек на плоскости, у каждой точки есть ценность. Нужно быстро обрабатывать запросы двух типов:

- Присвоить всем точкам в области $[x_1..x_2] \times [y_1..y_2]$ ценность K.
- Найти точку с минимальной ценностью в области $[x_1..x_2] \times [y_1..y_2]$.

Формат входного файла

Число точек N ($1 \le N \le 262\,144$) и N точек. Каждая точка задается тремя числами — x, y, начальная ценность.

Число запросов M ($1 \le M \le 10^4$) и M запросов в формате `=` x_1 y_1 x_2 y_2 $value$ для присваивания и `?` x_1 y_1 x_2 y_2 для взятия минимума.

Все координаты от -10^9 до 10^9. Все ценности от 0 до 10^9.

Формат выходного файла

На каждый запрос `?` выведите минимальную ценность точек в прямоугольнике. Если в прямоугольнике нет ни одной точки, выведите `NO`.

Пример

k.in	k.out
4 1 1 1 -1 1 1 -1 -1 1 1 -1 1 7 = 0 0 3 3 2 = -3 -3 0 0 2 ? 0 0 3 3 ? -3 -3 3 3 = -1 -1 1 1 0 ? 0 0 0 0 ? -1000 -1000 1000 1000	2 1 NO 0

Разбор задачи K. Rectmin

Эту задачу предполагалось решать КД-деревом. См. лекцию.

Задача L. Точки в полуплоскости

Имя входного файла:	`l.in`
Имя выходного файла:	`l.out`
Ограничение по времени:	0.75 с
Ограничение по памяти:	256 Мб

Есть N точек на плоскости. Точки равномерно распределены внутри квадрата $[0..C] \times [0..C]$. Вам нужно научиться отвечать на запрос “сколько точек лежит в полуплоскости”?

Формат входного файла

Число точек N ($1 \le N \le 5 \cdot 10^4$), число запросов M ($1 \le M \le 5 \cdot 10^4$), константа C (целое число от 1 до 10^4). Далее N точек (X, Y) с целочисленными координатами. Далее M полуплоскостей (a, b, c). Числа a, b, c — целые, по модулю не превосходят 10^4. $a^2 + b^2 \neq 0$. Считается, что точка лежит в полуплоскости тогда и только тогда, когда $ax + by + c \ge 0$.

Формат выходного файла

Для каждого из M запросов одно целое число — количество точек в полуплоскости.

Пример

l.in	l.out
3 4 10 5 5 1 7 7 4 1 1 -9 1 1 -10 1 1 -11 1 1 -12	2 2 1 0

Разбор задачи L. Semiplane

Разобьем квадрат $[0..C] \times [0..C]$ на $\sqrt{N} \times \sqrt{N}$ маленьких квадратиков. Прямая полуплоскости пересекает не более $2\sqrt{N}$ квадратиков, остальные целиком снаружи или внутри и их можно учесть с помощью частичных сумм. На матрице $\sqrt{N} \times \sqrt{N}$. В пересеченных же $2\sqrt{N}$ квадратах мат-ожидание количества точек = $2\sqrt{N}$. Время ответа на запрос получилось $O(\sqrt{N})$.

Задача M. Жесть

Имя входного файла:	`m.in`
Имя выходного файла:	`m.out`
Ограничение по времени:	4 с
Ограничение по памяти:	256 Мб

Дан массив из N чисел. Нужно уметь обрабатывать 3 типа запросов:

- `get(L, R, x)` — сказать, сколько элементов отрезка массива $[L..R]$ не меньше x.
- `set(L, R, x)` — присвоить всем элементам массива на отрезке $[L..R]$ значение x.
- `reverse(L, R)` — перевернуть отрезок массива $[L..R]$.

Формат входного файла

Число N ($1 \le N \le 10^5$) и массив из N чисел. Далее число запросов M ($1 \le M \le 10^5$) и M запросов. Формат описания запросов предлагается понять из примера. Для всех отрезков верно $1 \le L \le R \le N$. Исходные числа в массиве и числа x в запросах — целые от 0 до 10^9.

Формат выходного файла

Для каждого запроса типа `get` нужно вывести ответ.

Примеры

m.in	m.out
5 1 2 3 4 5 6 get 1 5 3 set 2 4 2 get 1 5 3 reverse 1 2 get 2 5 2 get 1 1 2	3 1 3 1

Разбор задачи M. SqrtReverse

Давайте воспользуемся корневой оптимизацией с операциями `Split` и `ReBuild`. Для каждого куска будем хранить $L, R, isReversed, setValue$. Если $setValue \neq -1$, то всему отрезку присвоено значение $setValue$.

Задача N. Подстроки со сдвигом

Имя входного файла:	`n.in`
Имя выходного файла:	`n.out`
Ограничение по времени:	2 с
Ограничение по памяти:	256 Мб

Вам даны K текстов. Все тексты имеют одинаковую длину.

Ваша задача — научиться искать `подстроку со сдвигом`. Подстрока S со сдвигом $a_1, a_2, \cdots, a_K$ входит в набор из K текстов $T_1, T_2, \cdots, T_K$, если существует такое число x, что для всех i $LCP(T_i + a_i + x, S) \ge |S|$. Где `LCP` — длина наибольшего общего префикса, $(T_i + j)$ — j-й суффикс строки T_i, $|S|$ — длина строки S.

Формат входного файла

Число K от 1 до 10 и K текстов (длины текстов одинаковы и лежат от 1 до 10^5). Далее M от 1 до 10^5 — число запросов и сами запросы. Каждый запрос это строка и K чисел от -10^9 до 10^9. Суммарная длина всех строк в запросах не более 10^5. Все строки и тексты состоят только из маленьких символов английского алфавита. **Все строки S по всем запросам различны**.

Формат выходного файла

Для каждого запроса выведите `NO` или `YES x` (x — величина из условия).

Пример

n.in	n.out
3 abacabaa ababbbaa aababbbb 4 a 0 0 1 b 0 0 0 ba 0 0 -1 aa 6 6 0	YES 0 YES 5 NO YES 0

Разбор задачи N. TotalSubstr

• Научимся за $O($`суммарной длины текстов`$)$ отвечать на все запросы с $|S| = len$ при фиксированном len. Алгоритм: перебираем хэши подстрок длины len первого текста, если текущий хэш совпал с хэшом строки какого-то i-го запроса, то нужно проверить еще $K - 1$ хэшей в других текстах. Получаем число i и факт *совпал/не совпал* мы за $O(1)$ с помощью хэш-таблицы.

• Различных длин, как говорилось в лекции, $O(\sqrt{sumLen})$. Итого, наше решение работает за $O(|T|\sqrt{|S|})$

Задача O. Антитест

Имя входного файла:	`o.in`
Имя выходного файла:	`o.out`
Ограничение по времени:	0.3 с
Ограничение по памяти:	256 Мб

Вам дана реализация декартова дерева. Ошибок тут нет, не волнуйтесь. Есть одна особенность: вместо `y`-ков в операции `Merge` используется `rnd.next(2)` (функция возвращает случайное число от 0 до 1).

Утверждается, что построенное таким образом дерево может иметь слишком большую глубину. Ваша задача — найти последовательность из не более чем 3000 операций, создающую дерево глубины не менее 1000 (дерево из одной вершины имеет глубину 1).

```
1 typedef struct tree * ptree;
2 struct tree { ptree l, r; int x; };
3 ptree root, a, b, c;
4
5 // comment: l = {y < x}, r = {y >= x}
6 void Split( ptree t, ptree &l, ptree &r, int x ) {
7   if (t == 0)         l = r = 0;
8   else if (t->x >= x) Split(t->l, l, t->l, x), r = t;
9   else                Split(t->r, t->r, r, x), l = t;
10 }
11 void Merge( ptree &t, ptree l, ptree r ) {
12   if (l == 0)           t = r;
13   else if (r == 0)      t = l;
14   else if (rnd.next(2)) t = l, Merge(t->r, t->r, r);
15   else                  t = r, Merge(t->l, l, t->l);
16 }
17 void Add( int x ) {
18   b = new tree(x),
19   Split(root, a, c, x);
20   Merge(a, a, b);
21   Merge(root, a, c);
22 }
23 void Del( int x ) {
24   Split(root, a, b, x);
25   Split(b, b, c, x + 1);
26   Merge(root, a, c);
27 }
```

Формат входного файла

Вам предоставлен абсолютно пустой файл. Дерево изначально также пусто.

Формат выходного файла

Последовательность операций вида `ADD x` и `DEL x`.

`ADD x` вызывает функцию `Add(x)` (см. реализацию)
`DEL x` вызывает функцию `Del(x)` (см. реализацию)

Пример

o.in	o.out
	ADD 1 ADD 2 DEL 3 DEL 2 DEL 1

Примечание

В чекере используется `Random` не зависящий от времени. Т.е. одинаковые решения всегда получают одинаковый результат. Если вы получили `PE`, ваш вывод не является корректной последовательностью из не более чем 3000 операций. Если вы получили `WA`, последовательность операций корректна, а получившееся дерево имеет глубину меньше 1000.

Разбор задачи O. Treaptest

Подходит почти любой output кроме рандома. Пример: $1\ 2\ 3 \cdots N$.

День восьмой (25.02.2012 г.) Контест Эльдара Богданова и Ираклия Мерабишвили

Об авторах...

Богданов Эльдар Фаикович, родился в 1988 году в городе Тбилиси. Окончил Тбилисский Государственный Университет со степенью бакалавра компьютерных наук. Занимался программированием в учебном центре "Мзиури". В 2009 году окончил Тбилисский Государственный Университет со степенью бакалавра компьютерных наук. С 2010 года — тренер команд Тбилисского Государственного Университета по спортивному программированию.

Основные достижения:

- призер республиканской Олимпиады школьников по информатике 2005 года;
- третий призер Открытого чемпионата Москвы 2008 года, Кубка Векуа 2008 и Открытого чемпионата Украины 2007 и 2009 года;
- второй призер Открытого чемпионата Южного Кавказа 2007 года, Олимпиады имени Лебедева и Глушкова 2008 года и Зимней Школы по программированию 2009 года;
- полуфиналист Google Code Jam 2008 года;
- член команды “Tbilisi SU Triumvirate”, которая является первой командой Грузии, ставшей призёром Чемпионата Мира: 11-е место и бронзовые медали на Чемпионате Мира ACM ICPC 2009 года.

Мерабишвили Ираклий, родился в 1988 году в г.Тбилиси. Закончил Санкт-петербургский государственный университет, математико-механический факультет, кафедра информатики.
Квалификация: Математик-программист
Специальность: Математическое обеспечение и администрирование информационных систем.
Научные интересы: Компьютерное зрение, методы оптимизации.

Основные достижения:

- Многократный участник и победитель олимпиад по математике и физике (в Грузии).
- Член сборной Грузии по математике и участник международных математических олимпиад 2004(Афины, бронзовая медаль) и 2005(Мерида, бронзовая медаль).
- Участник многих олимпиад по информатике в России.
- Активный участник соревнований по программированию на сайте “Top-Coder” (таргет в рейтинге марафонских соревнований)

Задачи и разборы

Задача A. Игра с числами

Имя входного файла:	`a.in`
Имя выходного файла:	`a.out`
Ограничение по времени:	1 с
Ограничение по памяти:	64 Мб

Анна и Бека выдумали следующую игру. Для начала, они выбирают натуральное число K, а потом по очереди называют числа в диапазоне $[1, K]$. Игру всегда начинает Анна с числа 1. Если на некотором ходу было названо число X, тогда в ответ обязательно должно быть названо или число $X + 1$, или число $2X$ - при условии, что соответствующее число не превосходит K. Победителем становится тот из игроков, кто первым называет число K.

Известно, что и Анна, и Бека играют оптимально. Они собираются сыграть для всех K в диапазоне $[L, R]$. Найдите, сколько игр закончится победой Анны.

Ограничения

$1 \leq L \leq R \leq 10^{18}$

Формат входного файла

Единственная строка входного файла содержит пару чисел L и R.

Формат выходного файла

Выведите количество игр, которые закончатся победой Анны.

Пример

a.in	a.out
1 7	6

Примечание

Из предстоящих семи игр, Бека выиграет только при $K = 2$.

Разбор задачи A. Игра с числами

В первую очередь докажем, что игра является проигрышной для первого игрока тогда и только тогда, когда K представимо в виде суммы нечетных степеней двойки. Для фиксированного K будем называть число Z из диапазона $[1, K]$ выигрышным, если игрок, которому предстоит делать ход после того, как было названо Z, имеет выигрышную стратегию, и проигрышным в противном случае. Обозначим $x_0 = K$, $x_{i+1} = [x_i/2]$ для неотрицательных i и выберем N такое, что $X_N = 1$. Рассмотрим множества $A_i = x_{i+1} + 1, \ldots, x_i$. Назовём множество A_i выигрышным, если все числа в нем выигрышные, четно-выигрышным, если только четные числа в нем выигрышные и нечетно-выигрышным в противном случае. Допустим, A_i четно-выигрышное и рассмотрим A_{i+1}. Умножив любое число из A_{i+1} на два, мы получим четное число из A_i, которое выигрышно для противника и, соответственно, проигрышно для ходящего. Поэтому Z из A_{i+1} выигрышно только если $(Z + 1)$ проигрышно, соответственно A_{i+1} так-же четно-выигрышное. Если A_i нечетно-выигрышное, тогда любое число из A_{i+1} при умножении на два приводит в проигрышную позицию - соответственно, A_{i+1} выигрышное. Рассмотрим выигрышное A_i. Умножать на два числа из A_{i+1} - проигрышный ход, поэтому Z из A_{i+1} выигрышно тогда и

только тогда, когда $(Z+1)$ проигрышно. Так как x_{i+1} проигрышно, A_{i+1} нечетно-выигрышно, если x_{i+1} четно, и четно-выигрышно, если нет. Соответственно, A_0 четно-выигрышно для нечетного K и нечетно-выигрышно для четного. Также, если четно-выигрышно некоторое A_i, тогда четно-выигрышны все $A_{i+1}, A_{i+2}, \ldots$ и число 1 проигрышно. Второй игрок имеет победную стратегию только если множества $A_0, A_1, \ldots$ поочередно нечетно-выигрышные и выигрышные (при этом A_0 будет нечетно-выигрышным), что эквивалентно условию, что все $x_0, x_2, x_4, \ldots$ четны. Значит, второй игрок побеждает тогда и только тогда, когда в двоичном представлении числа K на нечетных позициях записаны нули. Теперь нужно научиться считать числа, представимые в виде суммы нечетных степеней двойки и находящиеся в диапазоне $[L, R]$. Для начала воспользуемся стандартным приёмом: представим ответ в диапазоне $[L, R]$ как разницу ответов в диапазоне $[1, R]$ и $[1, L-1]$. Таким образом, умея считать числа в $[1, X]$, можно решать и эту задачу. Можно построить решение за $O(logX)$ на основании следующей идеи: рассмотрим максимальную нечетную степень B такую, что $2^B \leq X$. Количество ответов, в которых она не участвует, равно $2^{B/2} - 1$, а в которых участвует - ответу задачи на $[0, X - 2^B]$. От добавления 0 в рассматриваемый диапазон, логика не сильно меняется: просто количество ответов в первом случае будет $2^{B/2}$. Таким образом, за $O(1)$ задача сводится от X к как минимум $X/2$, то есть сложность всего алгоритма - $O(logR)$.

Задача B. K-вложенные отрезки

Имя входного файла:	`b.in`
Имя выходного файла:	`b.out`
Ограничение по времени:	2 с; для программ на Java - 3 с
Ограничение по памяти:	64 Мб

Назовём неупорядоченную пару отрезков $([a_1, b_1], [a_2, b_2])$ K-вложенными, если найдётся такая пара чисел (X, Y), что $X + Y \leq K$ и выполняется хотя бы одно из следующих условий:

a) Отрезок $[a_1 - X, b_1 + Y]$ содержит отрезок $[a_2, b_2]$, т.е. $a_1 - X \leq a_2 \leq b_2 \leq b_1 + Y$;

b) Отрезок $[a_2 - X, b_2 + Y]$ содержит отрезок $[a_1, b_1]$, т.е. $a_2 - X \leq a_1 \leq b_1 \leq b_2 + Y$.

Заметим, что при $K = 0$ мы имеем дело с обычным определением вложенности. Для заданных N отрезков найдите, при скольких целых неотрицательных значениях K среди этих отрезков есть ровно M пар K-вложенных отрезков.

Ограничения

$2 \le N \le 20\,000$
$0 \le M \le N * (N-1)/2$
Координаты концов отрезков - целые числа в диапазоне $[-10^9, 10^9]$. Отрезки невырождены.

Формат входного файла

Первая строка входного файла содержит количество отрезков N и желаемое количество K-вложенных пар M. Каждая из следующих N строк содержит по паре чисел - соответственно начала и концы заданных отрезков.

Формат выходного файла

Выведите количество целых неотрицательных значений K, при которых выполняется условие задачи. Если таких значений бесконечно много, выведите -1.

Пример

b.in	b.out
3 1 -2 2 -3 1 2 4	1

Примечание

При $K = 1$ второй отрезок можно удлинить так, чтобы он покрыл первый. Если увеличить K до двух, первый отрезок сможет покрыть третий, поэтому больше подходящих значений K не существует.

Разбор задачи B. K-вложенные отрезки

Научимся быстро считать количество K-вложенных пар для конкретного K. Такие пары можно поделить на две категории: изначально вложенные пары и те, в которых один из отрезков нужно удлинять.

Подсчёт вложенных пар - классическая задача на структуры данных. Упорядочим отрезки по координате левого конца, а при их равенстве - по координате правого. Начнём рассматривать их в этом порядке. Тогда количество отрезков, в которые вложен i-ый - это количество рассмотренных до него отрезков с большей координатой правого конца. Чтобы быстро определять такие количества (а затем и изменять их), мы можем воспользоваться деревом отрезков, битово-индексированным деревом или чем-либо ещё.

Дерево будет строиться на массиве, в котором на i-ом индексе записано, сколько отрезков имеют координату правого конца равную i. Координаты в задаче большие, но на самом деле нас интересует не точная координата, а её очередность относительно остальных координат правых концов отрезков, которых не более чем N - а значит можно координаты сжать до чисел от 1 до N. Для остальных пар выполняется, что один из отрезков нужно удлинять или только вправо, или только влево. Упорядочим отрезки по возрастанию длины и будем их рассматривать по очереди. Для рассматриваемого отрезка $[A_i, B_i]$ количество отрезков, которые отрезок i покроет при удлинении влево - это количество уже рассмотренных отрезков с координатой левого конца в диапазоне $[A_i - K, A_i - 1]$. Аналогично, при удлинении вправо i-ый отрезок покроет все отрезки с координатой правого конца в диапазоне $[B_i+1, B_i+K]$. Вследствие упорядочения по возрастанию длины, мы не подсчитаем одну и ту же пару дважды. Считать эти данные можно снова с помощью деревьев. Если использовать дерево отрезков, тогда опять-таки придётся сжать координаты, поэтому нужно будет также уметь быстро считать, какому интервалу соответствует $[A_i - K, A_i - 1]$ и $[B_i + 1, B_i + K]$, что можно делать с помощью двоичного поиска в заранее упорядоченных списках координат левых и правых концов.

Изначальную задачу теперь можно решать с помощью двоичного поиска. Очевидно, что при увеличении K количество пар не может уменьшиться, поэтому можно найти максимальное K, при котором пар ровно M и максимальное K, при котором их меньше - разница и будет ответом. Бесконечное количество K будет лишь в том случае, когда все пары отрезков должны быть K-вложенными. Итоговая сложность алгоритма - $O(logKNlogN)$.

Задача C. Камикадзе

Имя входного файла:	`c.in`
Имя выходного файла:	`c.out`
Ограничение по времени:	1 с
Ограничение по памяти:	64 Мб

Вася уже давно играет на Фейсбуке в очень известную игру “Kamikadze Race”. Он считает, что авторы игры поступили нечестно, и какой бы он маршрут не выбрал, рано или поздно он обязательно столкнется с одной из машин.

В этой игре на трассе фиксированной ширины T метров с постоянной скоростью движется N машин (не считая Васиной), имеющих прямоугольную форму. Чтобы четко оценить ситуацию, Вася построил декартову си-

стему координат и направил ось X вдоль дороги. Точка $(0, 0)$ находится на её левой границе, а точка $(0, T)$ - на правой.

Каждая из машин позиционируется параллельно границам трассы, а их начальное положение задаётся координатами левых нижних углов (точек с минимальными координатами). Для машины Васи это точка $(0, S)$, а для i-ой из остальных машин это точка $(x[i], y[i])$. Габариты Васиной машины - длина L и ширина W метров, где длина отмерена вдоль оси X. В свою очередь, i-ая из остальных машин имеет длину $l[i]$ и ширину $w[i]$ метров.

Скорость всех машин направлена вдоль оси X, i-ая из них движется со скоростью $v[i]$. Скорость Васиной машины V, но он, в отличие от остальных, также может поворачивать. При этом скорость машины вдоль оси X остаётся неизменной, а скорость вдоль оси Y не может превышать по модулю заданной величины HV. Изменять скорость машина Васи способна мгновенно.

Машина Васи может касаться другой машины, но при пересечении их контуров происходит взрыв. Остальные машины на трассе могут соприкасаться и пересекаться, при этом совершенно ничего не происходит, и они продолжают движение, как и раньше. Выходить за границы трассы Васина машина, естественно, не может. Помогите ему найти максимальное время, на протяжении которого он может ехать без столкновений.

Ограничения

$1 \le N, T, L, W, V, HV \le 1000$
$0 \le S \le T - W$
$1 \le v[i], l[i], w[i] \le 1000$ для каждого $1 \le i \le N$.
$0 \le y[i] \le T - w[i]$ для каждого $1 \le i \le N$.
$-1000 \le x[i] \le 1000$ для каждого $1 \le i \le N$.
Все числа во входных данных целые.

Формат входного файла

Первая строка входного файла содержит единственное число T - ширину дороги. Вторая строка содержит числа S, L, W, V, HV - изначальная Y-координата машини Васи, ее длина, ширина, скорость вдоль оси X и максимальная скорость вдоль оси Y. Следующая строка содержит число N - количество машин; i-ая из следующих N строк содержит числа $x[i], y[i], l[i], w[i], v[i]$ - координаты остальных машин, их габариты и их скорости.

Формат выходного файла

Выведите максимальное время, которое Васина машина может лавировать на трассе без столкновений, или "-1", если она может увернуться от

всех машин. Ваш ответ не должен отличаться от правильного более чем на 10^{-6}.

Пример

c.in	c.out
100 50 10 10 2 1 2 100 0 1 90 1 200 10 1 90 1	190.0

Разбор задачи C. Камикадзе

Сразу заметим, что в этой задаче важны только относительные скорости машин, поэтому будем считать, что X-координата машины Васи всегда 0, и он может передвигаться только налево и направо. Мы можем узнать координаты любой машины в любой момент времени и, следовательно, за $O(N)$ найдем множество точек, в которых должен находится Вася, чтобы ни с кем не столкнуться. Очевидно, что это множество является объединением непересекающихся отрезков. Обозначим его через $P(t)$ для конкретного момента времени t. Оно меняется только в тех точках, когда какая-нибудь машина занимает участок на оси X или наоборот - освобождает его. Для начала найдем все такие моменты времени и упорядочим их по возрастанию. Пусть получилась последовательность $t[1], \dots t[k]$. Далее мы будем хранить множество точек $F(t)$, в которых может находится Вася с учетом того, что до этого момента он ни с кем не столкнулся. Очевидно $F(t)$ является подмножеством $P(t)$. Мы будем вычислять F в точках $t[1], \dots t[k]$. Если в какой то момент $t[i]$ оно окажется пустым, значит $t[i]$ и есть ответ задачи. В том случае когда $F(t[k]) \neq \varnothing$ ответом является -1.

В начале $F(0)$ представляет собой одну точку. Предположим, мы знаем $F(t[i])$ для некоторого $i < k$. На отрезке $[t[i], t[i+1])$ каждый отрезок $F(t[i])$ постепенно расширяется, поскольку Вася может двигаться горизонтально, но не выходит за границы соответствующего отрезка $P(t[i])$ (отрезка, который содержит его), обозначим расширенное множество через A. В момент $t[i+1]$ возможно сокращение функции P, поэтому $F(t[i+1])$ = пересечению A и $P(t[i+1])$. Если предварительно упорядочить машины, на нахождение этих пересечений требуется $O(N)$ операций, поэтому общая сложность алгоритма - $O(N^2)$.

Задача D. Бескорневые пары

Имя входного файла:	`d.in`
Имя выходного файла:	`d.out`
Ограничение по времени:	1 с
Ограничение по памяти:	64 Мб

Будем называть пару чисел бескорневой для данного простого числа P, если не существует такого X, что X^2 и произведение данной пары чисел имеют одинаковый остаток при делении на P. Дано N чисел A_1, A_2, ..., A_N. Посчитайте, сколько среди них бескорневых пар.

Ограничения

$2 \le N \le 10\,000$
$2 \le P \le 10^9$, P - простое число
$0 \le A_i \le 10^9$ для каждого $1 \le i \le N$

Формат входного файла

Первая строка входного файла содержит единственное число P. Вторая строка содержит число N, а в третьей строке через пробел записаны N целых чисел - A_1, A_2, ..., A_N.

Формат выходного файла

Выведите единственное число - количество бескорневых пар среди чисел A_i.

Пример

d.in	d.out
5 3 2 4 4	2

Примечание

Из возможных трех пар две пары $\{2, 4\}$ бескорневые, а для $\{4, 4\}$ можно взять $X = 1$.

Разбор задачи D. Бескорневые пары

Для начала давайте вспомним некоторые определения из теории чисел. Пусть дано простое число P. Число $X(0 \le X < P)$ называется квадратичным остатком, если существует такое A, что $A^2 = X(modP)$. Символ

Лежандра определяется следующим образом: $(X/P) = \{1$, если X кв. остаток и $X \neq 0$; -1, если X некв. остаток; 0, если $X = 0\}$.

Утверждается, что $(A * B/P) = (A/P) * (B/P)$. Когда один из множителей равен нулю, равенство очевидно. Покажем его правильность в остальных случаях:

1) $(A/P) = (B/P) = 1$. Следовательно, существуют такие X и Y, что $A = X^2(modP)$ и $B = Y^2(modP)$. Тогда $AB = X^2 * Y^2 = (X * Y)^2(modP)$, соответственно $(A * B/P) =1$.

2) Ровно один из множителей равен 1. Без потери общности, возьмём $(A/P) = 1$, $(B/P) = -1$. Допустим, что $(AB/P) = 1$. Тогда должно существовать X такое, что $A * B = X^2(modP)$. Также существует такое C, что $A = C^2(modP)$. Возьмём обратное числу C по модулю P число и назовём его R. Получаем: $C^2 * B = X^2(modP) \Rightarrow C^2 * R^2 * B = X^2 * R^2(modP) \Rightarrow B = (X * R)^2(modP)$, то есть B является квадратичным вычетом, а это противоречит условию. Следовательно, $(AB/P) = -1$.

3) Для начала покажем, что если $(A/P) = -1$, а $S_1, S_2, \ldots, S_{(P-1)/2}$ - все квадратичные остатки, тогда множество $T = A * S_i : 1 \leq i \leq (P-1)/2$ дополняет S до $1, 2, \ldots, P-1$ и у S и T нет общих элементов. Последнее утверждение напрямую следует из предыдущего пункта, поэтому остаётся показать, что T содержит ровно $(P-1)/2$ элементов. Это следует из того, что если $i \neq j$, $A * S_i$ и $A * S_j$ имеют различный остаток по модулю P - иначе $A * (S_i - S_j)$ должно делиться на P, но ни A, ни данная разница на P делиться не могут. Покажем, что если $(A/P) = (B/P) = -1$, то $(AB/P) = 1$. Раз A и B не входят в S, они являются элементами T. То есть существует такое i, что $B = A * S_i(modP)$. Также существует такое X, что $S_i = X^2(modP)$. Поэтому, $A * B = A^2 * S_i = A^2 * X^2 = (A * X)^2(modP)$, то есть $A * B$ является квадратичным остатком. Из доказанного равенства следует, что произведение двух чисел является неквадратичным остатком тогда и только тогда, когда один из сомножителей является ненулевым квадратичным остатком, а другой - неквадратичным. Иными словами, либо $(A/P) = -1$ и $(B/P) = 1$, либо $(A/P) = 1$, $(B/P) = -1$. Остается вычислить символ Лежандра на заданных числах и посчитать количество положительных и отрицательных значений. Произведение этих величин и будет нашим ответом. Существует не один метод, позволяющий вычислить символа Лежандра за время $O(logP)$, но, пожалуй, самый простой из них основан на следующей формуле: $(X/P) = X^{(P-1)/2}$.

Задача E. Стратегия

Имя входного файла:	`e.in`
Имя выходного файла:	`e.out`
Ограничение по времени:	2 с
Ограничение по памяти:	64 Мб

Рассмотрим упрощенную схему битвы одной из культовых стратегических игр. Битва происходит между двумя армиями, в каждой из которых не более семи юнитов (боевых единиц). Каждый юнит принадлежит к одному из N видов существ, обитающих в игровой вселенной. Каждое существо характеризуется тремя показателями: атакой, защитой и скоростью, которые измеряются в целых числах.

Битва происходит в дискретном режиме: на каждом шагу, один из юнитов одной из армий делает свой ход. Первым делает ход юнит с максимальной скоростью среди всех юнитов на поле битвы. Если таких несколько, они ходят по очереди в случайном порядке. Далее делает ход юнит с максимальной скоростью среди всех выживших ещё не ходивших юнитов, если таковых несколько - очередность снова устанавливается в случайном порядке. Когда таким образом сделают ход все выжившие юниты обеих армий, ход снова отдаётся самому быстрому юниту, и так далее до тех пор, пока не будет полностью уничтожена одна из армий.

Ход юнита заключается в уничтожении одного из юнитов противника. Будем считать, что юнит A при нападении на юнит B уничтожает противника тогда и только тогда, когда показатель атаки юнита A выше или равен показателю защиты юнита B.

Когда главнокомандующий одной из армий видит, что проигрывает, он может сбежать с поля битвы. Он может это сделать только когда очередь ходить одного из его юнитов.

Вы собираетесь напасть на армию соперника, состоящую из M заданных юнитов. Её главнокомандующий владеет очень ценным трофеем, поэтому нельзя позволить ему сбежать. Очевидно, что гарантировать это можно лишь не дав ходить ни одному его юниту. Найдите, сколькими способами можно выбрать M юнитов для своей армии так, чтобы одолеть противника, не оставив ему возможности сбежать. Два выбора отличаются друг от друга, если в них встречается различное количество юнитов хотя бы одного вида.

Ограничения

$1 \leq N \leq 400$
$1 \leq M \leq 7$

Показатели атаки, защиты и скорости - целые числа в диапазоне $[1, 100]$.

Формат входного файла

Первая строка входного файла содержит количество различных видов существ N и количество юнитов в армиях M. Каждая из следующих N строк содержит по три числа - соответственно показатели атаки, защиты и скорости очередного существа. В последней строке дано M чисел в диапазоне от 1 до N; i-ое из этих чисел - номер вида i-го юнита противника. Виды пронумерованы в том порядке, в котором они заданы во входных данных.

Формат выходного файла

Выведите количество различных вариантов укомплектовать армию так, чтобы одолеть соперника и не дать сбежать его главнокомандующему.

Пример

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

Примечание

Следующие четыре армии удовлетворяют поставленным условиям: $\{1, 1\}, \{1, 2\}, \{1, 4\}, \{2, 4\}$.

Разбор задачи E. Стратегия

Попробуем решать задачу полным перебором вариантов. Всего кандидатов на ответ $C(N+M-1, M)$ штук, рассмотрение всех в разумное время значительно превосходит возможности современного компьютера. Тем не менее, это число можно значительно уменьшить за счёт того, что многие варианты аналогичны.

В первую очередь введем следующие обозначения: пусть показатели атаки, защиты и скорости i-ого типа существ будут A_i, D_i и S_i. Если для двух существ X и Y выполняется $A_X \geq D_Y$ и $S_X \geq S_Y$, будем говорить, что юнит типа X побеждает юнит типа Y. Для каждого типа существ найдём множество юнитов армии противника, которых оно побеждает. Теперь

сравним все эти множества. В худшем случае (когда $M = 7$ и для юнитов противника выполняется, что чем они быстрее, тем ниже их показатель защиты) различных множеств получится всего 29. Очевидно, что если в конкретной армии заменить существо на любое другое с тем же множеством побеждаемых противников, исход битвы не изменится. Поэтому все существа делятся на не более чем 29 групп с одинаковыми свойствами.

Будем перебирать варианты следующим образом: на i-ом шагу будем выбирать из 29 групп существ такую группу, существа которой побеждают i-го юнита противника. Таких вариантов всего около 25 миллионов. Но некоторые варианты могут повториться (например, если сначала выбрать для первого юнита группу 1, для второго группу 2, а затем наоборот - такой вариант возможен при наличие юнита 3, которого существа группы 1 побеждают, а существа группы 2 нет). Чтобы не считать их дважды, будем запоминать уже встретившиеся варианты. Вариант можно описать с помощью одного 64-битового числа следующим образом: рассмотрим систему счисления, в которой i-ая цифра может принимать значения от 0 до мощности множества побеждаемых i-ой группой юнитов. Так как произведения увеличенных на 1 мощностей множеств помещаются в диапазон значений 64-битового числа, можно сопоставить варианту число, в котором на i-ой позиции будем записано, сколько юнитов из i-ой группы выбрано в армию. Сами варианты можно хранить в хэш-таблице. Для ускорения перебора можно сделать не одну, а M таблиц, в i-ой из которых будут записаны полученные на i-ом шагу варианты - тогда, получив на некотором шагу уже встречавшийся ранее вариант, эту ветвь перебора можно будет не рассматривать.

Осталось пояснить, как считать количество вариантов, соответствующих конкретному распределению групп. Если в нашей армии будет K юнитов из некоторой группы, в которой всего P существ, тогда выбрать их можно $C(P + K - 1, K)$ способами. Количество вариантов для конкретного распределения - произведение этих величин по всем группам.

День девятый (26.02.2012 г.)
Контест Пака Станислава Олеговича

Об авторе...

Пак Станислав Олегович, родился в 1986 году в городе Янгиюль республики Узбекистан. Учился в лингвистической гимназии №8 города Энгельса Саратовской области. С 8 класса участвовал в олимпиадах по физике, химии, математике, информатике, литературе. Закончил механико-математический факультет Саратовского государственного университета с красным дипломом, учится в аспирантуре СГУ по специальности 01.01.01 "Математический анализ". Разработчик компании Яндекс. Интересуется психологией.

Основные достижения:

- первое место на областной олимпиаде по математике (Саратов, 2004);
- второе место в Южном четвертьфинале NEERC 2006 и 2007 года;
- чемпион России NEERC ACM 2008;
- золотая медаль ACM ICPC World Finals 2009 (Стокгольм).

Теоретический материал. Комбинаторная геометрия

Метод вращающихся калиперов Метод вращающихся калиперов — очень распространенный метод перебора граничных состояний. Здесь будет рассматриваться его применение в геометрических задачах. Рассмотрим два множества точек в R^n A и B. Точки можно рассматривать как вектора, тогда можно определить сумму этих множеств как множество $\{a + b | a \in A, b \in B\}$, сумма векторов определяется по правилу параллелограмма геометрически, алгебраически это просто сумма соответствующих координат. Если A и B связны и выпуклы, то их сумма также связна и выпукла. Например, выпуклость сразу следует из тождества

$(a_1 + b_1)\lambda + (a_2 + b_2)(1 - \lambda) = a_1\lambda + a_2(1 - \lambda) + b_1\lambda + b_2(1 - \lambda)$, где $a_1, a_2 \in A, b_1, b_2 \in B, \lambda \in (0, 1)$. В частности, сумма двух выпуклых многоугольников есть выпуклое множество.

На самом деле справедливы следующие утверждения, относительно равенства $C = A + B$, где A и B — выпуклые многоугольники.

- C — выпуклый многоугольник.
- Число вершин в C не превосходит $2n$.
- Вершины C представляются как сумма некоторых вершин A и B.

Эти наблюдения уже достаточны, чтобы сформулировать полиномиальный алгоритм. Нужно найти выпуклую оболочку попарных сумм вершин многоугольников. Сложность алгоритма $O(n^2 \log(n))$, если строить выпуклую оболочку за $O(n \log(n))$.

Можно строить сумму гораздо быстрее, если заметить следующий факт. Для этого введем понятие одноположных вершин. Это такие вершины двух многоугольников, что через них можно провести параллельные прямые, оставляющие соответствующие многоугольники по одну и ту же сторону. Можно показать, что сумма одноположных вершин не может быть внутренней точкой. С другой стороны, сумма не одноположных вершин a и b обязательно будет внутренней, потому что для любого вектора достаточно малой длины v имеет место $a + v \in A$ или $b + v \in B$. Это как раз означает, что $a + b$ — внутренняя точка.

Пара самых левых нижних точек очевидно является одноположной. Ориентируем вершины многоугольников против часовой стрелки. Пусть pa и pb — одноположные вершины. Рассмотрим следующие за ними вершины соответственно pan и pbn. Так как pa и pb одноположны, то можно провести прямую через $pa + pb$ такую, что pan и pbn будут лежать по одну сторону. Можно рассматривать углы между векторами $\overline{pa, pan}$, $\overline{pb, pbn}$ и направлением прямой, которая оставляет многоугольники слева. Обозначим такие углы α и β. Пусть $\alpha < \beta$, тогда нетрудно проверить, что пара pb и pan не будет одноположной. Тогда следующая вершина суммы есть $pa + pbn$. Если для углов выполняется равенство, то следующая вершина суммы — это $pan + pbn$, потому что $pa + pbn$ и $pb + pan$ будут лежать на стороне. Конечно, явно вычислять углы не требуется, можно обойтись векторным произведением.

По другому этот метод можно назвать методом бегущих указателей, методом двух (нескольких) указателей, которые «бегут» сообща. Конечно, его применение выходит за рамки комбинаторной геометрии.

Другая не менее классическая задача, которая эффективно решается методом калиперов — это задача о диаметре дискретного множества точек.

Диаметры множества точек и их выпуклой оболочки совпадают, потому что любые две внутренние точки не дают максимального расстояния. Точки, дающие максимум, не могут лежать строго на стороне, они обязательно совпадают с вершинами выпуклой оболочки по теореме косинусов, например. Тем не менее, число вершин в выпуклой оболочке может быть большим, перебирать пары вершин выпуклой оболочки нельзя.

Рассмотрим пару различных вершин a и b выпуклой оболочки P. Проведем через них прямые ортогонально отрезку ab, они задают бесконечную полосу. Если все точки P лежат между этими прямыми (или на них, то есть на полосе), то назовем эту пару вершин противоположными. Ясно, что максимум расстояния достигается на противоположных точках, потому что в противном случае ответ можно улучшить, опять же по теореме косинусов. Оказывается, пар противоположных вершин линейное от размера P число, их можно все перебрать. Зафиксируем порядок вершин P, например, против часовой стрелки. Зафиксируем вершину p. Пусть предыдущая pp, следующая pn. Найдем первую по порядку после вершины p вершину ps такую, что она максимально удалена от прямой $\overline{pp, p}$. В силу монотонности следующая после ps вершина находится не дальше. Также найдем последнюю после p вершину pe, которая максимально удалена от прямой $\overline{p, pn}$. Все вершины, которые образуют противоположную пару с p лежат между ps и pe включительно, потому другие вершины вместе в p порождают полосу, не содержащую либо pp, либо pn. При переходе от рассмотрения вершины p к рассмотрению вершины pn, аналогичные вершины ps и pe могут сдвинуться только дальше против часовой стрелки. Понятно, что если в выпуклой оболочке нет трех подряд идущих коллинеарных точек (лежащих на одной прямой), то следующая вершина ps может продолжаться либо от pe, либо от вершины, непосредственно предшествующей pe. Таким образом, суммарно будет рассмотрено не более $O(|P|)$ пар вершин, где $|P|$ — число вершин в многоугольнике P.

Метод вращающихся калиперов также применим в задачах поиска выпуклого k-угольника наименьшей площади или периметра, накрывающего выпуклый n-угольник. Чтобы этот метод стал применим, докажем один замечательный факт.

Сторона выпуклого k-угольника наименьшего периметра (площади) Q, накрывающего выпуклый n-угольник P, перекрывается с его стороной.

Предположим, что стороны P и Q не перекрываются. Тогда на сторонах Q $q_1, ..., q_k$ лежит ровно по одной вершине P, пусть это вершины $p_1, \cdots, p_k$. Пусть $\alpha_i = angle(q_{i-1}, q_i)$, где $angle(a, b)$ — угол между сторонами a и b, s_i — отрезок $\overline{p_{i-1}, p_i}$, а a_i — его длина. Пусть также $\beta_i = angle(q_i, s_i)$, а $\gamma_i = angle(q_{i-1}, s_i)$. Тогда по теореме синусов для периметра многоугольника Q

имеет место равенство

$$Pr(0) = \sum_{i=1}^{k} \frac{a_i}{\sin(\alpha_i)}(\sin(\beta_i) + \sin(\gamma_i)) = \sum_{i=1}^{k} \frac{a_i}{\cos(\alpha/2)} \cos(\frac{\beta_i - \gamma_i}{2}).$$

k-угольник можно повернуть на угол θ (он может менять форму и объем, но сохраняет углы α_i). Пусть $[\theta_{min}, \theta_{max}]$ — отрезок, внутри которого поворот не приводит к перекрытию сторон многоугольников. Для фиксированного поворота на угол $\theta \in [\theta_{min}, \theta_{max}]$ периметр

$$Pr(\theta) = \sum_{i=1}^{k} \frac{a_i}{\cos(\alpha/2)} \cos(\frac{\gamma_i - \beta_i - 2\theta}{2}),$$

а $\frac{\gamma_i - \beta_i - 2\theta}{2} \in (-\pi/2, \pi/2)$, поэтому функция периметра вогнута как линейная комбинация вогнутых функций. Отсюда, минимум периметра достигается на концах отрезка.

Для площади справедливы аналогичные рассуждения.

Задачи и разборы

Задача A. Asteroids collision

Имя входного файла:	`a.in`
Имя выходного файла:	`a.out`
Ограничение по времени:	1 с
Ограничение по памяти:	256 Мб

Даны два выпуклых многоугольника с числом вершин N и M, соответственно, а также два вектора, задающие их движение. Длина вектора определяет скорость в единицах длины в секунду. Ваша задача — выяснить, столкнутся ли эти многоугольники.

Ограничения

$3 \leq N, M \leq 40000$. Все координаты не превосходят по модулю 10^8.

Формат входного файла

В первой строке содержится число вершин в первом многоугольнике N. В следующих N строках содержатся координаты вершин. В следующей строке находятся координаты вектора скорости. Далее идет аналогичная информация о втором многоугольнике.

Формат выходного файла

Если ответ отрицательный, выведите «No collision», в противном случае выведите время столкновения (касание считается столкновением) как несократимую дробь в виде x/y. Движение начинается в момент времени 0.

Пример

a.in	a.out
4 0 0 1 0 1 1 0 1 0 0 3 2 0 3 0 3 1 -1 0	1/1
4 0 0 1 0 1 1 0 1 0 0 3 2 0 3 0 3 1 0 1	No collision

Разбор задачи A. Asteroids collision

Пусть P, Q — многоугольники, которые движутся со скоростями vp и vq. Зафиксируем некоторый момент $t \geq 0$, как узнать, что многоугольники $P(t) := P + vp \cdot t$ и $Q(t) := Q + vq \cdot t$ пересекаются? Отразим второй многоугольник относительно начала координат в $Q'(t)$ и найдем прямую сумму (или сумму Минковского) $P(t)$ и $Q'(t)$. Многоугольники $P(t)$ и $Q(t)$ имеют общую точку тогда и только тогда, когда их сумма содержит точку $(0, 0)$. Можно легко показать, что сумма Минковского для выпуклых множеств

выпукла и сдвигается на вектор, если аргумент суммы сдвигается на вектор. Таким образом, сумма $P(t)$ и $Q'(t)$ есть сумма P и Q' (центрально симметрично отраженный относительно начало координат), сдвинутая на вектор $\overline{vp, vq}$, сумма движется вдоль вектора $\overline{vp, vq}$, или можно думать, что точка $(0, 0)$ движется вдоль вектора $\overline{vq, vp}$. Нужно найти момент, когда луч из начала координат, распространяющийся со скоростью $\overline{vq, vp}$, пересечет сумму P и Q'.

Задача B. Most distant points pair

Имя входного файла:	`b.in`
Имя выходного файла:	`b.out`
Ограничение по времени:	1 с
Ограничение по памяти:	256 Мб

Найдите квадрат диаметра множества точек на плоскости.

Ограничения

$1 \leq N \leq 40000$ Все координаты не превосходят по модулю 10^8. Все числа целые.

Формат входного файла

В первой строке содержится число точек N. В следующих N строках содержатся их координаты.

Формат выходного файла

Выведите одно целое число — квадрат расстояния между парой точек, наиболее удаленных друг от друга.

Пример

b.in	b.out
4 0 0 10 0 10 10 5 6	200

Разбор задачи B. Most distant points pair

В данную задачу можно решать методом указателей, бегущих по вершинам выпуклого многоугольника, как рассказано в лекции. Чтобы сравнить вершины c и d по расстоянию до прямой, проходящей через сторону

$\overline{a,b}$, достаточно сравнить их векторные произведения $\overline{a,b} \times \overline{a,c}$ и $\overline{a,b} \times \overline{a,d}$, потому что они выражают площадь параллелограмма, натянутого на вектора, и, с другой стороны, площадь равна произведению высоты (расстояния) на фиксированное основание $\overline{a,b}$.

Задача C. Minimum area bounding box

Имя входного файла:	`c.in`
Имя выходного файла:	`c.out`
Ограничение по времени:	1 с
Ограничение по памяти:	256 Мб

Найдите минимальную площадь прямоугольника, содержащего данные точки.

Ограничения

$1 \leq N \leq 40000$. Все координаты не превосходят по модулю 10^8. Все числа целые.

Формат входного файла

В первой строке содержится число точек N. В следующих N строках содержатся их координаты.

Формат выходного файла

Выведите одно целое число — площадь прямоугольника как несократимую дробь в виде x/y.

Пример

c.in	c.out
4 0 0 10 0 10 10 5 6	100/1

Разбор задачи C. Minimum area bounding box

Можно заметить и показать, что накрывающий прямоугольник минимальной площади на одной из сторон имеет две точки множества. Таким образом, после построения выпуклой оболочки достаточно перебрать ее

сторону $\overline{sa, sb}$ (sa, sb — вершины) и достроить остальные стороны прямоугольника. Ориентируем стороны, например, против часовой стрелки. Противоположная сторона будет проходить через наиболее удаленную от $\overline{sa, sb}$ точку p. Если сторона будет меняться последовательно против часовой стрелки, то противоположная точка p тоже будет меняться последовательно против часовой стрелки.

Чтобы найти боковые стороны прямоугольника нужно найти стороны выпуклой оболочки, образующие с вектором $\overline{n}$ наименьший угол, где $\overline{n}$ — вектор, ортогональный $\overline{sa, sb}$.

Как можно сделать все в целых числах? Пусть две боковые противоположные вершины многоугольника это a и b такие, что вершины a, p, b расположены в порядке против часовой стрелки. Тогда длина накрывающего прямоугольника — это $(\overline{sa, sb} \times \overline{sa, b})/dist(sa, sb)$, ширина — это $((\overline{n} \times \overline{sa, b}) - (\overline{n} \times \overline{sa, a}))/dist(sa, sb)$, где $dist(a, b)$ — длина вектора $\overline{a, b}$. Видно, что площадь прямоугольника — представляется как отношение целых чисел. Произведение числителей не помещается в 64-битный целый тип, поэтому нужно сократить знаменатель с обоими числителями и умножить два длинных числа.

Задача D. Burn after reading

Имя входного файла:	`d.in`
Имя выходного файла:	`d.out`
Ограничение по времени:	1 с
Ограничение по памяти:	256 Мб

Агент Вася 00* Пупкин снова обнаружил в своем почтовом ящике треугольное письмо. Послание из Центра. Для соблюдения секретности все послания из Центра снабжаются механизмом самоуничтожения, и Вася знает, что как только он вскроет конверт, специальное вещество, нанесенное на все три вершины треугольного послания, прореагирует с воздухом и воспламенится. Пламя распространяется по посланию равномерно во все стороны от очагов возгорания с постоянной скоростью 1 [см/сек]. Вася знает, что обычно самая полезная часть письма сгорает самой последней. Теперь он хочет узнать, сколько времени у него будет на прочтение письма, и как много он упустит, если пойдет сварить себе чашечку кофе.

Ограничения

Все координаты не превосходят по модулю 10^5. Все числа целые. Три вершины не лежат на одной прямой.

Формат входного файла

Входные данные содержат три строки. На каждой строке содержится пара чисел — координаты вершины треугольника.

Формат выходного файла

В единственную строку выходных данных вывести одно вещественное число минимальное время за которое треугольник сгорит полностью. Ответ необходимо выводить с не менее чем шестью знаками после десятичной точки

Пример

d.in	d.out
0 0 3 0 3 4	2.5000000000

Разбор задачи D. Burn after reading

Какая точка треугольника сгорит последней? Рассмотрим центр описанной окружности. Если он лежит внутри или на границе треугольника (это означает, что треугольник не тупоугольный), то это та самая точка. В противном случае эта точка есть вершина равнобедренного треугольника с основанием, совпадающим с большей стороной тупого угла, лежащая на самой большой стороне, то есть по теореме синусов напротив тупого угла.

Задача E. Clear after burning

Имя входного файла:	`e.in`
Имя выходного файла:	`e.out`
Ограничение по времени:	1 с
Ограничение по памяти:	256 Мб

Агент Вася 00* Пупкин обнаружил в своем почтовом ящике треугольное письмо. Послание из Центра. Для соблюдения секретности все послания из Центра снабжаются механизмом самоуничтожения, и Вася знает, что как только он вскроет конверт, специальное вещество, нанесенное на все три вершины треугольного послания, прореагирует с воздухом и воспламенится.

После того как Вася вычислил время сгорания письма и выпил чашку кофе, перед ним встала следующая задача. От письма на столе остался

пепел треугольной формы, и теперь он хочет собрать его в одну точку с помощью совка с прямолинейным основанием. Вася может провести основанием совка по столу, передвигая таким образом пепел. Вектор движения совка не меняется и не обязательно ортогонален основанию. Можно считать, что те точки треугольника, которые попали на основание совка, не меняют своего положения относительно совка, но остаются на столе.

Зафиксируем какую-нибудь точку на основании совка, тогда можно определить расстояние, которое пройдет эта точка, пока Вася собирает пепел. Назовем эту величину штрафом уборки. Так как Вася все пытается оптимизировать, то сейчас его интересует, каков минимальный штраф уборки?

Ограничения

Все координаты не превосходят по модулю 10^5. Все числа целые. Три вершины не лежат на одной прямой.

Формат входного файла

Входные данные содержат три строки. На каждой строке содержится пара чисел — координаты вершины треугольника.

Формат выходного файла

В единственную строку выходных данных вывести одно вещественное число — минимальный штраф. Ответ необходимо выводить с не менее чем шестью знаками после десятичной точки.

Пример

e.in	e.out
0 0 3 0 3 4	7.0000000000

Разбор задачи E. Clear after burning

Ясно, что всегда придется изменить направление, потому что треугольник невырожден. Наблюдением можно установить, что в задаче нужно найти накрывающий параллелограмм минимального периметра. Ответом будет его полупериметр.

Согласно теореме 1 из лекции, граница оптимального параллелограмма содержит сторону треугольника. При фиксированной стороне ab оптимальный параллелограмм есть либо прямоугольник, если на стороне ab нет

тупых углов, либо параллелограмм, вторая сторона которого равна второй стороне тупого угла.