Задача 1 Урок физкультуры

Имя входного файла:	`sport.in`
Имя выходного файла:	`sport.out`
Максимальное время работы на одном тесте:	1 секунда
Максимальный объем используемой памяти:	256 мегабайт
Максимальная оценка за задачу:	100 баллов

Сегодня в 5-А классе праздник — урок физкультуры. Традиционно ребята в это время после небольшой разминки играют в футбол. Коля очень любит футбол, но сегодня, проходя мимо учительской, он случайно услышал, что несколько самых сильных школьников из 5-А во время урока физкультуры должны помочь разгружать новые парты. Что же делать? Ведь Коля предпочитает поиграть в футбол!

В голове Коли моментально созрел план. Коля знает, что в 5-А классе N школьников. Он хочет воспользоваться тем, что учитель физкультуры Иван Петрович на уроках вместо обычной расстановки школьников в шеренгу по росту практикует расстановку «по силе». Для этого Иван Петрович сначала расставляет N школьников по росту, а затем $(N-1)$ раз проходит вдоль шеренги слева направо, каждый раз начиная с самого левого (первого) школьника и заканчивая предпоследним школьником справа. Проходя мимо школьника, который стоит на i-м месте ($1 \le i \le N-1$), Иван Петрович просит его помериться силой с соседом справа, который стоит на $(i+1)$-м месте. Если школьник, стоящий левее, оказывается сильнее своего соседа справа, то они меняются местами. Если же силы школьников оказываются равны, либо слева стоит более слабый школьник, то школьники остаются на своих местах. После этого Иван Петрович просит помериться силой школьников, стоящих на местах $(i+1)$ и $(i+2)$, и т. д., заканчивая каждый проход школьниками, которые стоят на местах $(N-1)$ и N. При любом сравнении «по силе» все школьники, включая Колю, показывают свою реальную силу, так как не хотят прослыть слабаками.

Для увеличения шансов поиграть в футбол, Коля хотел бы оказаться как можно левее в получившейся шеренге. Силу каждого из своих одноклассников он знает. По предыдущим занятиям Коля заметил, что если присесть «завязывать шнурки» при каком-либо из проходов учителя, то Иван Петрович во время такого прохода не будет просить Колю мериться силой ни с соседом слева, ни с соседом справа, и, тем более, не будет просить мериться силой соседей Коли, так как они не стоят рядом. Однако, чтобы сохранить силы для игры в футбол, Коля не может присесть более k раз.

Требуется написать программу, которая определит, как нужно действовать Коле, чтобы после проведения расстановки «по силе» оказаться в шеренге как можно левее.

Формат входных данных

В первой строке входного файла задаются три целых числа: N — число школьников в классе, p — место Коли в шеренге по росту, и k — количество приседаний, которое Коля может сделать, не потеряв при этом способность играть в футбол ($2 \le N \le 100\,000$, $1 \le p \le N$, $1 \le k \le N-1$).

Во второй строке задаются целые числа $a_1, a_2, ..., a_N$ ($1 \le a_i \le 10^9$). Число a_i показывает силу школьника, который стоит на i-м месте по росту. Школьник, который стоит на i-м месте в расстановке по росту, сильнее школьника, который стоит j-м месте в расстановке по росту, если $a_i > a_j$.

Формат выходных данных

В первой строке выведите самую левую позицию в шеренге, в которой может оказаться Коля после построения школьников «по силе». Во второй строке выведите любую из возможных стратегий Коли, приводящую к этому результату. Стратегия выводится в виде строки из $(N-1)$ символов, j-й символ этой строки должен быть равен символу «+», если на j-м проходе Коле необходимо присесть, и символу «-» — в противном случае.

Пример

sport.in	sport.out
10 7 7 8 3 5 4 5 7 4 2 1 3	3 ---++-+++

Примечание

Решения, корректно работающие при $N \le 3000$, будут оцениваться, исходя из 70 баллов.

Разбор

Первое замечание, существенно упрощающее понимание решение данной задачи, состоит в том, что нас интересует только соотношение сил остальных учеников с силой Коли, но не соотношение их сил между собой. А именно, заменим все силы учеников, которые слабее Коли, на 0, силу Коли на 1, а силы учеников сильнее Коли на 2. Если какие-то ученики равны Коле по силе, то в случае, когда они стоят левее его, их сила станет равной 0, а если правее – то 2. Нетрудно проверить, что при таком преобразовании ответ в задаче не изменится, и более того, не изменится с точностью до перестановки двоек и нулей результат после каждого прохода Ивана Петровича.

Предположим теперь, что Коля вообще не хочет приседать. Что тогда с ним произойдёт? Пока слева от Коли будет хоть одна двойка, на каждой итерации он будет двигаться на одно место влево, меняясь с одной из таких двоек. А как только слева от Коли останутся одни нули, то он дальше сможет двигаться только вправо. Более того, можно понять, на сколько именно позиций он будет сдвигаться вправо. Разобьём всех учеников правее Коли на группы – группа нулей (возможно, пустая), потом группа двоек, потом опять группа нулей, и так далее. Тогда на каждом проходе учителя ровно одна двойка из каждой группы будет меняться со всеми нулями справа от неё, и тем самым переходить в следующую группу. Одна двойка переходит из первой группы двоек во вторую, одна из второй в третью, одна из третьей в четвёртую, и так далее – в результате просто фактически одна двойка переходит из первой группы в конец. Тем самым, если Коля будет двигаться вправо на x-ом шаге, то он поменяется местами со всеми нулями, стоящими левее x-ой двойки (ведь первые x-1 двойка уйдут в конец, и Коля будет меняться с нулями, пока не упрётся в x-ую двойку).

Отсюда становится ясной оптимальная стратегия Коли – необходимо в последний раз двигаться вправо как можно раньше, т.е. приседать на как можно большем числе последних раундов. Объединяя это соображение с вышеизложенными, решение целиком таково: необходимо не приседать, пока слева есть двойки, а затем присесть либо все оставшиеся разы, либо, если сил на это не хватает, то последние k раз.

Для определения окончательной позиции Коли также можно воспользоваться вышеизложенными соображениями. А именно, левее его не будет ни одной двойки, а нулей будет столько, сколько их в исходной позиции до x-ой двойки, где x – количество шагов, на которых Коля не приседает (эти соображения относятся к нулям и двойкам правее Коли, к ним ещё надо прибавить количество нулей левее).

Такое решение работает за время O(n) и легко укладывается в отведённое время.

Задача «Электрички на перегонах не меняют»

Хранение входных данных

Построим граф, в котором все вершины будут соответствовать исходным станциям, а ребра – перегонам. Так как по условию в графе не может быть циклов, граф является лесом, и состоит из нескольких компонент связности, каждая из которых – дерево. Для хранения графа можно использовать список. При использовании матрицы смежности из-за большого количества вершин графа возникает переполнение по памяти.

Для хранения маршрутов электричек необходимо использовать списки, либо динамически выделять память под каждый маршрут. Применение двумерного массива недопустимо из-за того, что длина отдельного маршрута и общее количество электричек может быть достаточно большим и это приведет к переполнению по памяти.

Идея решения

В этой задаче существует несколько принципиально разных решений. В каждом решении мы последовательно обрабатываем вершины, ребра или пути электричек при этом мы должны каждой вершине присвоить некоторое уникальное число, которое обозначает тарифный номер станции. Наша цель – ориентировать непротиворечивым образом электрички так, чтобы вдоль каждого маршрута электрички тарифные номера строго возрастали или строго убывали. Такая «ориентация» маршрутов автоматически задает ориентацию ребер, входящих в маршрут.

В процессе решения задачи мы должны учитывать два типа ограничений: ограничения в вершинах и ограничения на ребрах. Ограничение в вершинах означает, что все маршруты, проходящие через одну и ту же вершину должны иметь один и тот же тарифный номер в этой вершины. Ограничения на ребрах означает, что все маршруты, проходящие через одно и то же ребро должны иметь ориентацию, согласованную с ориентацией ребер. Таким образом, задача сводится к построению корректной ориентации ребер графа. Если граф ориентировать не удается (возникает цикл), необходимо вывести ответ «NO». В противном случае граф маршрутов будет ациклическим (исходный граф – дерево), поэтому при помощи топологической сортировки можно восстановить тарифные номера всех станций, через которые проходят маршруты электричек. Для оставшихся вершин тарифные номера можно указать произвольным образом.

Реализация алгоритма

Существует несколько видов «квадратичных решений», с асимптотиками $O(kl)$, $O(nl)$, $O(n^2)$, $O(k^2)$, где n – количество вершин графа, k – количество маршрутов, l – общая длина всем маршрутов электричек (количество ребер). Каждое из этих решений может

быть доведено до правильного решения, работающего за линейное время O(n+l) при помощи несложных технических приемов. Все квадратичные решения набирают не более 70 баллов.

Решение с асимптотикой O(kl) обрабатывает маршруты электричек в некотором порядке и для каждого маршрута устанавливает такую тарифную нумерацию (нумерацию), чтобы не возникало противоречий в вершинах и на ребрах. Если мы будем рассматривать пути в хаотичном порядке, тогда получим некоторую эвристику, которая набирает от 20 до 40 баллов. Чтобы это решение стало правильным, мы должны сначала обрабатывать маршруты, для которых можно согласовать ограничения на ребрах, затем маршруты, для которых можно согласовывать ограничения в вершинах. Если маршруты, для которых необходимо согласовывать ограничения отсутствуют, можно перейти к рассмотрению произвольного маршрута. Фактически в этом решении мы нумеруем первый маршрут, затем все маршруты, у которых есть общие ребра с данным маршрутом, затем все маршруты, у которых есть общие вершины и так далее. Для поиска подходящего маршрута потребуется просмотреть ребра всех маршрутов, то есть потребуется O(l) времени, а так как всего k маршрутов общая оценка работы данного алгоритма O(kl). Преимущество этого алгоритма в том, что он не использует топологической сортировки, так как строит нумерацию непосредственно в ходе работы, и совершенно не использует объект «ребро», а вместо этого оперирует с объектом «пара вершин». Таким образом, мы можем в этом решении полностью отказаться от хранения графа.

В предыдущем решении мы рассматривали маршруты, а затем ориентировали ребра. В этом решение будем выполнять то же самое в обратном порядке: сначала ориентируем ребра, а после этого ориентируем все маршруты, содержащие данное ребра. Заведем очередь и будем в нее добавлять все ребра графа, на которых уже задана нумерация. Извлечем очередное ребро из очереди и рассмотрим все маршруты, которые содержат данное ребро. Ориентируем каждое из этих ребер в соответствие с ориентацией маршрута и добавим их в очередь. Перейдем к рассмотрению следующего ребра. На каждом шаге алгоритма количество ориентированных ребер увеличивается на единицу. Таким образом, нам потребуется не более чем O(n) итераций. На каждой итерации нам придется по заданному ребру определять множество маршрутов, которые проходят через заданное ребро. Для этого нам необходимо пробежаться по всем ребрам маршрутов, то есть потребуется O(l) операций. Если для каждого ребра заранее завести список маршрутов, проходящих через ребро, время выполнения этой операции будет O(1). Кроме того, нам необходимо для каждого ребра маршрута определять, какое это будет ребро в исходном графе. В зависимости от реализации время выполнения этой операции будет O(n), O(Log(n)) или O(1). Таким образом, суммарное время работы данного алгоритма может быть O(nl), $O(n^2)$, O((n+l)Log(n)) или O(n+l).

Третий тип решения заключается в том, что мы строим вспомогательный граф, вершинами которого являются маршруты. Проводим ребро в этом графе в том случае, если указанная пара маршрутов содержит общее ребро в исходном графе. Затем разбиваем граф на несколько компонент связности и восстанавливаем ориентацию исходных ребер графа. Для поиска компонент связности потребуется порядка $O(k^2)$ ребер. Это решение наиболее сложное для реализации. Для того чтобы получить линейное время исполнения, необходимо поиск компонент связности в новом графе объединять с одновременной ориентацией ребер в исходном графе.

Во всех решениях кроме первого, на заключительном этапе необходимо применять топологическую сортировку. В зависимости от реализации алгоритма для сортировки потребуется O(n) O(n Log(n)) или $O(n^2)$ операций.

Разбалловка

В зависимости от размера входных данных все тесты были разбиты на пять групп (см. таблицу). Отдельно были пять тестов на ответ «NO» и шесть тестов, в которых было несколько компонент связности. Было пять тестов, в которых маршруты не имели общих ребер. В одном из тестов присутствовал маршрут максимальной длины. Многие решения с применением рекурсии на этом тесте завершались аварийно из-за нехватки размера стека.

№ теста	Кол-во баллов	Описание
01-12	24	Небольшие тесты, которые должны пройти все решения
13-15	6	Граничные случаи
16-30	30	Тесты, которые пройдут квадратичные решения: $O(k^2)$, $O(n^2)$, O(kl), O(nl)
31-39	18	Тесты для разделения квадратичных решений
40-50	22	Большие тесты. Должны проходить только правильные решения

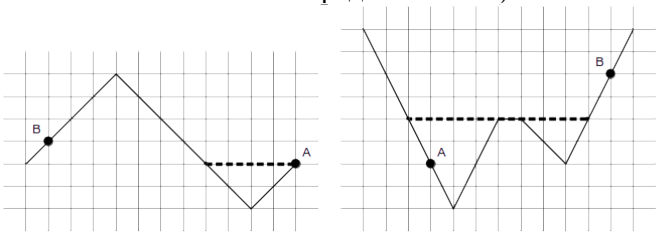
Кол-во баллов	Описание
100	Правильное решение: O(n+l), O((n+l)Log n)
90	Правильные решения, которые не обрабатывают тесты на «NO»
88	Правильные решения, которые обрабатывают только один компонент
70	Квадратичные решения: $O(k^2)$, $O(n^2)$, O(kl), O(nl)
50	Жадная эвристика с применением топологической сортировки
40	Жадная эвристика со склеиванием по ребру и вершине
20	Случай, когда маршруты электричек не имеют общих точек
10	Решения, которые проходят тесты на «NO» и какое-то количество тестов «Yes»

Задача 3 Ударим мостом по бездорожью

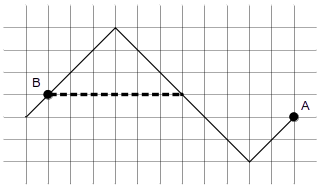
Имя входного файла:	`bridge.in`
Имя выходного файла:	`bridge.out`
Максимальное время работы на одном тесте:	1 секунда
Максимальный объем используемой памяти:	256 мегабайт
Максимальная оценка за задачу:	100 баллов

Профиль Уральских гор задается ломаной (x_1, y_1), (x_2, y_2), …, (x_N, y_N), для координат вершин которой верны неравенства $x_1 < x_2 < \ldots < x_N$. Начальные и конечные точки профиля расположены на уровне моря ($y_1 = y_N = 0$).

На горном профиле заданы две различные точки A и B, между которыми требуется проложить дорогу. Эта дорога будет проходить по склонам гор и проектируемому горизонтальному мосту, длина которого не должна превышать L. Оба конца моста находятся на горном профиле. Дорога заходит на мост с одного конца и выходит с другого. Мост не может содержать точек, расположенных строго под ломаной (строительство тоннелей не предполагается).



Возможные примеры расположения моста



Невозможное расположение моста

Достоверно известно, что строительство такого моста в данной местности возможно, причем позволит сократить длину дороги из точки A в точку B. Требуется написать программу, которая определит такое расположение горизонтального моста, что длина дороги от точки A до точки B будет наименьшей.

Формат входных данных

Первая строка входного файла содержит два целых числа N и L — количество вершин ломаной ($2 \le N \le 100\,000$) и максимальную длину моста ($1 \le L \le 10^6$) соответственно. Вторая строка входного файла содержит координаты точки A, третья строка — координаты точки B. Точки A и B различны.

Последующие N строк содержат координаты вершин ломаной (x_1, y_1), (x_2, y_2), …, (x_N, y_N). Координаты вершин ломаной, а также точек A и B, задаются парой целых чисел, не превосходящих по абсолютному значению 10^6. Гарантируется, что $x_1 < x_2 < \ldots < x_N$ и $y_1 = y_N = 0$, а также, что точки A и B принадлежат ломаной.

Формат выходных данных

В первой и второй строках выходного файла выведите координаты концов моста с точностью не менее 5 знаков после десятичной точки. В случае, когда решений несколько, выведите любое из них.

Примеры

bridge.in	bridge.out
4 6 13 0 2 1 1 0 5 4 11 -2 13 0	13.00000 0.00000 9.00000 0.00000
6 8 3 -6 11 -2 0 0 4 -8 6 -4 7 -4 9 -6 12 0	2.00000 -4.00000 10.00000 -4.00000

Примечание

Решения, корректно работающие при $N \le 2000$, будут оцениваться, исходя из 80 баллов.

Разбор

Если город *A* находится правее города *B*, то обменяем их местами. После этого добавим города *A* и *B* к множеству вершин ломаной. При этом, если точка не совпадает с вершинами ломаной, то содержащее ее ребро разбивается на два. В дальнейшем будем считать, что *A* и *B* — индексы соответствующих вершин ломаной.

Отметим, что при малых смещениях моста вверх или вниз между парой склонов его длина изменяется линейно, при этом длина пути из A в B так же изменяется линейно (см. рис. 1). Следовательно, зависимость дины пути от высоты моста является кусочно-линейной функцией, и, соответственно, оптимальный мост будет либо иметь длину *L*, либо иметь общие точки с вершинами ломаной.

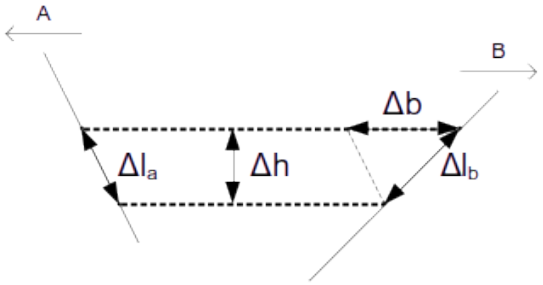


Рис 1. Изменение длины кратчайшего пути

Также заметим, что мост иногда выгодно не только поднимать, но и опускать, как, например, в случае на рис. 2. При этом движение в начале производится в направлении обратном ожидаемому, что требуется учитывать при вычислении расстояний..

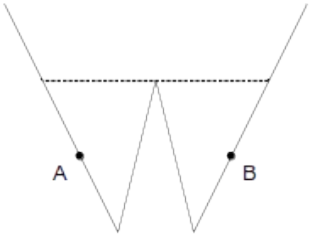


Рис 2. Пример выгодности опускания моста

В процессе решения нам потребуется быстро находить расстояние от точки на ломаной до вершин *A* и *B*. Для этого подсчитаем функцию *d(i)* — расстояние от первой до *i*-й вершины ломаной. Данный предподсчет может быть выполнен за линейное время проходом по ломаной слева направо.

Возьмем пустой стек *S*. Будем двигаться по ломаной слева направо и заносить вершины в стек *S* по следующему правилу: пока в стеке есть вершины с высотой, меньшей высоты текущей вершины, будем удалять вершины из стека *S*, а затем добавим в стек текущую вершину. Очевидно, что при таком построении стека все вершины в нем будут идти по невозрастанию высоты (рис 3). Таким образом, чтобы проверить наличие в стеке вершин с высотой меньшей высоты текущей вершины, достаточно проверить самую последнюю вершину. Отметим, что после обработки *i*-й вершины в стеке находятся верхние вершины ломаной, которые не закрываются первыми *i* отрезками ломаной при взгляде справа.

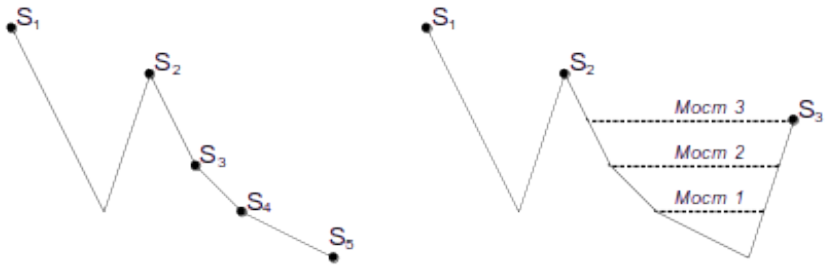


Рис 3. Добавление ребра ломаной и полученные мосты

Пусть в процессе обработки *i*-го отрезка ломаной из стека *S* удаляется вершина с номером *j*, тогда Тогда, если мы проведем мост из вершины *j* вправо, то он упрется в ребро ломаной, ведущей из вершины *i* вниз и влево (мосты 1 и 2 на рис. 3). В тот момент, когда высота вершины, лежащей на вершине стека превысит высоту рассматриваемой вершины, то если мы проведем мост из вершины *i* влево, то он упрется в ребро ломаной, ведущей из вершины, находящейся на вершине стека, вниз и влево (мост 3 на рис. 2).

Если длина моста, построенного таким образом, превышает L, то сдвинем его вниз так, чтобы его длина стала равна L. Это легко сделать из простых геометрических вычислений, основанных на подобии треугольников. Если полученный мост оказался расположен не ниже, чем вершина, лежавшая в стеке сверху от него, то данный мост является допустимым, так как не пересекает ребер ломаной.

В полученном таким образом решении разбираются все случаи, кроме случая гряды пиков одинаковой высоты. Отметим, что данное решение является линейным по количеству звеньев ломаной.

Добавим обработку гряд пиков. Повторим операцию со стеком. Вершины, образующие гряду будут извлекаться из стека последовательно, так что выделить их не представляет труда.

Пусть мы выделили гряду $a_1, a_2, \ldots, a_k$. На этой гряде можно построить $k+1$ сегмент моста: от вершины a_1 влево, от a_1 до a_2, … , от a_{k-1} до a_k и от a_k вправо (причем сегменты мы строим строго на высоте гряды, даже если они при этом получаются длиннее, чем L). Наш мост будет составлен из одного или более подряд идущих сегментов. Заметим, что если нам удалось построить на гряде наилучший мост, то любой его сегмент будет сокращать расстояние между A и B. Значит можно отбросить первые и последние сегменты, которые либо не уменьшают длины пути между A и B, либо имеют длину больше L. Из оставшихся сегментов мост можно конструировать жадно с помощью двух указателей.

Эта часть решения также работает за линейное время.

Эвристики

Большинство эвристик, возможных в этой задаче, как и правильное решение, основаны на жадных алгоритмах.

Рассмотрим несколько подходов, основанных на интуитивно понятных соображениях.

Заметим, что в первую очередь наиболее естественно сажать те типы деревьев, количество которых максимально. Отсортируем a_i в порядке убывания. Рассмотрим p первых чисел $a_1,\ldots, a_p$ и вычтем из них a_p. Это соответствует рассадке a_p групп деревьев вида 1, …, p. Заново отсортируем все a_i и повторим операцию. Будем повторять эту операцию, пока на очередном шаге после сортировки не получим a_p равное нулю. После этого без возникновения противоречий дополнительно можно высадить столько деревьев, сколько в массиве a_i осталось чисел, отличных от нуля.

Заметим, что на каждом шаге одно из чисел массива a обнуляется. Поэтому общее количество шагов алгоритма имеет порядок $O(k)$. На каждом шаге производится сортировка за $O(k \log k)$. Общая сложность алгоритма составляет $O(k^2 \log k)$.

Если заметить, что после каждого вычитания массив разбивается на два невозрастающих участка ($a_1,\ldots, a_p$ и $a_{p+1},\ldots, a_k$), то сортировку за $O(k \log k)$ можно заменить слиянием этих частей за $O(k)$. Тогда общая сложность составит $O(k^2)$.

Для повышения эффективности такого рода эвристики можно также использовать структуру данных «куча». Все элементы a_i нужно хранить в куче, на каждом шаге извлекая из нее p максимальных. После этого уменьшаем все элементы и возвращаем их обратно в кучу. Тогда время работы одного шага алгоритма занимает $O(p \log k)$. Суммарное время работы составляет $O(kp \log k)$.

Для повышения качества эвристики на каждом шаге можно вычитать не a_p, а $a_p/2$.

Такое решение при аккуратной реализации могло набирать около 60 баллов.

Неэффективные правильные решения

Данную задачу можно решать при помощи жадного алгоритма: на каждом шаге выбираем максимальный элемент массива a_i и сажаем дерево соответствующего типа. При этом уменьшаем a_i на один и увеличиваем ответ на один. При выборе максимального элемента рассматриваются только те типы деревьев, которых не было среди последних $p - 1$ выбранных.

Пусть s — сумма всех a_i. На каждом шаге максимум a_i можно выбирать за $O(k)$. Всего шагов не более s. Таким образом, суммарное время работы алгоритма составляет $O(sk)$. Такое решение набирало порядка 40 баллов.

Для повышения эффективности решения можно использовать структуру данных «куча», что позволяет выбирать максимум на каждом шаге за $O(\log k)$. Суммарное время работы в таком случае составляет $O(s \log k)$. Такое решение набирало порядка 60 баллов.

Используя тот факт, что на каждом шаге a_i уменьшается ровно на единицу, для повышения эффективности решения можно использовать следующую структуру данных. Представим массив a в виде набора пар (x, num), где num — число элементов равных x (все x различны). Отсортируем пары по возрастанию x. Дополнительно будем поддерживать очередь последних p посаженных типов деревьев. Чтобы выбрать тип с максимальным числом деревьев, нужно взять последний элемент массива пар (x, num), уменьшить num на единицу и добавить тип с $x - 1$ деревом (одно дерево этого типа мы только что посадили) в очередь. Если num обратился в ноль, удаляем пару из массива.

Теперь, если в очереди p элементов, то нужно взять из очереди первый элемент и добавить его в массив так, чтобы массив остался отсортированным. Утверждается, что новый элемент окажется на последней или предпоследней позиции в массиве. Поэтому эта операция делается за $O(1)$.

Таким образом, на каждом шаге выбор максимума осуществляется за $O(1)$, а суммарное время работы алгоритма составляет $O(s)$. Такое решение получало около 75 баллов.

Эффективные правильные решения

Отсортируем типы деревьев по убыванию их количества, получится массив $a[1] \geq a[2] \geq \ldots \geq a[k]$. Вместо расстановки деревьев в ряд для наглядности будем заполнять таблицу с p строчками (количество столбцов будет зависеть от конкретного случая), первый столбец будет соответствовать расстановке первых p деревьев, второй — от $p + 1$ до $2p$ и так далее. Тогда два соседних дерева одного типа будут в разных столбцах, причем если на соседних — то левый не выше правого (иначе расстояние будет меньше p).

p	2p	..		
..		..		
..		..		¥
3	...	..	¥	
2	p+2	..		
1	p+1	...	Δ	Δ

Первая часть решения — как раз найти длину этой таблицы. То есть найти такое X, что таблицу длины X можно по правилам заполнить имеющимися деревьями, а таблицу длины $X + 1$ — нельзя. Очевидно, что такое число существует, поскольку для $X = 0$ ответ можно, а для $X = [S / p] + 1$ – нельзя (здесь и далее S — это суммарное количество деревьев), и если для X можно, то и для $X – 1$ – тоже. Таким образом, из-за монотонности можно применить двоичный поиск.

Осталось научиться проверять, можно ли заполнить таблицу длины X.

1. Вспомним, что массив a отсортирован по невозрастанию, поэтому первые m чисел там будут больше X, остальные — меньше или равны. Понятно, что любой из этих m типов мы сможем написать в таблицу ровно X раз (можно и меньше, но от этого может быть только хуже). Это будет выглядеть так: i-е дерево полностью заполняет i-ю строчку в таблице (будем заполнять начиная с нижней строки и вверх). Таким образом, m строчек нашей таблицы можно считать заполненными.
2. Утверждается, что если $a[m + 1] \leq X$, то все следующие деревья можно беспрепятственно вставлять в таблицу либо пока она не будет заполнена, либо пока не кончатся деревья. Действительно, будем как в предыдущем случае заполнять таблицу сверху вниз, слева направо. Так как $a[i]$ теперь не более X, то в один столбец два одинаковых дерева попасть не смогут.

Итак, если $S` = a[m + 1] + a[m + 2] + \ldots + a[k]$ больше либо равна $X(p – m)$, то деревья не кончатся, и таблица $p \times X$ будет заполнена. В противном случае деревьев не хватит.

Вторая часть решения состоит в нахождении остатка — сколько клеток помимо найденных pX удастся заполнить. Они состоят из двух частей. Первая — это все типы, количество которых больше X (тогда они будут вылезать за пределы таблицы хотя бы на один). Пусть теперь можно вставить еще L деревьев, количество которых не более X. Тогда оставшихся (то есть без этих L) опять же должно хватать для замощения таблицы. Получаем неравенство: $(S` – L) >= p(X – m)$ (где $S` = a[m + 1] + a[m + 2] + \ldots + a[k]$). Таким образом, максимальное $L = S` – p(X – m)$. Учитывая первые m, получаем итоговый ответ $answer = pX + k + S` – p(X – m)$.

Двоичный поиск будет работать за $O(log\ S)$ умножить на $O(p)$ на каждую проверку (так как смотреть $a[i > p]$ не придется), в итоге $O(p\ log\ S)$.

Можно заметить, что при проверке возможности заполнения таблицы длины X надо сначала найти m (количество $a[i] > X$), а потом проверить, достаточно ли остальных. Первое ищется двоичным поиском, так как массив отсортирован. Если предподсчитать частичные суммы $a[j] + a[j + 1] + \ldots + a[k]$, то второе можно проверять за $O(1)$. Таким образом, проверку на возможность замощения можно реализовать за $O(log\ p\ log\ S)$.

Разбор задачи «Теория цифр»

Участникам была поставлена задача найти наименьшее K-значное число с суммой цифр S, которое после умножения на D дает сумму цифр P.

Рассмотрим сначала частные случаи $D = 1, 2, 5$; в этих случаях, как выясняется, задача имеет асимптотически наиболее быстрое решение.

При $D = 1$ нам дважды дана сумма цифр искомого числа (умножение на 1 оставляет число прежним). Соответственно, если $S \neq P$, то следует сообщить, что такого числа не существует. В противном случае, требуется вывести наименьшее K-значное число с суммой цифр S. Несложно показать, что ответ имеет вид 100..00A99..99 или B99..99, где A и B – некоторые цифры (В $\neq 0$).

Теперь рассмотрим случай $D = 2$. Будем умножать число на 2 «в столбик». При умножении на 2 каждая цифра числа удваивается, и, кроме того, возможны переносы в следующий разряд. Если бы переносов не было, то результирующая сумма цифр P была бы равна $2S$. Каждый перенос превращает 10 единиц одного разряда в 1 единицу следующего, поэтому $P = 2S – 9N$, где N – число переносов. Исследуем теперь, в каких случаях происходит перенос. Если цифра в некотором разряде не больше 4, то после умножения на 2 она будет не больше 8 и, даже если к ней прибавится единица из предыдущего разряда, в следующий разряд перенос не произойдет. Наоборот, если цифра больше либо равна 5, то перенос в следующий разряд точно произойдет. Иными словами, N – число переносов – в точности равно числу цифр от 5 до 9 в данном числе.

Теперь перейдем на время к случаю $D = 5$. В плане суммы цифр умножение на 5 эквивалентно делению на 2. Рассмотрим, как ведут себя цифры числа при делении на 2. Четные цифры делятся пополам, превращаясь в цифры от 0 до 4. Нечетные же цифры также делятся пополам, после чего оставшаяся единица переходит в младший разряд как десятка и прибавляет 5 к цифре, стоящей в том разряде. Поскольку там стояла цифра от 0 до 4, переноса дальше направо не происходит.

Таким образом, если все цифры числа четные, то после деления на два сумма цифр P станет равна $S / 2$. Каждая же нечетная цифра вместо того, чтобы прибавить 0.5 в своем разряде, прибавляет 5 в соседнем, то есть увеличивает сумму цифр на 4.5. Откуда получаем $P = S / 2 + 4.5N = (S + 9N) / 2$, где N – количество нечетных цифр в числе.

Итак, в случаях $D = 2$ и $D = 5$ имеют место «особые» цифры: в одном случае это цифры от 5 до 9, в другом – нечетные цифры. Их число N вычисляется по несложной формуле (при $D = 2$ имеем $N = (2S – P) / 9$, при $D = 5$: $N = (2P – S) / 9$). Если N получилось отрицательным, нецелым или превышающим K, то искомого числа не существует. В противном случае, требуется найти наименьшее число из K цифр с суммой цифр S и ровно N «особыми» цифрами.

Для этого можно предложить общий подход следующего вида. Научимся по тройке $<K, S, N>$ определять, существуют ли K цифр с суммой S, из которых N – «особые». Если «особые» числа – нечетные, это проверяется следующим условием: $N \le S \le N + 8K$, а если «особые» – больше либо равные пяти, то условие выглядит так: $5N \le S \le 5N + 4K$ (убедитесь в этом!). Если быть точным, необходимо также проверить условие $0 \le N \le K$.

Умея проверять такие тройки, минимальное K-значное число находится следующим общим алгоритмом. Переберем все цифры-кандидаты на первую позицию – от 1 до 9. Для каждой из них проверим, можно ли заполнить оставшееся место, если на первую позицию мы поставим именно ее, применив описанные выше условия. Как только подходящая цифра найдена, фиксируем ее и переходим ко второй цифре. Там мы также находим наименьшую цифру (перебирая все кандидаты от 0 до 9), такую, что выбрав ее, оставшиеся места можно заполнить. Ясно, что этот алгоритм найдет наименьшее из всех подходящих K-значных чисел. Время работы этого алгоритма составляет $O(K)$, таким образом при $D = 1, 2$ и 5 задача решается за линейное время.

Разберем теперь общий случай произвольного D от 1 до 9 (кстати, случаи $D = 0$ и $D = 10$ тривиально сводятся к случаю $D = 1$). Для решения задачи используется динамическое программирование.

Поставим следующую задачу: A_{ijk} – минимальное число L, такое что существует L-значное число, сумма которого цифр равна i, а после умножения этого числа на D, сумма L младших цифр произведения равна j, а (L+1)-я цифра произведения равна k.

Научимся вычислять эту трехмерную матрицу. Пусть нам известно значение некоторой ячейки $A_{ijk} = L$, и существует некоторое подходящее L-значное число. Попытаемся приписать к нему слева цифру всеми возможными способами. Новая сумма цифр будет равна i плюс новая цифра, а зная j и k можно вычислить новые значения j и k для получаемого произведения.

При таком переходе из одной ячейки в другую, параметр i не уменьшается, причем, если он остается неизменным (мы приписали слева «0»), то либо строго увеличивается j, либо тройка $<i, j, k>$ вообще не меняется (убедитесь в этом!) Но если она вообще не меняется, то смысла в таком переходе нет: мы уже знали что для тройки $<i, j, k>$ минимальное подходящее число имеет длину L – теперь мы получили информацию, что существует такое число длины $L + 1$, но она совершенно бесполезна. Таким образом, для вычисления матрицы A можно применять динамическое программирование, увеличивая во внешнем цикле параметр i, а во внутреннем – j.

Теперь, когда матрица вычислена, можно приступить к решению задачи.

Для начала выберем k – обозначим так (L+1)-ю цифру произведения A*D. Переберем все варианты от 0 до 9, понятно, что наименьшая цифра в старшем разряде произведения соответствует наименьшему ответу на задачу, поэтому первое же подходящее значение k будет искомым.

Итак, зная k, понятно, что сумма остальных цифр произведения равна $P – k$, то есть текущее состояние соответствует ячейке $A_{S,P-k,k}$. Дальнейший алгоритм таков: как и выше, каждый раз следует перебирать все цифры-кандидаты на очередную позицию – от 0 до 9, или от 1, если речь идет о первой цифре. Сделаем предположение, что определенная цифра окажется на рассматриваемой сейчас позиции. Тогда необходимо проверить, что оставшиеся позиции можно заполнить цифрами, так чтобы все условия были выполнены. Но чтобы это проверить, достаточно обратиться к соответствующей ячейке матрицы A; координаты этой ячейки несложно найти.

Стоит заметить, что здесь используется следующее соображение: если в ячейке A_{ijk} записано число L, то есть существует соответствующее число из L цифр, то существует и число из любого большего числа цифр. Действительно, к существующему числу из L цифр можно приписать нули справа и удовлетворить данные i, j и k. Заметьте, что приписывание нулей слева не доказывает требуемого факта.

Помимо описанного выше квадратичного решения существует кубическое, более простое для понимания и написания. В нем рассматривается матрица A_{ijkL} – минимальное L-значное число с суммой цифр равной i, которое после умножения на D, имеет (L+1)-ю цифру равную k и сумму младших L разрядов равную j. Точнее, следует в этой матрице хранить не само число, а, например, его первую цифру.

Зная значения этой матрицы в слое $L=L_0$, можно вычислить слой $L=L_0+1$, этот переход описан выше и прост в реализации. Когда мы дойдем до последнего слоя, необходимо будет найти ответ на задачу. Ему соответствует ячейка матрицы вида $A_{S,P-X,X,K}$, где X – это (K+1)-я справа цифра числа A*D. Перебрав 10 вариантов X (на самом деле, 9, так как девятки в старшем разряде A*D быть не может), выбирается наименьшее из возможных чисел и выводится как ответ на задачу.

Данный алгоритм имеет временную сложность $O(K^3)$, и его несложно реализовать, хотя некоторые особо неэффективные реализации могут требовать больше, чем предложенные 256 мегабайт памяти, так что следует быть осторожным при выборе хранимых массивов.

Задача 6 Интернет на черный день

Имя входного файла:	`internet.in`
Имя выходного файла:	`internet.out`
Максимальное время работы на одном тесте:	1 секунда
Максимальный объем используемой памяти:	256 мегабайт
Максимальная оценка за задачу:	100 баллов

В городе Шахматовске два интернет-провайдера выполняют план по всеобщей интернетизации страны. Город расположен на бесконечной целочисленной решетке, по всем линиям которой проходят прямые улицы, а единичные квадраты сетки определяют кварталы. Координатами квартала считаются координаты вершины левого нижнего угла соответствующего единичного квадрата. Кварталы города окрашены в черный и белый цвета в шахматном порядке, при этом квартал с координатами (0, 0) окрашен в черный цвет.

Интернет-провайдер «Черный интернет» занимается подключением кварталов черного цвета. Недавно стало известно, что жителям квартала, подключенного K-м, будет предоставлена скидка в 10%.

В соответствии с планом компании «Черный интернет» интернетизация будет проводиться в течение N дней. В i-й день бригада сотрудников компании движется по какой-то из улиц города, начиная из точки (x_i, y_i). Бригада проходит l_i кварталов в заданном направлении. При этом она подключает ранее не подключенные кварталы черного цвета, граничащие по стороне с путем движения бригады (см. рис.).

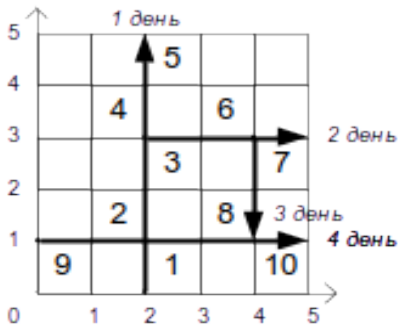


Рисунок к примеру 1

Требуется написать программу, которая определит координаты квартала, подключенного во время реализации плана K-м по очереди. Гарантируется, что в процессе реализации плана будет подключено не менее K кварталов.

Формат входных данных

В первой строке входного файла заданы два целых числа N и K ($1 \le N \le 2\,000$, $1 \le K \le 10^{18}$).

Далее следуют N строк с описанием плана развития компании. В i-й строке описания плана записан путь бригады в i-й день: x_i и y_i ($-10^{15} \le x_i \le 10^{15}$, $-10^{15} \le y_i \le 10^{15}$) — координаты начальной точки пути, символ c_i — направление движения, и l_i ($1 \le l_i \le 10^{15}$) — расстояние, которое пройдет бригада. Направление движения задается одним из следующих символов: «`N`» — север (по увеличению y-координаты), «`E`» — восток (по увеличению x-координаты), «`S`» — юг (по уменьшению y-координаты), «`W`» — запад (по уменьшению x-координаты).

Формат выходных данных

Выведите в выходной файл координаты x и y квартала, подключенного K-м.

Примеры

internet.in	internet.out
4 10 2 0 N 5 2 3 E 3 4 3 S 2 0 1 E 5	4 0
4 7 -1 0 E 4 -1 1 E 4 -1 0 E 4 -1 -1 E 4	0 -2

Разбор

Если бы ограничение на координаты в задаче было бы, скажем, 1000, а ограничение на *K* порядка 1000000, то задача бы легко решалась простым моделированием. А именно, заведем булевский массив покрашенных клеток, и будем моделировать движение бригады в соответствии с инструкциями, данными бригаде по подключению. Решение, основанное на этой идее, получает ? баллов.

Существует также подход, улучшающий это решение, связанный с использованием хэша вместо булевского массива. Таким образом можно избавиться от ограничения координаты, однако остается ограничение на *K* – простое моделирование успеет выполнить только несколько миллионов операций. Решение такого вида получает примерно ? баллов в зависимости от эффективности реализации хэша.

Для того, чтобы решить задачу полностью, можно модифицировать самый первый подход, связанный с использованием булевского массива. А именно, можно заметить, что существует не очень много случаев, когда производится попытка покрасить одну и ту же клетку два раза. Все эти случаи происходят, когда координаты *x* и *y* отличаются не более, чем на 1 от координат начала и конца какого-либо из отрезков движения бригады.

Будем использовать булевский массив для того, чтобы хранить состояние не клеток, а прямоугольников. А именно, положим все “интересные” значения координат в упорядоченные массивы xc[i] и yc[j], и будет считать, что значение в нашем булевском массиве f[i, j] есть истина, если покрашены все черные клетки внутри прямоугольника с координатами углов (xc[i], yc[j]) и (xc[i + 1], yc[j + 1]).

Интересующие нас прямоугольники будут на самом деле полосками, то есть, одна из сторон такого прямоугольника будет всегда равна 1 (вертикальная или горизонтальная, в зависимости от того, горизонтальным или вертикальным является движение бригады.

Теперь установим, какие именно значения координат являются для нас “интересными”. Рассмотрим движение бригады в северном направлении (остальные три случая разбираются аналогично). Ясно, что при этом красятся две полоски: первая – с координатами углов (x[i] - 1, y[i]) и (x[i], y[i] + l), вторая – с координатами углов (x[i], y[i]) и (x[i] + 1, y[i] + l). Эти полоски следует рассматривать независимо, так как они могут быть к моменту движения бригады покрашены по-разному. Таким образом, добавляется три интересных значения x и два значения y. Размерность булевского массива, соответственно, не будет превосходить 3n x 3n.

Теперь опишем решение, использующее такой массив. Массивы со значениями “интересных” координат легко строятся сортировкой с последующим удалением повторяющихся значений. Пусть теперь необходимо выполнить какое-либо движение бригады (например, на север, остальные три случая разбираются аналогично). Найдем позицию координат точек начала и конца отрезка в массиве упорядоченных координат. Пусть это позиции (xs, ys) и (xe, ye). Заметим, что xe = xs, а ye > ys, так как отрезок вертикальный. Будем красить полоски в цикле по возрастанию положения y (обозначим эту переменную j) в массиве упорядоченных координат. На каждом шаге у нас есть три возможных случая:

1) Полоски слева и справа от пути движения бригады уже покрашены – в этом случае никаких новых клеток красить не нужно.
2) Обе полоски не покрашены – в этом случае будут покрашены ровно y[j + 1] – y[j] клеток. Соответственно, если число *K* не больше этого значения, то *K*-я клетка находится именно на этом участке пути, и можно легко найти ее координаты. Иначе уменьшаем значение *K* на количество покрашенных клеток на этом участке и продолжаем выполнение нашего цикла.
3) Одна из полосок покрашена. В этом случае нам понадобится процедура, которая вычисляет количество непокрашенных клеток на оставшейся полоске. Это легко сделать следующим образом. Пусть x-координата этой полоски четна (случай нечетной координаты разбирается аналогично). Тогда, если y[j] нечетно, то обозначим ya = y[j] + 1, иначе ya = y[j]. Аналогично, если y[j + 1] нечетно, то yb=y[j + 1] – 1, иначе yb = y[j + 1]. Легко видеть, что количество клеток, которые

нужно покрасить, равно (yb – ya) / 2 + 1. Аналогично случаю 2), если *K* не больше количества клеток, которые будут покрашены во время движения по этому участку пути, мы находим координаты клетки, иначе уменьшаем значение *K*.

Повторяя этот процесс для всех участков пути движения бригады, и закрашивая в нашем булевском массиве пройденные полоски, мы легко найдем координаты *K*-й закрашенной клетки. По условию такая клетка найдется.

Несложно видеть, что сложность этого решения равна O(n^2).