Задача 1. Школа танцев

Имя входного файла:	`dance.in`
Имя выходного файла:	`dance.out`
Ограничение по времени:	2 *секунды*
Ограничение по памяти:	64 *мегабайта*
Максимальная оценка	100 *баллов*

В школу бальных танцев профессора Падеграса записались n учеников — мальчиков и девочек. Профессор построил их в один ряд, и хочет отобрать из них для первого занятия группу стоящих подряд учеников, в которой количество мальчиков и девочек одинаково. Сколько вариантов выбора есть у профессора?

Формат входных данных

В первой строке входного файла задано число n ($1 \le n \le 10^6$). Во второй строке задается описание построенного ряда из мальчиков и девочек — строка из n символов `a` и `b` (символ `a` соответствует девочке, а символ `b` — мальчику).

Формат выходных данных

В единственной строке выходного файла должно содержаться единственное число — количество вариантов выбора требуемой группы.

Примеры

dance.in	dance.out
3 Bab	2
8 Abbababa	13

Разбор

Автор задачи – Антонов В.Ю.

Автор разбора – Антонов В.Ю.

Рассмотрим данный ряд из мальчиков и девочек, сопоставив каждой девочке число 1, а каждому мальчику -1. Тогда в группе подряд стоящих учеников, в которой мальчиков и девочек одинаково, сумма соответствующих чисел будет равно 0. Заметим, что во всех остальных группах подряд стоящих учеников, указанная сумма не будет равна 0.

Благодаря этому утверждению, можно получить решение с $O(N^3)$ операций: перебираем все возможные подстроки и для каждой считаем соответствующую сумму. Если сумма равна 0, то увеличиваем ответ на единицу.

Заметим, что сумму для подстроки $[i, j]$, можно использовать для вычисления суммы для подстроки $[i, j+1]$. При помощи такой оптимизации получается решение, требующее порядка $O(N^2)$ операций.

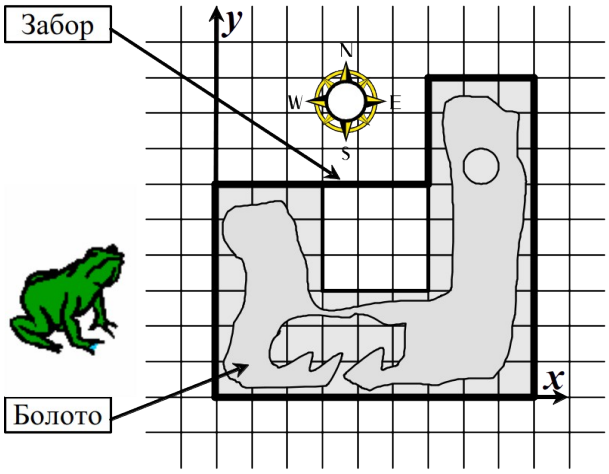
Теперь обозначим сумму для подстроки $[i, j]$ за $S[i, j]$. Тогда при $i > 1$ $S[i, j] = S[1, j] – S[1, i-1]$. Заметим, что если $S[i, j] = 0$, то $S[1, i] = S[1, j]$. Значит, для того, чтобы посчитать ответ, достаточно посчитать только суммы для $S[1, i]$, при этом среди всех сумм нас будут интересовать только количество одинаковых. Используя эти рассуждения, приходим к следующему решения:

1) Считаем все суммы $S[1, i]$ для $1 < i < N$.
2) Для каждой суммы считаем количество её вхождений в $S[1, i]$. Обозначим это количество за $Count[j]$.
3) Ответом будет сумма по всем j $Count[j]*(Count[j]-1)/2$. При этом стоит заметить, что к количеству подстрок, у которых $S[1, i] = 0$ следует добавить пустую строку.

Задача 2. Стеклянный забор

Имя входного файла:	`swamp.in`
Имя выходного файла:	`swamp.out`
Ограничение по времени:	2 *секунды*
Ограничение по памяти:	64 *мегабайта*
Максимальная оценка	100 *баллов*

В известном городе Санкт-Тверь решили построить новый микрорайон, представляющий в плане прямоугольную область. Границы микрорайона и его улицы по проекту ориентированы строго по сторонам света, причем улицы разбивают микрорайон на кварталы размером 1 км × 1 км.



Во время привязки исходного проекта к местности выяснилось, что некоторые кварталы по проекту микрорайона оказываются полностью или частично расположенными на топком болоте. Область, занимаемая болотом, связна и со всех сторон окружена подлежащими застройке кварталами микрорайона (область связна, если из любой ее точки можно добраться в любую другую, не выходя за пределы области).

Для сохранения экологии местности и обеспечения безопасности жителей занятую болотом область решили оградить стеклянным забором. Забор должен проходить только по границам кварталов проектируемого микрорайона, отделяя болото, и, возможно, некоторые кварталы проекта, не занятые болотом, от остальной части микрорайона.

Для экономии строительных материалов забор должен иметь минимальную длину. Среди всех заборов минимальной длины нужно выбрать тот, для которого площадь части микрорайона, попадающей внутрь забора, минимальна.

Требуется написать программу, которая спроектирует забор с заданными выше свойствами.

Формат входных данных

Входной файл содержит описание многоугольника — границы области, состоящей только из кварталов с заболоченными участками. Стороны многоугольника параллельны осям координат.

В первой строке задано целое число n — количество вершин в многоугольнике ($4 \le n \le 100\,000$, n четное). В каждой из следующих n строк заданы два целых числа — координаты очередной вершины при обходе этого многоугольника против часовой стрелки. Все числа не превосходят 10^9 по абсолютной величине. Никакие три последовательные вершины границы не лежат на одной прямой. Граница многоугольника не содержит самопересечений и самокасаний.

Формат выходных данных

Выходной файл должен содержать описание многоугольника, определяющего искомый забор. Формат описания многоугольника тот же, что и для входных данных. Никакие три последовательные вершины этого многоугольника не должны лежать на одной прямой.

Пример

swamp.in	swamp.out
8 0 0 9 0 9 9 6 9 6 3 3 3 3 6 0 6	6 0 0 9 0 9 9 6 9 6 6 0 6

Разбор

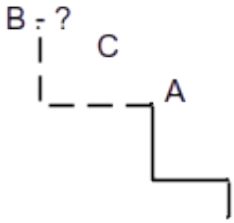
Автор задачи — Корнеев Г. А.
Автор разбора — Дворкин М. Э.

Рассмотрим вершины данного многоугольника, имеющие максимальную координату y. Среди них рассмотрим вершины с наименьшей и наибольшей координатой x. Легко понять, что отрезок, соединяющий эти две вершины, является стороной искомого забора. Аналогично, известны еще три стороны забора: крайняя левая, крайняя нижняя и крайняя правая.

Теперь следует решить четыре подзадачи — вывести участки забора между этими четырьмя отрезками. Опишем решение одной из этих задач — между правым и верхним отрезком, остальные три задачи решаются аналогично.

Будем строить забор от правой стороны. Ясно, что следующие две стороны забора будут направлены влево и вверх. Осталось понять, до какой вершины дойдут эти две стороны. Заметим, что внутри угла, образованного этими двумя сторонами, не должно быть вершин исходного многоугольника, иначе окажется, что некоторая вершина лежит вне забора.

Формализуем это условие. Пусть уже построенный участок забора заканчивается вершиной A, научимся определять, какой вершиной B будут оканчиваться следующие две стороны.

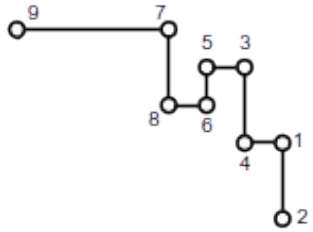


Не должно существовать вершины C, оказывающейся после проведения новых двух сторон снаружи забора, то есть такой вершины C, что

$$\begin{cases} x_B < x_C < x_A \\ y_C > y_A \end{cases}$$

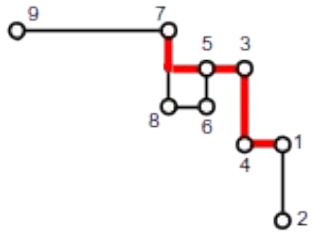
Значит из всех вершин, расположенных строго левее и строго выше A, вершина B имеет наибольшую координату x. Из всех же вершин с такой координатой x вершина B, как легко понять, имеет наибольшую координату y. (Если на этой вертикали есть вершина выше, то забор следует довести до нее). Это подталкивает к идее правильного решения.

Упорядочим все вершины исходного многоугольника по убыванию координаты x, в случае равенства координат x — по убыванию координат y.



Начнем строить забор с первой вершины. Будем перебирать вершины в указанном порядке, и как только обнаружится вершина со строго меньшей координатой x и строго большей координатой y, чем текущая, построим две стороны забора из текущей вершины в новую.

Например, для вершин, изображенных на рисунке, будет происходить следующее. Начав с вершины 1, мы пропустим вершину 2 (она ниже текущей, следовательно, не подходит) и остановимся на вершине 3. Построив две стороны забора, соединяющие 1-ю и 3-ю вершины, мы продолжим просматривать вершины, начиная с 4-й. Пропустив вершины 4, 5 и 6, мы найдем вершину 7 — первую по номеру вершину, лежащую строго левее и строго выше текущей (3-й). Построим забор между 3-й и 7-й вершинами. Здесь можно остановиться, поскольку мы дошли до верхней стороны забора, а можно продолжить просмотр вершин до конца, поскольку вершин с большей координатой y все равно не найдется.



Время работы указанного алгоритма складывается из времени сортировки вершин, составляющего $O(n \log n)$, и времени просмотра вершин, составляющего $O(n)$. Описанная процедура повторяется четыре раза (для построения четырех участков забора), но на асимптотику времени работы это не влияет. Итоговое время работы: $O(n \log n)$.

Частичные решения

В процессе построения можно искать каждую следующую вершину путем перебора всех вершин, то есть за время $O(n)$. Таким образом, построение всего забора требует $O(n^2)$ времени. Такое решение оценивалось в 70 баллов.

Более медленная реализация правильного решения, работающая за время $O(n^3)$ оценивалась в 40 баллов.

Возможны также решения, хранящие многоугольник (или вершины многоугольника) в двумерном массиве. Такие решения работают только на тестах, в которых площадь прямоугольника, ограничивающего болото, невелика. В зависимости от качества реализации такие решения оценивались в 40—50 баллов.

Наконец, можно было только разобрать случаи, когда ответом является прямоугольник, ограничивающий болото, и случаи, когда искомый забор совпадает с исходным многоугольником. Разбор одного из этих случаев приносил участнику 15 баллов, разбор обоих — 25 баллов.

В архиве олимпиады можно найти тесты, которые использовались для оценки решений участников, а также визуализатор, позволяющий увидеть исходный многоугольник и найденный забор (правильный или неправильный).

Задача 3. Несчастливые номера

Имя входного файла: `unlucky.in`
Максимальная оценка 100 *баллов*

Обычно автобусный билет с номером, состоящим из 6 цифр, считается счастливым, если сумма первых трех цифр его номера была равна сумме трех последних. Школьник Вася очень любил получать счастливые билеты, однако это случалось не так часто. Поэтому для себя он изменил определение счастливого билета.

Счастливым он считал тот номер, сумма некоторых цифр которого равнялась сумме оставшихся цифр. В его представлении билет с номером 561743 счастливый, так как 5+1+4+3=6+7.

Вася вырос, но по привычке в номерах различных документов пытается найти признаки счастливого номера. Для этого он расширил свое определение счастливого номера на n-значные номера лицевых счетов и других документов, состоящих из цифр от 0 до k ($1 \le k \le 9$). Номер документа он называет счастливым, если сумма некоторых цифр этого номера равняется сумме оставшихся. Остальные номера для него несчастливые. К сожалению, несмотря на расширенное понимание “счастья”, несчастливых номеров остается еще много...

Вам предлагается определить количество *несчастливых* n-значных номеров, которые можно составить, используя цифры от 0 до k. В номерах допускается любое количество ведущих нулей.

Входной файл `unlucky.in` находится в каталоге `c:\work\unlucky` вашего компьютера. Файл содержит описание нескольких видов номеров. Каждый вид номеров определяется значениями n и k. Для данного входного файла вы должны создать соответствующий ему выходной файл и отправить его на проверку жюри.

Формат входных данных

Входной файл содержит несколько пар значений n и k, каждая пара записана в отдельной строке.

Формат выходных данных

Для каждой пары значений n и k входного файла выведите в соответствующей строке выходного файла искомое количество несчастливых билетов или 0, если такое число вам получить не удалось. Количество строк во входном и выходном файлах должно совпадать.

Пример

Входной файл	Выходной файл
1 7 4 3 50 8 11 9	7 164 0 50184219171

Примечания

За правильное решение задачи для каждого вида номеров вы получите 5 баллов. Так, представленный в примере выходной файл соответствует 15 баллам.

При сдаче на проверку выходного файла во время тура вы будете получать одно из двух сообщений:

- Presentation Error — если файл не соответствует формату вывода. В этом случае файл не принимается на проверку.
- Accepted — если файл формату вывода соответствует. В этом случае файл принимается на проверку. Проверка правильности ответов в выходном файле осуществляется только после окончания тура.

Разбор

Автор задачи – Кузнецов В.А.

Автор разбора – Царев Ф.Н.

Прежде чем решать поставленную задачу, решим подзадачу, заключающуюся определении по заданному номеру, является ли он счастливым. Пусть задан билет, номер которого состоит из цифр $d_1d_2d_3...d_n$ и требуется проверить, является ли он несчастливым. Обозначим сумму цифр номера билета как $S = d_1 + d_2 + d_3 + ... + d_n$. Тогда рассматриваемая подзадача может быть сформулирована как «проверить, что не существует поднабора цифр числа, сумма по которому равна $S/2$».

Для этого решим противоположную задачу – «проверить, существует ли поднабор цифр числа, сумма по которому равна $S/2$». Это – известная NP-полная задача, называемая «задача о сумме подмножеств». Для ее решения можно применять перебор подмножеств или динамическое программирование [1, 2]. При использовании перебора время работы составляет $O(2^n)$, а при использовании динамического программирования – $O(Sn)$, где d – максимальная используемая цифра.

Для решения исходной задачи теперь можно использовать два метода. Более простой из них – перебор всех возможных номеров и проверка каждого из них одним из описанных выше способов на «несчастливость». Такое решение способно вычислить ответы для восьми тестов из предложенных 20. Таким образом, оно позволяет получить 40 баллов.

Более сложное решение основано на наблюдении о том, что для проверки номера на «несчастливость» достаточно знать мультимножество его цифр (какая цифра и сколько раз в него входит). В этом случае возникает вопрос, как по мультимножеству цифр вычислить число номеров с таким мультимножеством.

Решим эту подзадачу. Пусть в номере можно использовать цифры от *0* до *d*, цифра *0* может входить в номер n_0 раз, цифра *1* может входить в номер n_1 раз, …, цифра *d* может входить в номер n_d раз, при этом $\sum_{i=0}^{d} n_i = n$. Тогда цифру *0* можно расставить $C_n^{n_0}$, цифру *1* на оставшихся позициях можно расставить $C_{n-n_0}^{n_1}$, цифру *2* на оставшихся позициях можно расставить $C_{n-n_0-n_1}^{n_2}$, …, цифру d можно расставить $C_{n-n_0-n_1-\ldots-n_{d-1}}^{n_d}$. Для получения ответа на рассматриваемую подзадачу необходимо вычислить произведение указанных выше биномиальных коэффициентов: $C_n^{n_0} \cdot C_{n-n_0}^{n_1} \cdot C_{n-n_0-n_1}^{n_2} \cdot \ldots \cdot C_{n-n_0-n_1-\ldots-n_{d-1}}^{n_d}$.

Итого, правильное решение задачи получается, если скомбинировать перебор мультимножеств цифр (их число равно C_{n+k-1}^{k}) и проверку «несчастливости» номера с помощью динамического программирования. Кроме этого, решение можно ускорить, если учитывать, что номер с нечетной суммой цифр не может быть «счастливым». Описанное решение при аккуратной программной реализации без использования «длинной» арифметики правильно вычисляет ответы на 17 тестов из 20 примерно за 5 минут. Ответы на оставшиеся 3 теста превосходят границы стандартных целочисленных типов данных.

Кроме описанных решений, можно было получить 10 баллов за решение, обрабатывающее случай $k=1$, и 15 баллов за решение, обрабатывающее случай $k=2$.

Источники

1. *Шень А.* Программирование: теоремы и задачи. – М.: МЦНМО, 2004.
2. Сайт http://informatics.mccme.ru, раздел «Динамическое программирование» http://informatics.mccme.ru/moodle/mod/book/view.php?id=266&chapterid=60

Задача 4. Счастливые цифры

Имя входного файла:	`lucky.in`
Имя выходного файла:	`lucky.out`
Ограничение по времени:	2 *секунды*
Ограничение по памяти:	64 *мегабайта*
Максимальная оценка	100 *баллов*

Школьнику Васе нравятся числа, которые заканчиваются счастливыми для него цифрами k. Поэтому каждый раз, когда он видит какое-нибудь натуральное число n, он сразу пытается подобрать такое d ($d \geq 2$), что число n в системе счисления с основанием d заканчивается как можно большим количеством цифр k.

Требуется написать программу, которая по заданным числам n и k найдет такое d, чтобы число n в системе счисления с основанием d заканчивалось как можно большим количеством цифр k.

Формат входных данных

Входной файл содержит два целых десятичных числа n и k ($1 \leq n \leq 10^{11}$; $0 \leq k \leq 9$).

Формат выходных данных

В выходной файл выведите два числа: d — искомое основание системы счисления и l — количество цифр k, которым заканчивается запись числа n в этой системе счисления. Если искомых d несколько, выведите любое из них, не превосходящее 10^{12} (такое всегда существует).

Примеры

lucky.in	lucky.out	комментарий
49 1	3 2	$49_{10} = 1211_3$
7 5	3 0	Ни в одной системе счисления 7 не заканчивается на цифру 5

Разбор

Автор задачи – научный комитет XX Всероссийской олимпиады школьников по информатике.

Автор разбора – Батузов К. А.

Пусть в системе счисления d на конце числа N встречается хотя бы одна цифра k, тогда $N=Ad+k$. Таким образом d является делителем числа $N-k$. Перебрать все делители числа $N-k$ можно за $O(\sqrt{N})$. Действительно, любой делитель числа N либо не превосходит $\sqrt{N}$, либо равен N/a, где a – делитель числа N, меньший $\sqrt{N}$. Таким образом, перебрав все числа d, являющиеся делителями числа $N-k$, посчитав количество цифр k на конце числа N и выбрав лучшее, мы найдем ответ на задачу. В данном решении нужно отдельно разобрать случай $N=k$, так как у числа 0 бесконечно много делителей.

Существует решение данной задачи, работающее за $O(\sqrt[3]{N})$. Оно основывается на следующих фактах:

- Если $N=k$, то искомое число $d=N+1$, иначе
- если $N \leq 2k$, то N не может заканчиваться на цифру k и нам подойдет любое d, например $d=2$, иначе
- если на конце числа N в системе счисления d идет более двух цифр k на конце, $d \leq \sqrt[3]{N}$, иначе
- если на конце числа N в системе счисления d идет более одной цифры k на конце, то $N=Ad^2+kd+k$, где A – целое, $0 \leq A \leq N/d^2 \leq \sqrt[3]{N}$. Решить квадратное уравнение и найти возможные значения d не представляет труда. Иначе
- в системе счисления $N-k$ число N заканчивается на одну цифру k.

Решения, перебирающие d от 2 до $N+1$ в данной задаче набирают 45 баллов. Если к нему добавить отсечение по времени и отдельное рассмотрение случая $d=N-k$, то оно может набрать до 75 баллов.

Задача 5. Тапкодер

Имя входного файла:	`topcoder.in`
Имя выходного файла:	`topcoder.out`
Ограничение по времени:	2 *секунды*
Ограничение по памяти:	64 *мегабайта*
Максимальная оценка	100 *баллов*

Ассоциация Тапкодер организует Всемирное парное соревнование сильнейших программистов. К участию в соревновании допущены первые 2^k зарегистрировавшихся участников, которым присвоены номера от 1 до 2^k.

Соревнование будет проходить по олимпийской системе. В первом туре первый участник встречается со вторым, третий с четвертым и так далее. В каждой паре победителем становится участник, первым решивший предложенную задачу, при этом ничьих не бывает. Все победители очередного тура и только они являются участниками следующего тура. В каждом туре пары составляются из участников в порядке возрастания присвоенных им номеров. Соревнование продолжается до тех пор, пока не останется один победитель.

Организаторам стало известно, что некоторые пары участников заранее договорились о результате встречи между собой, если такая встреча состоится. Для всех остальных встреч, кроме n договорных, возможен любой исход.

Некоторые m участников соревнования представили свои резюме в ассоциацию Тапкодер с целью поступления на работу. Организаторов интересует, до какого тура может дойти каждый из претендентов при наиболее благоприятном для него стечении обстоятельств. При этом для каждого участника в отдельности считается, что все недоговорные встречи, в том числе те, в которых он не участвует, закончатся так, как ему выгодно, а все состоявшиеся договорные встречи закончатся в соответствии с имеющимися договоренностями.

Требуется написать программу, которая для каждого из претендентов определяет максимальный номер тура, в котором он может участвовать.

Формат входных данных

В первой строке входного файла заданы три целых числа k ($1 \le k \le 60$), n ($0 \le n \le 100\,000$) и m ($1 \le m \le 100\,000$). В следующих n строках описаны n пар участников, которые договорились между собой о том, что первый из двух участников пары выиграет встречу, если она состоится. Гарантируется, что каждая пара участников присутствует во входных данных не более одного раза, при этом, если задана пара x y, то пары y x быть не может, кроме того, $x \ne y$. В последней строке файла перечислены номера участников, желающих работать в Тапкодере, в порядке возрастания их номеров. Все номера претендентов на работу различны.

Формат выходных данных

Выходной файл должен содержать m целых чисел — максимальные номера туров, до которых могут дойти соответствующие претенденты на работу. Туры нумеруются от 1 до k.

Примеры

topcoder.in	topcoder.out	комментарий
2 0 3 1 3 4	2 2 2	У каждого из участников есть возможность выйти в финал, так как договорных матчей нет
3 1 1 3 1 1	3	Если четвертый участник выиграет у третьего, то договорная встреча первого и третьего не состоится, что благоприятно для первого
3 3 4 1 2 1 3 4 1 1 2 3 4	3 1 2 3	Первому участнику благоприятно во втором туре играть с третьим, а не с четвертым, в свою очередь, четвертый может выиграть у третьего и также выйти в финал

Разбор

Автор задачи – Андреева Е.В.

Авторы разбора – Назаров С. И., Денисов Д. В.

Для начала отметим, что для каждой договорной игры (*i, j)* тур, в котором она может состояться, определяется однозначно. Рассмотрим некоторую игру (*i, j)*. Подсчитаем *x = (i — 1) xor (j — 1)*, где *xor* — исключающее или. Тогда номер старшего единичного бита в числе *x* равен номеру тура, в котором состоится игра (*i, j)*. Например, для игроков 1 и 5 получим (1 — 1) *xor* (5 — 1) = 1 *xor 4 = 5*. Номер старшего единичного бита в числе $5 = 101_2$ равен 3, поэтому игроки 1 и 5 могут сыграть только в третьем туре.

Будем моделировать соревнование тур за туром. В процессе моделирования очередного тура для каждого игрока будем определять, сможет ли он пройти в следующий тур или нет. Для этого рассмотрим его договорные матчи с еще не выбывшими участниками, в которых этот игрок проигрывает. Очевидно, что если у него нет возможности сыграть с кем-либо другим, то он не выйдет из этого тура, в противном же случае он пройдет дальше. То есть необходимо сравнить количество договорных матчей с еще не выбывшими противниками этого игрока в этом раунде и число всех возможных его противников. Подсчитать количество проигранных договорных матчей не представляет большой сложности. Количество всех возможных матчей для этого игрока в этом туре будем вычислять как *right — left + 1 — amount(left, right)*, где *[left, right]* — отрезок, содержащий номера всех игроков, с которыми он может сыграть в этом туре, а *amount(left, right)* — количество выбывших на предыдущих турах игроков с номерами от *left* до *right*.

Нетрудно заметить, что для каждого матча тура *t* может дойти 2^t игроков. Поэтому для игрока *i left* и *right* определяются так. Сначала определим *l* и *r* — номера игроков, которые могут дойти до матча с участием игрока *i*. *r* равно наименьшему числу, кратному 2^t и не меньшему *i*. А *l* определяется как $r — 2^t + 1$. Пусть *mid = (l + r - 1) / 2*. В туре номер *t* игроки с номерами от *l* до *mid* играют с игроками с номерами от *mid + 1* до *r*. Поэтому, если *i > mid,* то отрезок *[left, right]* есть *[mid + 1, r]*, в противном случае, отрезок *[left, right]* есть отрезок *[l, mid]*.

Рассмотрим пример. Пусть в туре номер 3 игрок 1 проигрывает игрокам 5, 6, 7, а выбывшие к данному моменту игроки: 2, 6, 8. Тогда игрок 3 может проиграть в этом туре двум игрокам: 5 и 7 (игрок 6 выбыл). А может сыграть в этом туре с игроками, номера которых лежат в отрезке [5, 8], причем двое из них выбыли, то есть *amount(5, 8) = 2*. Получается, что игрок 1 может сыграть с 8 — 5 + 1 — 2 = 2 игроками. То есть ровно с теми игроками, которым он проигрывает. Поэтому у первого игрока нет шансов выйти в следующий тур.

К сожалению, количество игроков, которые имеют шанс пройти в следующий тур велико. Однако число игроков, которые точно не пройдут в следующий тур соревнования, не превышает число договорных матчей в этом туре. Действительно, игрок, не проигрывающий в этом туре ни одного договорного матча, обязательно выйдет в следующий тур. То есть за весь турнир максимум про *n* игроков можно сказать, что они точно проиграют в некотором раунде, у остальных же есть шанс не проиграть ни в одном туре. Поэтому будем хранить массив игроков, которые выбыли на предыдущих турах.

Единственную сложность теперь представляет определение *amount(left, right)*. Один из вариантов — при необходимости просматривать весь массив уже выбывших игроков и запоминать, сколько из них лежит в отрезке *[left, right]*. Тогда получим решение с асимптотикой $O(n^2)$, так как для каждого игрока приходится рассматривать весь список уже выбывших игроков. Такое решение не укладывается в требуемое ограничение по времени на эту задачу.

Для того чтобы избежать просмотра всего списка выбывших участников, можно воспользоваться следующей идеей. Будем хранить список выбывших участников в сортированном виде. Тогда несложно видеть, что игроки с номерами из отрезка *[left, right]* будут находиться в массиве в некотором отрезке индексов *[lb, rb]* и только в нем. Границы отрезка *lb* (минимальный индекс числа, не меньшего *left*) и *rb* (максимальный индекс числа, не большего *right*) можно найти двоичным поиском. Однако для того, чтобы массив хранился в отсортированном виде, необходимо после моделирования каждого тура поддерживать это состояние. Для этого можно просто пересортировывать его заново (решение за *O(knlogn)*). Другой вариант — перебирать в

каждом туре участников в порядке возрастания и сливать два отсортированных массива за время $O(n)$. Асимптотическая оценка этого решения $O(n\log n + kn)$. Можно заметить, что для того, чтобы игрок выбыл в туре t, требуется не менее 2^t матчей. Поэтому в турах, больших чем $\log n$ игроки выбывать не будут. Поэтому моделировать туры с номерами, большими чем $\log n$ бессмысленно. Это означает, что асимптотика решений изменится на $O(n\log^2 n)$ и $O(n\log n)$ соответственно.

Все правильные решения, асимптотика которых зависит от 2^k, набирают от 40 до 50 баллов. Решение за $O(n^2)$ набирает около 70 баллов. Все правильные решения с асимптотической оценкой $O(kn\log n)$ или лучше набирают полный балл.

Задача 6.Сочи-2014

Имя входного файла:	`olymp.in`
Имя выходного файла:	`olymp.out`
Ограничение по времени:	2 *секунды*
Ограничение по памяти:	64 *мегабайта*
Максимальная оценка	100 *баллов*

К предстоящей олимпиаде в Сочи требуется возвести N олимпийских объектов. Процесс строительства каждого объекта определяется освоением выделяемых на него денежных средств.

В строительстве объектов готовы участвовать K фирм. Фирмы имеют разные строительные мощности, выраженные в количестве денежных средств, которые фирма может осваивать в единицу времени.

В каждый момент времени фирма может осуществлять работы только на одном объекте. В строительстве одного объекта не могут одновременно участвовать несколько фирм. В любой момент времени любой объект может быть передан для продолжения строительства любой фирме.

Администрация строительства олимпийских объектов заинтересована в скорейшем освоении денежных средств, поэтому хочет составить такой график работ, при следовании которому строительство будет завершено в кратчайшие сроки. В графике будет указано время, в течение которого тот или иной объект будет строиться какой-то фирмой.

Напишите программу, результаты работы которой позволят администрации построить требуемый график.

Формат входных данных

Первая строка входного файла содержит целое число N — количество объектов ($1 \le N \le 50$). Во второй строке содержатся разделенные пробелами целочисленные значения $S_1, S_2, S_3, \ldots, S_N$ объемов денежных средств, выделяемых для строительства каждого из объектов. Числа S_i выражены в тысячах рублей, положительные и не превышают 1000.

В третьей строке находится целое число K — количество строительных фирм ($1 \le K \le 50$). Четвертая строка содержит разделенные пробелами целочисленные значения мощностей каждой из фирм $V_1, V_2, V_3, \ldots, V_K$ в тыс.руб/час. Числа V_j положительные и не превышают 1000.

Формат выходных данных

Первая строка выходного файла содержит действительное число T — время в часах окончания всех работ, считая с начала строительства, выведенное не менее чем с тремя точными знаками после запятой. Далее в каждой строке выходного файла содержатся разделенные пробелами три числа: t, i, j, где действительное число t — время от начала строительства в часах, в которое j-я фирма приступает к строительным работам на i-м объекте.

Значения времен необходимо выводить с максимально возможной точностью.

Строки должны быть отсортированы по неубыванию t.

Примеры

olymp.in	**olymp.out**
2 24 20 2 3 2	8.800 0 1 1 0 2 2 6.4000000 1 2 6.4000000 2 1
3 100 100 100 4 5 5 10 10	12.00000 0 1 3 0 2 4 0 3 1 4 2 2 4 3 4 8 1 1 8 3 4 8 2 3

Разбор

Автор задач – Волченков С. Г.

Авторы разбора – Вальтман В. А., Копелиович С. В.

Пусть объёмы работ и мощности фирм упорядочены по убыванию. Понятно, что если работ меньше, чем фирм, то можно оставить только лучшие N фирм. С другой стороны, если фирм меньше, чем объектов, то можно добавить N-K фиктивных фирм, с нулевой скоростью. Таким образом, можно считать, что фирм и объектов поровну. Начиная с этого момента, так и будем считать.

Заметим, что самая объёмная работа не может быть выполнена быстрее, чем её выполнит самая мощная фирма. Поэтому, отношение S_1/V_1 ограничивает T снизу. Также, две самые объёмные работы не могут быть выполнены быстрее, чем их выполнят две самые мощные фирмы, т.е. $(S_1+S_2)/(V_1+V_2)$ – тоже нижняя граница для T. Аналогичные ограничения верны для любого количества работ и фирм. Сравнив все отношения частичных сумм, мы можем найти T, при котором ограничение максимально. Не очень трудно показать (по индукции), что это ограничение достигается. Найденное T назовём критическим.

Опишем алгоритм поиска расписания, при котором достигается критическое T.

В нулевой момент назначим для выполнения работ объекты и фирмы по порядку (некоторые объекты или фирмы могут оказаться не у дел). Зададим произвольное t (большее нуля, но меньшее T). Рассчитаем ситуацию, когда строительство t часов ведётся по указанному распределению. Найдём, как после этого изменилось критическое время. Если оно не уменьшилось – значит t можно увеличить. Если оно увеличилось – значит t нужно уменьшить. Дихотомия (двоичный поиск по ответу) даст результат с заданной погрешностью. В момент t проводим всё заново, т.е. пересчитываем оставшиеся объёмы работ, пересортировываем, находим новое T и повторяем алгоритм. Конечность алгоритма очевидна, т.к. после каждого шага какая-то фирма покидает какой-то объект навсегда (иначе t можно было бы увеличить).

Докажем, что количество шагов не более N. Критическим номером u ($1<=u<=N$) назовём такое u, что $(S_1+S_2+...+S_u)/(V_1+V_2+...+V_u) = T$ – критическое. Тогда докажем, что при каждом шаге количество критических номеров строго увеличивается. Очевидно, что при любом t для критического u, соответствующее частное будет в точности равно T-t. Поэтому для крайнего значения t появится ещё один u, для которого частное будет равно критическому T. Таким образом, количество критических номеров строго увеличивается, что и означает, что количество шагов не превосходит N. В каждой итерации выполняется двоичный поиск, внутри которого нужно отсортировать массив из N элементов. Таким образом, время работы получается $O((N \log(N))^2)$. Более того, переключений будет не более N на каждом шаге, то есть в сумме не более $O(N^2)$.

Разобрав отдельные случаи ($N<=2$, $K<=2$, $V_1=V_2=\ldots=V_n$) можно было получить до 30 баллов. Жадное решение, выполняющее назначение фирм на объекты только в начальный момент времени и когда строительство какого-либо объекта заканчивается, могло получить от 40 до 50 баллов. Решение, выполняющее переназначение каждые ε единиц времени (где ε – небольшое число) набирает от 50 до 80 баллов.