СОДЕРЖАНИЕ

Выступление Владимира Павлова, директора Международного НИИ проблем программирования INTSPEI

День первый (14.02.08 г.)

Vladimir L. Pavlov «Today's and Tomorrow's ALM Solutions» (International Software & Productivity Engineering Institute) ***vlpavlov@intspei.com***

Abstract

Today new methodologies appear almost as often as new programming languages used to arise 20 years ago. Agile Unified Process (AUP), Design-Driven Development (D3), Dynamic Systems Development Method (DSDM), Test-Driven Development (TDD)… more and more new software development methodologies, approaches and philosophies become available every quarter. This great diversity is an indication that software engineering is not mature yet, it evolves rapidly and we will see even more changes soon.

What is the fundamental trend in the industry? What kind of methodologies will be widely used in 5 years? In 10 years? And, the most important question – how should a software development company pick the most appropriate methodology to get a real competitive advantage?

The keywords in all the answers to these questions are: "productivity engineering". Moore's Law did not double the productivity for human beings. To accomplish that will require a change, not in computers, but in the way human beings use them. The focus of IT now shifts towards human productivity engineering. New software development methodologies will revolutionarily optimize the workflows for the first element in the pair "human + computer".

This presentation will explain how the described trend may impact both domestic and international companies, how to identify the right strategy to benefit from it and how to mitigate associated risks.

День второй (15.02.08 г.).

Контест Виталия Неспирного.

Об авторе…

Неспирный Виталий Николаевич, родился 9 августа 1979 года. С 1986 по 1994 учился в общеобразовательной школе №88 и был постоянным участником олимпиад по математике и физике районного и областного уровня. В 1994 году поступил в лицей при Донецком тогда еще государственном университете. Благодаря знаниям, полученным на спецкурсах Алексея Владимировича Вайсруба и Галины Петровны Кравец, будучи учеником 11-го класса занял четвертое место (диплом третьей степени) в областной олимпиаде по информатике. Это был достаточно высокий результат, поскольку ребята, получившие вторые дипломы на области, заняли соответственно первое и третье абсолютное место на Всеукраинской олимпиаде. Однако, выбрав из республиканских олимпиад математику, получил на ней всего лишь диплом третьей степени. В то же время с работой по построению графических изображений в текстовом режиме выступал на Открытой Российской научно-практической конференции школьников в энергофизическом лицее №1502 при МЭИ, где был отмечен дипломом второй степени. В последствии эта же работа на Всеукраинском конкурсе Малой Академии Наук также была оценена вторым дипломом.

В 1996 году поступил в Донецкий государственный университет и в 2001 году закончил его с отличием. В 1999 году был приглашен в состав жюри областной олимпиады школьников по информатике и практически сразу стал одним из тренеров команды Донецкой области. Среди выступлений в студенческие годы следует отметить победу (первое место) на Всеукраинской олимпиаде по информатике в Николаевском университете кораблестроения (2000г.), организацией которой занимался небезызвестный в украинском олимпийском движении Евгений Юрьевич Беркунский. Но еще более ценной победой является первое место в составе команды математического факультета ДонНУ «Фортуна»

(Неспирный (капитан), Симонов, Вежнин) на первой Всеукраинской АСМ-олимпиаде по программированию (2001 г.), которая проводилась в Винницком национальном техническом университете. На той же олимпиаде в личном зачете занял восьмое абсолютное место. К сожалению, той команде было не суждено попасть ни в полуфинал АСМ-олимпиады в Румынию, ни продолжить свое выступление в последующих украинских олимпиадах, поскольку двое из ее участников были уже студентами 5-го курса и заканчивали университет.

В 2001 году поступил в аспирантуру Института прикладной математики и механики НАН Украины, и одновременно с этим стал тренером команды матфака ДонНУ, которая на протяжении многих лет выходила в финал украинской АСМ-олимпиады и достойно выступала там. Кроме того, был приглашен в лицей при ДонНУ преподавателем спецкурса по решению олимпиадных задач, а в последствии и в лицей Эрудит. Благодаря плодотворной работе, учащиеся занимали высокие места на Всеукраинских и Международных олимпиадах. После незначительного перерыва вновь вернулся в состав жюри школьных олимпиад в Донецкой области и стал главой методической комиссии по разработке заданий.

В 2007 году защитил кандидатскую диссертацию «Стабилизация динамических систем импульсным и гибридным управлением» по специальности теоретическая механика.

В настоящее время работает младшим научным сотрудником отдела технической механики ИПММ НАН Украины, старшим научным сотрудником кафедры теории упругости и вычислительной математики Донецкого национального университета и учителем первой категории в лицее при ДонНУ.

Теоретический материал.

Некоторые задачи вычислительной геометрии.

Задачи вычислительной геометрии требуют для своего решения не только алгоритмической подготовки, но и хорошей математической базы. Необходимы знания из аналитической геометрии, линейной алгебры, методов оптимизации, вариационного исчисления. А кроме того, многие методы решения довольно чувствительны к точности.

Фундаментальное изложение многочисленных результатов в области вычислительной геометрии выполнено в книге [Ф.Препарата, М. Шеймос «Вычислительная геометрия: введение»]. В частности, приведены алгоритмы решения многих задач, а также даны описания специальных структур данных, которые позволяют реализовать эти алгоритмы достаточно эффективно.

Безусловно, данная лекция не претендует на то, чтобы полностью охватить всю тематику. Здесь будут лишь рассмотрены несколько задач, каждая из которых использует методы различных областей математики. Там, где это возможно, будут предложены несколько вариантов решения.

Задача 1. Даны два выпуклых многоугольника: P с L вершинами и Q с M вершинами. Построить их пересечение.

Решение. Существует несколько решений этой задачи, которые имеют различную вычислительную сложность алгоритма и сложность для реализации. Ясно, что пересечением двух выпуклых многоугольников является опять же выпуклый многоугольник. Можно показать, что он будет иметь не более, чем $L+M$ вершин (многоугольник является пересечением полуплоскостей, определяемых сторонами этих многоугольников). Поэтому если мы найдем все его вершины в порядке обхода, то мы решим задачу.

Очевидный метод формирования пересечения состоит в обходе границы одного из многоугольников ребро за ребром, в процессе которого просто и безыскусно определяются все точки пересечения с границей второго многоугольника (не более двух на каждое ребро) и попутно ведется список точек пересечения и вершин. Это потребует затрат времени $O(LM)$, поскольку буден проверено пересечения каждого ребра P с каждым ребром Q. Естественно для сокращения вычислительной работы попытаться использовать факт выпуклости многоугольников.

Другой подход заключается в том, чтобы рассматривать каждый многоугольник как пересечение полуплоскостей. Каждая полуплоскость получается следующим образом: сторона многоугольника продолжается до прямой, которая разбивает плоскость на две полуплоскости, из них выбирается та, в которой лежит многоугольник. Таким образом, N-угольник задается N полуплоскостями. Тогда пересечение многоугольников может быть найдено как пересечение $L+M$ плоскостей. Прямой (brute force) метод потребует также $O(LM)$ операций, однако если воспользоваться методом «разделяй и властвуй», можно добиться сложности $O((L+M)\log(L+M))$. Еще один $O((L+M)\log(L+M))$-алгоритм предложен в статье [Kundu, S., (1987) A new O(n log n) algorithm for computing the intersection of convex polygons. *Pattern Recognition*.20:419-424]. Он основан на сортировке полуплоскостей по углу наклона ограничивающих их прямых и затем использует линейную по времени процедуру для нахождения их пересечения.

Для этой задачи были предложены несколько линейных по времени алгоритмов. Один из них [Shamos M.I., Hoey D. (1976) Geometric intersection problems *Proc. Seventeenth Annual IEEE Symposium on Foundations of Computer Science*. October 1976, pp 208-215] основан на разбиении плоскости, индуцированном пучком лучей, на области в каждой

из которых пересечение можно вычислить легко. Выберем произвольную точку O (эту точку можно взять на бесконечности, тогда лучи будут параллельными прямыми). Из нее проводим лучи через все вершины многоугольников. Эти лучи разбивают плоскость на секторы. Упорядочиваем лучи по углу вокруг точки O (благодаря выпуклости это легко сделать за линейное время). Ключевым является наблюдение, что пересечение каждого сектора с выпуклым многоугольником образует четырехугольник. Поэтому внутри каждого сектора пересечением P и Q будет пересечение двух четырехугольников, которое можно найти за константное время. Результирующие куски можно собрать воедино за один обход секторов, на который понадобится линейное время.

Другой подход [O'Rourke J., Chien C.-B., Olson T., Naddor D. (1982) A new linear algorithm for intersecting convex polygons. *Comput. Graph Image Processing* 19:384-391] является, как это ни удивительно, элегантным усовершенствованием того грубого метода, который был описан в самом начале. Его основная идея следующая. Предположим, что $P \cap \delta Q \neq \varnothing$, и рассмотрим многоугольник $P^* = P \cap Q$. Его границей является чередующаяся последовательность участков границ P и Q. Если одним из таких участков является кусок границы, скажем, P, то некий участок границы Q окружает ее во внешней области P^*. Благодаря выпуклости обоих многоугольников в каждой паре таких смежных участков выделяются внешняя и внутренняя цепи, которые в совокупности называются «серпом». Серп ограничен парой точек пересечения, именуемых начальной и конечной в соответствии с единой ориентацией границ P и Q. Алгоритм проходит одновременно по границам многоугольников P и Q, выделяя внутреннюю и внешнюю цепь для каждого серпа и находя все точки пересечения. На каждом шаге мы продвигаемся к следующему против часовой стрелки ребру многоугольника P либо Q. Главная идея заключается в том, чтобы не двигаться по той границе, текущее ребро которой еще может содержать необнаруженную точку пересечения (при любом положении двух ребер такой выбор может быть сделан). Если при таком движении за $L+M$ шагов не будет обнаружено точек пересечения, то границы многоугольников не пересекаются. И тогда либо один многоугольник содержится внутри другого, либо они не имеют пересечения. Если же будет найдено пересечение границ, то еще не более чем за $L+M$ шагов понадобится для построения многоугольника P^*.

Еще один достаточно простой алгоритм приведен в работе [*Toussaint G. T.* (1985) A simple linear algorithm for intersecting convex polygons]. В отличии от предыдущих алгоритмов, он является комбинацией известных простых процедур для вычисления выпуклой оболочки и триангуляции многоугольников. Сначала строится выпуклая оболочка объединения многоугольников, затем определяются «мосты», т.е. такие ребра в

выпуклой оболочке, концы которых принадлежат разным многоугольникам. Каждому мосту соответствует точка пересечения границ *P* и *Q*, которая может быть найдена с помощью триангуляции многоугольника, ограниченного этим мостом и частями границ *P* и *Q*. Наконец, остается «соединить» найденные точки пересечения соответствующими цепями границ *P* либо *Q*.

Последний алгоритм более прост для реализации, поскольку требует рассмотрения меньшего числа крайних случаев. Однако на практике работает несколько дольше алгоритма О'Рурка.

Задача 2. Плоскость задана уравнением $Ax+By+Cz+D=0$ (*A, B, C, D* – заданные числа). Кроме того, задана таблица *R*[1:8], состоящая из положительных чисел. Определить существует ли куб, расстояние от вершин которого до плоскости – есть элементы таблицы *R*. Если существует, то указать вершины хотя бы одного такого куба. (Все вершины куба должны лежать в одном из полупространств, на которые данная плоскость разбивает все пространство).

Решение. Пусть наименьшее из расстояний *R* равно *d*. Предположим, что можно построить нужный куб. Пусть проекции на перпендикуляр к плоскости ребер, выходящих из ближайшей к плоскости вершины, равны *x, y* и z соответственно. Тогда в массиве *R* должны быть еще и расстояния *d+x, d+y* и *d+z*. Упорядочим их так, чтобы выполнялось $x \le y \le z$. Остальные ребра должны быть параллельны и равны первым трем. А поскольку проекции параллельных и равных между собой отрезков на одну и ту же прямую равны, между собой, то должны быть еще расстояния - *d+x+y, d+x+z, d+y+z, d+x+y+z*. Итак, если можно построить куб, то расстояния, записанные в таблице *R* должны быть представимы в виде *d, d+x, d+y, d+z, d+x+y, d+x+z, d+y+z, d+x+y+z*, где $d>0$ и $0 \le x \le y \le z$. Если это не так, то куб построить невозможно. Напишем пока алгоритм этой проверки:

```
сортировка R
{ проверка }
d:=R[1]
x:=R[2]-d
y:=R[3]-d
temp:=d+x+y
s:="можно построить куб"
выбор
  при temp=R[4]: ;
  при temp=R[5]: R[5]:=R [4]; R[4]:=temp
  иначе s :="нельзя построить куб"
все
z:=R[5]-d
если R[6]≠d+x+z или R[7]≠d+y+z или R[8]≠d+x+y+z то
```

s :="нельзя построить куб"

все

Если бы требовалось построить не куб, а параллелепипед, то можно было бы взять 4 произвольные различные точки на данной плоскости. Затем сдвинуть их на расстояния $R[i]\cdot\frac{(A,B,C)}{\sqrt{A^2+B^2+C^2}}$ (для первой точки – i=1, для второй – i=2 и т.д.). Таким образом мы бы получили координаты ближней вершины (i=1) и еще трех смежных с ней вершин. Дальше все вершины определятся однозначно и формулы для них очевидны.

Несложно решить аналогичную задачу для квадрата и прямой на плоскости. Сторона этого квадрата будет точно определяться таблицей расстояний R и будет равна $\sqrt{x^2+y^2}$, и сам квадрат определяется однозначно с точностью до сдвига вдоль прямой и симметрии относительно прямой перпендикулярной данной.

С помощью аппарата аналитической геометрии и линейной алгебры (задача о нахождении ортогональной матрицы удовлетворяющей уравнению $Xa=b$) можно доказать, что для исходной задачи при выполнении найденного нами условия всегда можно построить куб. Укажем способ построения.

Пусть векторы, выходящие из ближайшей вершины имеют вид $X=(x_1,x_2,x_3)$, $Y=(y_1,y_2,y_3)$ и $Z=(z_1,z_2,z_3)$. Их проекции на перпендикуляр к плоскости соответственно равны x,y и z. Обозначим вектор $n=\frac{(A,B,C)}{\sqrt{A^2+B^2+C^2}}$ – единичный нормальный к данной плоскости. Тогда из условия ортогональности и равенства по модулю векторов X, Y, Z имеем систему: $(X,n)=x$, $(Y,n)=y$, $(Z,n)=z$, $(X,Y)=(X,Z)=(Y,Z)=0$, $(X,X)=(Y,Y)=(Z,Z)=a^2$, где a – сторона искомого куба. Если мы решим данную систему, то дальше построить куб не составит труда. Разделим X, Y, Z, x, y, z на a, тогда будем иметь: $(X',n)=x'$, $(Y',n)=y'$, $(Z',n)=z'$, $(X',Y')=(X',Z')=(Y',Z')=0$, $(X',X')=(Y',Y')=(Z',Z')=1$, где штрихами обозначены величины, которые получаются после деления. Отсюда видно, что X',Y' и Z' образуют ортонормированный базис трехмерного пространства. Следовательно, если записать эти векторы в строки некоторой матрицы, то получим ортогональную матрицу H. Обозначим $p=(x',y',z')$. Из первых трех равенств следует, что $Hn=p$. Таким образом, p – это образ вектора n, к которому применен ортогональный оператор H. Как известно, ортогональный оператор является поворотом (возможно в композиции с симметрией), поэтому векторы n и p имеют одну и ту же длину, а именно 1. Отсюда следует, что $x'^2+y'^2+z'^2=1$ или $x^2+y^2+z^2=a^2$. Т.е. длину ребра куба мы уже определили. Теперь осталось найти хотя бы одну

ортогональную матрицу, удовлетворяющую соотношению $Hn=p$. Если $n=p$, то можно взять тождественный оператор. Если же это не так, то можно взять, например, симметрию относительно плоскости. Эта плоскость должна быть перпендикулярной вектору $p-n$ и проходить через начало координат. Ее уравнение можно записать в виде $(\alpha,t)=0$, где $\alpha = \frac{p-n}{|p-n|}$, а t – точка пространства. Тогда наше преобразование можно записать в виде формулы $Ht=t-2(\alpha,t)\alpha$. Для того, чтобы получить саму матрицу, нужно применить данное преобразование к базисным векторам. Остается только получить координаты ближней вершины. Будем искать ее в виде kn, где k – некоторая постоянная. Проделав все преобразования получим, что $k = d + \frac{D}{\sqrt{A^2+B^2+C^2}}$.

```
a:=sqrt(x^2+y^2+z^2)
p[1]:=x/a; p[2]:=y/a ; p[3]:=x/a
temp:=sqrt(A^2+B^2+C^2)
n[1]:=A/temp ; n[2]:=B/temp ; n[3]:=C/temp
D_:=D/temp
tr:=0
для i от 1 до 3
нц
  temp:=p[i]-n[i]
  tr:=tr+temp^2
  α[i]:=temp
кц
tr:=sqrt(tr)
H :=((1,0,0),(0,1,0),(0,0,1))
если tr≠0 то
  для i от 1 до 3
    α[i]:=α[i]/tr
  для i от 1 до 3
    для j от 1 до 3
      H[i,j]:=H[i,j ]-2*α[i]*α[j]
все
для i от 1 до 3
  для j от 1 до 3
    H[i,j]:=H[i,j]*a
temp :=d +D_
для i от 1 до 3
нц
  точки[1,i ]:=temp*α[i]
```

точки[2,*i*]:=*точки*[1,*i*]+*H*[1,*i*]
точки[3,*i*]:=*точки*[1,*i*]+*H*[2,*i*]
точки[4,*i*]:=*точки*[2,*i*]+*H*[2,*i*]
точки[5,*i*]:=*точки*[1,*i*]+*H*[3,*i*]
точки[6,*i*]:=*точки*[2,*i*]+*H*[3,*i*]
точки[7,*i*]:=*точки*[3,*i*]+*H*[3,*i*]
точки[8,*i*]:=*точки*[4,*i*]+*H*[3,*i*]
кц
все
все

Замечание. Вторая строчка предложенного алгоритма не работает при a=0, но куба с длиной стороны 0 нет. Если же разрешить кубу быть вырожденным, то достаточно пропустить нахождение матрицы H, и найти просто точку на расстоянии d. Все вершины куба будут совпадать с нею. Или во второй строке в алгоритме выбрать произвольным образом единичным вектор p.

Задача 3. Есть N кораблей, которые находятся в точках (x_i,y_i). Каждый из них имеет максимальную скорость (для i-го корабля – положительное v_i), с которой он может двигаться по произвольной траектории. Определить (с точностью до какого-нибудь знака) наименьшее время, которое им потребуется для встречи в одной точке.

Решение. Эта задача может быть решена несколькими способами, каждый из которых представляет интерес.

1. Это решение не требует никакого знания методов оптимизации, и по видимому может быть написано даже школьником. Сразу заметим, что если за некоторое время T корабли могут встретиться, то за любое большее время t ($t\geq T$) они также могут встретиться (двигаясь по тем же самым траекториям, но с пропорционально меньшими скоростями). Аналогично, если за время T все корабли встретиться не могут, то очевидно они не смогут встретиться и за любое меньшее время t ($t\leq T$). Таким образом, для нахождения (с любой наперед заданной точностью) минимального времени для встречи можно применить метод половинного деления, взяв в качестве левого края – 0 (если только все корабли не находятся уже сразу в одной точке), а в качестве правого – некоторое заведомо большое время (например, время, за которое все корабли могут добраться в точку, где находится первый корабль – он будет стоять на месте, а остальные будут плыть к нему по прямой с максимальной скоростью).

Остается лишь научиться определять могут ли корабли встретиться за некоторое фиксированное время T. Для этого мы рассмотрим множество достижимости для каждого корабля. Нетрудно видеть, что i-ый корабль за время T может попасть в любую точку круга с центром в (x_i,y_i) и радиусом

v_iT, и не может попасть ни в какую точку за пределами этого круга. Таким образом, указанный круг является множеством достижимости для i-го корабля. И наша задача сводится к тому, чтобы проверить, есть ли у всех этих кругов хотя бы одна общая точка. Мы можем найти даже все такие точки – они образуют фигуру, которая является пересечением всех этих кругов. Эта фигура является выпуклой (поскольку круг – это выпуклая фигура, а пересечение любого количества выпуклых множеств – выпукло) и имеет границу представляет собой некоторое подобие многоугольника – она имеет некоторое (конечное) число вершин, но соединяются эти вершины не отрезками, а дугами соответствующих окружностей. Имея фигуру, являющуюся пересечением первых i кругов, можно (проверяя пересечение с каждой из ее сторон (дуг) с (i+1)-ым кругом и добавляя новые вершины и стороны) получить фигуру, являющуюся пересечением уже i+1 кругов. Если же на каком то шаге окажется, что очередная окружность не пересекает фигуру, построенную по предыдущим кругам, и лежит вне этой фигуры, то это будет обозначать, что за данное время корабли не смогут встретиться.

Оценим количество вершин, которое будет иметь фигура, являющаяся пересечением N кругов. На первый взгляд может показаться, что их количество очень велико. Окружность может пересекать каждую дугу уже построенной фигуры в двух точках, что приведет к увеличению количества сторон фигуры в 2 раза. И поэтому количество вершин будет расти как 2^N. Однако, несмотря на то, что на одном конкретном шаге количество вершин может увеличиться в два раза, оно не может увеличиваться вдвое на каждом шаге.

Посчитаем количество вершин иначе. Каждая вершина является пересечением двух окружностей (может быть и большего, но двух точно), а две окружности могут иметь максимум две точки пересечения. Таким образом, общее количество вершин не превосходит удвоенного количества пар окружностей, т.е. $2*N(N-1)/2 \sim N^2$.

Описанный алгоритм имеет вычислительную сложность $O(N^3 * (\log T - \log \varepsilon))$, где T – это выбранная в самом начале правая граница времени, а ε – требуемая точность.

Можно несколько упростить этот алгоритм, если не искать границу фигуры-пересечения кругов, а лишь проверять, чтобы хотя бы одна из точек пересечения двух окружностей была общей для всех кругов. Порядок роста вычислительной сложности при этом не изменится, но несколько увеличится константа.

2. В предыдущем решении мы фиксировали время T (одномерную величину) и пытались найти (хотя бы одну) точку (x,y) (двумерная величина), в которую корабли могут добраться за это время. Можно поступить наоборот – зафиксировать точку и посмотреть за какое время мы сможем в нее добраться. Нетрудно видеть, что минимальное время, за

которое корабли могут встретиться в заданной точке определяется по формуле:

$$T(x,y)=\max_{1\le i\le N}\left\{\frac{\sqrt{(x-x_i)^2+(y-y_i)^2}}{v_i}\right\}$$

И тогда задача сводится к тому, чтобы найти глобальный минимум функции двух переменных. Эта функция естественно непрерывна, но вообще говоря не дифференцируема. Множества, на которых эта функция будет дифференцируема – это множества, на которых максимум в формуле для T достигается при некотором фиксированном i. Если бы все скорости были равны, то каждое из этих множеств было бы многоугольником (возможно, неограниченным) со сторонами – отрезками некоторых прямых (есть подозрение, что их все можно построить за время $O(N \log N)$, как и диаграмму Вороного – единственное отличие, что диаграмма Вороного строится по относительно минимума расстояний). В общем случае, такие множества будут ограничены дугами некоторых окружностей, и довольно сложно будет строить множества дифференцируемости. Но, к счастью, этого и не нужно делать.

Каждая из функций стоящих под знаком минимума является вогнутой (т.е. выпуклой вниз). Это можно проверить, непосредственно вычислив гессиан (матрицу вторых производных) и убедившись в ее положительной определенности. Но гораздо проще наглядно заметить, что график этой функции – конус – обладает свойством вогнутости. А поскольку максимум (но не минимум) вогнутых функций также вогнут, то наша функция $T(x,y)$ является вогнутой. Поэтому она может иметь единственный минимум (точки минимума могут образовывать некоторое связное выпуклое множество, но значения функции в них будут совпадать) и он обязательно будет глобальным. Поскольку функция T ограничена снизу нулевым значением, то $T(x,y)$ обязательно имеет минимум. И он может быть найден, например, методом градиентного спуска. В качестве начального приближения может быть выбрана любая точка – например, центр масс (возможно с каким-то весами зависящими от v_i).

Сложность метода градиентного спуска неизвестна, но очевидно, что от N сложность всего алгоритма будет зависеть линейно, поскольку нам требуется $O(N)$ операций для вычисления функции $T(x,y)$, а скорость сходимости градиентного спуска зависит от того, насколько хорошо было начальное приближение к искомому значению и величины градиента функции в точках.

3. Мы можем записать задачу как задачу оптимизации:

Целевая функция:

$f(x,y,T)=T \rightarrow \min$

Ограничения:

$g_i(x,y,T)=(x-x_i)^2+(y-y_i)^2-v_i^2T^2\le 0.$

Несмотря на то, что целевая функция является линейной, задача не является задачей линейного программирования, поскольку ограничения нелинейны. Более того, это задача не является и задачей выпуклого программирования, поскольку функции g_i не являются выпуклыми.

Эта задача может быть названа квадратичной задачей программирования с квадратичными ограничениями, но показано, что эта задача в общем случае NP-трудная.

Тем не менее, можно преобразовать ограничения так, что их правые части станут выпуклыми:

$$g_i(x,y,T)=\sqrt{(x-x_i)^2+(y-y_i)^2}-v_iT\le 0.$$

Таким образом, мы сможем привести задачу к задаче выпуклого программирования. Для ее решения необходимо составить функцию Лагранжа и найти точку ее минимума. Найдем частные производные функции Лагранжа и получим n+3 условия для нахождения n множителей Лагранжа и 3 переменных x,y,T. Далее рассматриваются случаи, когда один из множителей Лагранжа отличен от нуля, два и не менее трех.

Полученные условия будут давать то, что нужно решать задачу для каждых трех кораблей и проверять, чтобы в точку, где оптимальным образом собираются эти три корабля за то же самое время смогли бы добраться и все остальные. Этот алгоритм наиболее прост в реализации, однако его сложность $O(N^4)$, хотя и с достаточно маленькой константой.

Замечание. Рассмотрим небольшую модификацию задачи. Пусть максимальная скорость корабля может быть нулевой. Тогда уже ответ не всегда будет существовать. Но случаи с нулевой скоростью хоть и требуют отдельного рассмотрения, разбираются достаточно просто. Если есть один корабль, который имеет нулевую скорость, то собираться всем кораблям нужно только в его точке. Если есть еще хотя бы один корабль с нулевой скоростью, начальное местоположение которого отличается от первого, то корабли не смогут встретиться вообще.

Задачи и разборы.

Задача A.Чего больше?

Входной файл: divisible.in
Выходной файл: divisible.out
Ограничение по времени: 1 секунда

А это Воронеж, город, где проводится всероссийская олимпиада по информатике. А еще здесь в одной из престижнейших школ работает чуткий и отзывчивый педагог Снежана Денисовна…

У Пети Костылькова день рождения и он принес конфеты, чтобы угостить одноклассников и, конечно же, свою любимую учительницу Снежану Денисовну. Чтобы сделать праздник более увлекательным, Снежана Денисовна предложила пронумеровать конфеты целыми числами между a и b (включая и сами эти числа) так, что каждому числу соответствовала ровно одна конфета. Кроме того, она поведала детям, что те конфеты, номера которых не делятся на некоторое целое число n, наверняка невкусные, и поэтому их она заберет себе. Вкусные же конфеты, то есть такие, номера которые делятся на n, Снежана Денисовна оставит детям.

Учащиеся с урока математики помнят, что число x называется делящимся на y, если существует хотя бы одно целое число k такое, что $x=ky$, но не знают соглашаться ли им на предложенный учительницей вариант или нет. Поэтому они просят вас посчитать каких чисел будет больше – делящихся на n или неделящихся.

Единственная строка входного файла содержит три целых числа a, b, n (каждое число не превосходит по модулю 10^{18}).

Выведите в выходной файл слово "Divisibles”, если между *a* и *b* больше чисел делящихся на *n*, "Indivisibles", если больше неделящихся, или "Equal", если их одинаковое количество.

divisible.in	*divisible.out*
1 4 2	Equal
1 2 3	Indivisibles
10 10 5	Divisibles

Разбор

Эта задача довольно простая, но имеет несколько подводных камней. Прежде всего, следует отметить, что «между числами *a* и *b*» не означает, что $a \le b$, и поэтому в случае, если это неравенство не выполняется, имеет смысл поменять их местами. Тогда мы будем иметь с «числами от *a* до *b*». Заметим, что из определения делимости следует, что если число делится на *n*, то оно делится и на $-n$. А потому в случае отрицательного *n* можно поменять его знак на противоположный.

Для решения задачи при $a \le b$ достаточно посчитать общее количество чисел от *a* до *b* (которое равно $b-a+1$) и количество таких из них, которые делятся на *y*. Нетрудно видеть, что последнее число равно $d(b,n)-d(a-1,n)$, где $d(x,y)$ – количество делящихся на *y* чисел в пределах от некоторого достаточно маленького числа *m* до *x*. Наиболее простая формула получается, если взять $m=0$ (или $m=1$), но тогда нужно уточнить, что обозначает $d(x,y)$ при отрицательном (или неположительном) *x* – количество делящихся на *y* чисел, в пределах от $x+1$ до $m-1$, взятое со знаком минус.

Случай $n=0$ требует отдельного рассмотрения, но обратите внимание, что согласно определению есть ровно одно число, которое делится на 0 – это само число 0.

Есть еще одно красивое решение этой задачи, которое позволяет не определять количество делящихся чисел. Для этого рассмотрим разные случаи для *n*. Любое число делится на 1, поэтому при $n=1$, ответ всегда «Divisibles». При $n=2$ ответ зависит от четности чисел *a* и *b*: если они оба четные, то больше будет четных чисел, если нечетные, то нечетных, а если они имеют разную четность, то будет равное количество четных и нечетных чисел. При $n=3$ ответ, отличный от «Indivisibles» возможно только тогда, когда хотя бы одно из чисел *a* и *b* делятся на 3. При этом, если разность между *a* и *b* равна 1 или 3, то ответ «Equal», а «Divisibles» может быть только при $a=b$. Для значений $n>3$ и для $n=0$, ситуация похожа на случай $n=3$, однако ответ «Equal» получается лишь тогда, когда разность равна 1.

Каждый из описанных алгоритмов имеет вычислительную сложность O(1).

В зависимости от реализации следующий факт может и не понадобиться, но следует знать, что машинное деление с остатком отличается от того, как оно определено в математике. По определению, которое можно взять из [Грехем, Кнут, Паташник «Конкретная математика»] x div y = floor(x/y) (где floor(x) – наибольшее целое, не превосходящее x), а x mod $y = x - (x \text{ div } y) \cdot y$ (хотя x div 0 не определено, тем не менее, x mod 0 считается равным x). Остаток от деления всегда лежит между 0 включительно и y не включительно, и таким образом знак остатка и делителя всегда совпадают. В машинном варианте x div y = trunc(x/y) (где trunc(x) – число, получающееся в результате отбрасывания дробной части, оно равно floor(x) при положительное x и ceil(x) при отрицательном), остаток определяется по той же формуле, что и в математике, но уже с новой трактовкой div. В результате знак остатка всегда совпадает со знаком делимого. Очевидно, что указанные определения совпадают только тогда, когда делимое и делитель имеют одинаковый знак или когда нет остатка, в остальных случаях полученные результаты нужно корректировать.

Задача B. Детали

Входной файл: parts.in
Выходной файл: parts.out
Ограничение по времени: 1 секунда

А это Челябинск, край суровых мужчин. Челябинские мужики настолько суровые, что на контестах по программированию, состоящих из 10 гробовых задач, аксептят по 20 с отрицательным суммарным штрафным временем. Именно здесь на труболитейном заводе №69 работает Иван Дулин, первый в мире фрезеровщик с нетрадиционной сексуальной ориентацией…

Во фрезерный цех челябинского завода поступило N заготовок, которые необходимо обработать на горизонтально-фрезерном станке 6Т83-1. Из одной заготовки получается одна

деталь. Отходы фрезерования, получившиеся при выделке K деталей, могут быть переплавлены в ещё одну заготовку. Начальник цеха Михалыч поручил Ивану Дулину эту работу. Естественно, тот хочет угодить своему начальнику и изготовить как можно больше деталей из поступивших заготовок. Помогите Дулину найти максимальное количество деталей, которые могут быть получены из N заготовок.

Единственная строка входного файла содержит два целых числа N и K ($0 \le N \le 10^{100000}$, $2 \le K \le 10^{10000}$).

В выходной файл нужно вывести сколько деталей можно получить из данных заготовок.

parts.in	*parts.out*
5 8	5
12 6	14

Разбор

Производство деталей может быть реализовано следующим образом. Возьмем все заготовки и обработаем их, получив соответствующее количество отходов. Сгруппируем их по K в одну группу и переплавим, получив новые заготовки. Обработка этих заготовок даст нам еще некоторое количество деталей. Но при этом мы снова получим отходы (да еще плюс те отходы, что не попали ни в одну группу после первой обработки), которые можно снова переплавить. Так, например, при N=36 и K=6 (в такой формулировке эта задача встречается среди математических задач на сообразительность для школьников младших классов) мы получаем сразу 36 деталей, отходы от них переплавляются в 6 заготовок, таким образом, мы получаем еще 6 деталей, отходы от которых образуют еще одну заготовку. Производя эту последнюю деталь, мы получаем общее количество – 43 детали.

Ясно, что мы должны действовать описанным выше способом до тех пор, пока у нас не станет отходов настолько мало, что из них нельзя будет выплавить ни одной заготовки. С каждым разом количество заготовок уменьшается в K раз, поэтому алгоритму, моделирующему этот процесс, потребуется порядка $O(\log_K N)$ операций. Но поскольку данные задачи являются большими числами, то нужно учесть также и время на выполнение деления (наиболее трудоемкой из выполняемых операций). Деление, как известно, выполняется за $O(\log_B N \cdot \log_B K)$, где B – основание системы счисления, в которой производятся вычисления.

Таким образом, сложность алгоритма равна $O(\log_K N \cdot \log_B N \cdot \log_B K) = O((\log_B N)^2)$, что достаточно много при указанных ограничениях.

Организуем производство несколько иначе. Обработаем первоначально все заготовки, получив N деталей и N отходов. Из полученных отходов

выделим один. И будем производить заготовки и детали следующим образом. Возьмем (K−1) из оставшихся отходов и, переплавив их вместе с выделенным отходом, получим одну заготовку. После ее обработки будем иметь еще одну деталь и один отход, который будем считать тем самым выделенным. Таким образом, у нас из N−1 отходов каждые K−1 образуют ровно одну деталь, и мы получаем решение задачи в виде формулы N + (N−1) div (K−1).

Эта формула дает неверный ответ при N=0 (если div понимается в математическом смысле, то при любом K, но даже в случае, когда div понимается в машинном смысле, при K=2 имеем неверный результат), поскольку в этом случае мы не можем выделить один отход. Но этот случай достаточно тривиален.

Вычисление по указанной формуле имеет сложность $O(\log_B N \cdot \log_B K)$ (сложность деления).

Для эффективной реализации деления следует выбрать как можно большее основание B, но не слишком большим, чтобы можно было пользоваться машинной арифметикой. Желательно, чтобы B было степенью 10, тогда ввод и вывод были максимально простыми. Удобно выбрать B=10000, поскольку это наибольшая степень десятки, при которой мы можем пользоваться арифметикой longint (при работе с двухцифровыми чисел). И второй момент – в алгоритме деления в столбик на каждом шаге приходится подбирать цифру частного, как частное от деления (L+1)-значного числа (обозначим его $(u_L,u_{L-1},\ldots,u_0)_B$) на L-значный делитель ($(v_{L-1},\ldots,v_0)_B$). В качестве оценки этого частного можно выбрать число $\tilde{q} = \min\left(B-1, \frac{u_L B + u_{L-1}}{v_L}\right)$. Как показано в [Кнут, «Искусство программирования для ЭВМ», т.2], настоящее частное q лежит в пределах $\tilde{q} - 2 \le q \le \tilde{q}$ при условии, что старшая цифра делителя v_{L-1} не меньше B/2 (этого всегда можно добиться, если домножить делимое и делитель на число B div (v_{L-1}+1)).

Задача С. Бильярд

Входной файл: billiard.in
Выходной файл: billiard.out
Ограничение по времени: 1 секунда

> Скоро список богатейших людей мира будет состоять только из преподавателей вузов, аспирантов и студентов, потому что о нашей стране думают депутаты Пронин и Мамонов из Нефтескважинска…

Даже играя на бильярде, Юрий Венедиктович думает о нашей стране и не может иначе, покуда в ней такое творится. Поэтому ему еще никогда не удавалось выиграть у Виктора Харитоновича. Но он не отчаивается и часто остается в бильярдных клубах после того, как все расходятся, и упорно тренируется. Его любимым является такое упражнение. Депутат оставляет на столе один шар и бьет по нему с очень большой силой, желая добиться того, чтобы шар летал по столу достаточно долго, надеясь, что таким образом он когда-нибудь залетит в лузу. Определите в какую лузу залетит шар после удара Юрия Венедиктовича.

В первой строке входного файла записаны длина стола M и его ширина N ($2 \le M,N \le 1000000$). Во второй строке заданы начальные координаты шара x и y ($0<x<M$, $0<y<N$). Третья строка содержит проекции v_x и v_y вектора скорости, которую приобрел шар после удара ($-1000000 \le v_x,v_y \le 1000000$, $v_x^2+v_y^2 \neq 0$, т.е. хотя бы одна из составляющих вектора скорости отлична от нуля). Все заданные числа – целые.

Лузы находятся в четырех углах стола: в левом нижнем – луза номер 0 с координатами (0,0), в правом нижнем – луза номер 1 с координатами (M,0), в левом верхнем – луза номер 2 с координатами (0,N), в правом верхнем – луза номер 3 с координатами (M,N). Считается, что трение отсутствует (а значит скорость шара не уменьшается во время движения), удары абсолютно упругие (то есть при ударе о борт, касательная составляющая скорости сохраняется, а перпендикулярная – изменяет знак), размеры шара и луз пренебрежимо малы (следовательно, шар попадает в лузу тогда, когда их координаты точно совпадают).

В выходной файл выведите либо номер лузы, в которую попадет шар, либо число −1, в случае, когда шар не сможет попасть ни в одну из луз.

billiard.in	*billiard.out*
3 7 1 3 3 6	3
4 3 1 2 2 1	-1

Разбор

Рассмотрим произвольную траекторию движения шара, только при достижении шаром борта мы будем отражать не вектор скорости относительно нормали, а весь стол относительно борта. Рассмотрев все возможные траектории, мы получим однозначное разбиение всей декартовой плоскости на отражения стола (прямоугольники с координатами $(k_xM,k_yN,(k_x+1)M,(k_y+1)N)$). Отражения луз будут находиться в точках с координатами (k_xM,k_yN), где k_x и k_y – целые числа. Причем лузе 0 будут соответствовать четные значения k_x и k_y, лузе 1 – нечетное k_x и четное k_y, лузе 2 – четное k_x и нечетное k_y, и, наконец, лузе 3 – нечетные k_x и k_y. Траектория же шарика на плоскости будет тогда просто лучом, начинающимся в точке (x,y) в направлении вектора (v_x,v_y). Таким образом, наша задача сводится к тому, чтобы найти самую первую точку вида (k_xM,k_yN), где k_x и k_y – целые числа, на этом луче.

Все точки луча имеют вид $(x+v_x\cdot t,y+v_y\cdot t)$, где $t\geq 0$. Следовательно, имеем систему двух уравнений с тремя неизвестными:

$x+v_x\cdot t=k_xM$,

$y+v_y\cdot t=k_yN$,

где неизвестные k_x и k_y – целые числа, а t – положительное (возможно вещественное) число.

Эта система может иметь много решений, но нас интересует решение с минимально возможным значением t. Сначала найдем все решения (или узнаем, что их нет).

Домножим первое уравнение на v_y и вычтем из него второе, умноженное на v_x, исключая тем самым параметр t:

$k_x\cdot Mv_y - k_y\cdot Nv_x = x\cdot v_y - y\cdot v_x$ (*)

Оставим пока что требование положительности t. Тогда исходная система эквивалентна этому уравнению в следующем смысле: каждое решение системы удовлетворяет (очевидным образом) уравнению (*), а каждому решению уравнения (*) соответствует единственное решение системы (значение t можно получить из того уравнения системы, в котором коэффициент при t ненулевой).

Последнее уравнение является линейным уравнением в целых числах вида $a\,x + b\,y = c$. Если c не делится на d=НОД(a,b), то такое уравнение не имеет целочисленных решений (действительно при любых целых x и y левая часть делится на НОД(a,b)). Если же c делится на d, то необходимо найти хотя бы одно (частное) решение (допустим (x^*,y^*)). Тогда общее решение может быть записано в виде

$x = x^* + r\cdot(b/d)$,

$y = y^* - r\cdot(a/d)$,

где r – произвольное целое число.

Найти же частное решение можно следующим образом: с помощью обобщенного алгоритма Евклида находится решение (x^0,y^0) уравнения $a\,x + b\,y = НОД(a,b)$. Тогда, очевидно $x^*=x^0\cdot(c/d)$, $y^*=y^0\cdot(c/d)$.

Таким образом, мы можем найти все решения (*), из которых нам нужно выбрать такое, которое обеспечивает минимально возможное неотрицательное t. Учитывая, что x находится в пределах $0<x<M$, то из уравнения $x+v_x\cdot t=k_xM$ следует, что при $v_x>0$ – k_x должно быть наименьшим среди возможных решений уравнения (*), не меньших 1, при $v_x<0$ – k_x должно быть наибольшим среди возможных решений уравнения (*), не превосходящих 1 (в случае $v_x=0$, любое решение имеет одно и то же значение k_x и тогда следует рассматривать аналогичные условия по координате y, но при указанных ограничениях (шар не может быть возле борта), мы просто получим, что уравнение (*) не имеет решений).

Следует отметить, что при ограничениях, указанных в условии задачи, значение выражения $x^*=x^0\cdot(c/d)$ может превысить пределы 64-разрядного целого (величины x^0 и c могут достигать порядка 10^{12}) поэтому необходимо реализовать арифметику двойной точности (хотя бы по модулю b/d – как следует из формулы $x = x^* + r \cdot (b/d)$, полученное значение остатка также будет решением).

Приложение. Обобщенный алгоритм Евклида.

Алгоритм Евклида предназначен для нахождения наибольшего общего делителя двух неотрицательных целых чисел a и b (отрицательное число всегда можно заметить соответствующим положительным).

```
m1=a; m2=b;
while (m2)
{
  q = m1 / m2;
  m1 -= q*m2;
  m1 ↔ m2;
}
```

По окончанию алгоритма m_1 станет равным значению НОД(a,b), а m_2 – нулю.

В обобщенном алгоритме Евклида добавляются переменные x_1, y_1 и x_2, y_2 и обеспечивается, что при всех изменениях m_1 и m_2 сохраняются равенства $a\,x_1 + b\,y_1 = m_1$ и $a\,x_2 + b\,y_2 = m_2$:

```
x1=1; y1=0; m1=a;
x2=0; y2=1; m2=b;
while (m2)
{
  q = m1 / m2;
  (x,y,m)1 -= q*(x,y,m)2;
  (x,y,m)1 ↔ (x,y,m)2;
}
```

Как уже отмечалось, по окончанию алгоритма m_1 станет равным значению НОД(a,b), а значит (x_1,y_1) будет решением уравнения $a\,x + b\,y =$ НОД(a,b), причем одно из этих чисел будет положительным, а другое неположительным, к тому же если ни одно из чисел a и b не равно 0, то $|x_1|<b/$НОД(a,b), $|y_1|<a/$НОД(a,b) (а такое решение, если зафиксировать знаки, является единственным).

Также вспомним, что m_2 оказывается нулю, но важнее другое – соответствующие ему значения x_2, y_2 будут равны соответственно $\pm b/$НОД(a,b) и $-(\pm\, a/$НОД(a,b)), то есть коэффициентам при r в общем решении уравнения в целых числах.

Вычислительная сложность алгоритма Евклида (и обычного, и расширенного) при значениях a и b не превышающих N составляет O(log N). Максимальное количество операций достигается тогда, когда a и b – последовательные числа Фибоначчи.

Задача D. Треугольники

Входной файл: triangle.in
Выходной файл: triangle.out
Ограничение по времени: 1 секунда

Москва, как много в этом звуке для сердца русского и нерусского. А это новый элитный район столицы Москва-Сити, в котором трудятся гости из Средней Азии Равшан и Джумшут. Сейчас они делают ремонт в роскошном пентхаусе светской львицы Ксении Собчак…

Хозяйка пентхауса дала Начальнику ремонтной бригады задание нарисовать и закрасить на самой большой стене кальянной два треугольника: белый и черный. Они должны символизировать соответственно концентрацию внутренней энергии и энергии космоса, и поэтому ни в коем случае не должны иметь общих точек. Не вдаваясь глубоко в трактовку мистических символов, Начальник поручил выполнение задания рабочим из Средней Азии

Равшану и Джумшуту. Как обычно, у них оказался свой взгляд на предложенную им работу. Надо отдать им должное в этот раз – они нарисовали действительно два треугольника и даже указанных цветов. Но у Начальника есть серьезные основания опасаться, что треугольники пересекаются. Помогите начальнику определить так ли это, и если так, то сколько вершин имеет многоугольник, закрашенный обоими цветами.

В первой строке заданы координаты вершин белого треугольника x_1, y_1, x_2, y_2, x_3, y_3, а во второй – черного треугольника x_4, y_4, x_5, y_5, x_6, y_6. Все координаты x_i, y_i – целые числа, не превосходящие по абсолютной величине 10^8. Гарантируется, что оба треугольника невырождены.

В выходной файл выведите количество вершин многоугольника-пересечения (считаем отрезок двухвершинным многоугольником, точку – одновершинным, пустое множество – 0-вершинным).

triangle.in	*triangle.out*
-1 -1 1 0 -1 1 0 -2 3 0 0 2	3
0 0 2 0 2 2 0 0 2 2 0 2	2
2 -1 3 0 2 1 -1 -1 1 0 -1 1	0

Разбор

Это несложная задача, но требующая аккуратной реализации. Общее количество вершин результирующего многоугольника лежит в пределах от 0 до 6. Каждая из этих вершин является либо вершиной одного из треугольников, либо внутренней (не совпадающей ни с одним из концов) точкой пересечения непараллельных отрезков, представляющих стороны разных треугольников. Точка пересечения отрезков заведомо является вершиной многоугольника, а вершина треугольника – только в том случае, когда она содержится внутри другого треугольника. Следует лишь позаботиться о том, чтобы при совпадении вершин треугольников не учесть их дважды.

Для решения задачи удобно ввести векторное произведение векторов на плоскости, понимая под ним координату z соответствующего векторного произведения этих векторов, рассмотренных в пространстве (полагая их координату z равной нулю).

Проверка принадлежности точки X треугольнику $A_1A_2A_3$ может быть выполнена следующим образом. Пусть вершины перечислены в порядке обхода треугольника против часовой стрелки, тогда для принадлежности необходимо и достаточно, чтобы векторное произведение вектора A_iA_{i+1} на A_iX должно быть неотрицательным при всех i.

Непараллельные отрезки AB и CD пересекаются тогда и только тогда, когда отрезок AB пересекает прямую CD, а отрезок CD – прямую AB. Пересечение же отрезка AB с прямой CD имеет место тогда и только тогда, когда векторные произведения CD на CA и CD на CB имеют разные знаки.

Другой способ проверки пересечения отрезков, требующий вычисления лишь трех векторных произведений заключается в решении уравнения:

$OA + t_1 AB = OC + t_2 CD$

Левая часть – параметрическое уравнение прямой AB (при $0 \le t_1 \le 1$ – отрезок), а правая – прямой CD.

Перенесем переменную влево, а свободный член OA в правую часть:

$t_1 AB - t_2 CD = AC$.

Домножая это уравнение векторно справа на CD, получим

$t_1 AB \times CD = AC \times CD$,

а домножая векторно справа на AB –

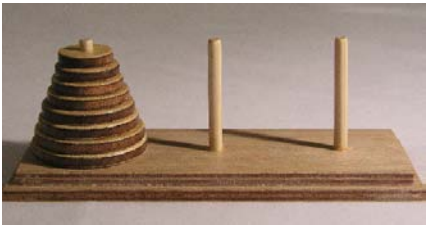
$t_2 AB \times CD = AC \times AB$.

Необязательно находить явно значения t_1 и t_2, чтобы проверить лежат ли они в пределах от 0 до 1 (не включительно, поскольку концы отрезков уже у нас учтены). Более того, при указанных в задаче ограничениях на координаты (10^8) векторные произведения будут иметь порядок 10^{16}, а значит значения t могут иметь порядок 10^{-16}. Поэтому нежелательно использовать в этой задаче арифметику с плавающей точкой.

Задача Е. Ханойские башни

Входной файл: hanoi.in
Выходной файл: hanoi.out
Ограничение по времени: 1 секунда

А это Тибет. Тибетские монахи занимаются общественно-полезным и производительным трудом, перенося диски с башен, и поминают незлым тихим словом Эдуарда Люка, который в 1883 году придумал эту легенду…



В одном из тибетских монастырей монахи уже несколько тысяч лет занимаются перекладыванием дисков. Они располагают тремя башнями, на которых надеты диски разных размеров. В начальном

состоянии все диски были надеты на первую башню и упорядочены по размеру (самый большой диск внизу). Монахи должны переложить все кольца с первой башни на вторую, перемещая только по одному диску за раз. При этом запрещено класть больший диск сверху на меньший. Монахи, не покладая рук, работают и днем, и ночью, перекладывая каждую секунду по одному диску. Как только они закончат свою работу и все диски будут находиться на второй башне, монастырь рассыплется в пыль, грянет гром и мир исчезнет.

Известно, что оптимальное (по количеству перекладываний) решение задачи монахов для каждого количества дисков N единственно и требует 2^N-1 перекладываний.

Вам нужно определить какие диски находятся в данный момент на каждой из башен. К сожалению, сами башни расположены далеко друг от друга и вы не можете просто посмотреть. Но монахи, пробегающие мимо вас за очередным диском, успели на ходу сообщить сколько дисков у них есть всего и сколько секунд прошло от начала их работы. В предположении, что монахи работают действительно оптимальным образом, этой информации вам должно быть достаточно.

В единственной строке входного файла записаны два целых числа – количество дисков N и время M от начала работы до текущего времени ($0<N\le 63$, $0\le M\le 2^N-1$). Размеры дисков определяются целыми числами от 1 до N.

В первую строку выходного файла выведите три целых числа – количество дисков на каждой из башен в момент времени M (то есть в момент, когда выполнено M перекладываний). Во второй строке должны быть записаны размеры дисков на первой башне, перечисленные снизу вверх. В третьей и четвертой строках – аналогичным образом размеры дисков на второй и третьей башнях соответственно. Отметим, что пустой башне должна соответствовать пустая строка.

hanoi.in	*hanoi.out*
3 2	1 1 1 3 1 2
4 14	0 3 1 4 3 2 1

Разбор

Идею задачи предложил Кирилл Симонов.

Вспомним, как строится оптимальная последовательность перекладываний N дисков с башни *src* на башню *dest*, используя башню *aux* в качестве вспомогательной. Чтобы добраться до самого большого диска и переложить его на *dest*, необходимо сначала переложить пирамиду из $N-1$ диска на *aux*, а после того, как диск N займет свое место, нужно будет переложить пирамиду с *aux* на *dest*.

```
Hanoi(N,src,dest,aux)
{
 if (N)
 {
  Hanoi(N-1,src,aux,dest); // перекладывания с номерами от 1 до 2^(N-1)-1
  перекладывание диска N с src на dest // перекладывание номер 2^(N-1)
  Hanoi(N-1,aux,dest,src); // перекладывания с номерами 2^(N-1)+1 до 2^N-1
 }
}
```

Рассмотрим диск с номером N. Он может находиться только на первой или на второй башне. При этом, если количество выполненных перекладываний M меньше 2^{N-1}, то он еще не перемещался и значит находится на первой башне, а выполненные перекладывания касались лишь пирамиды из $N-1$ диска при перемещении с первой башни на третью. Если же M не меньше 2^{N-1}, то диск N уже находится на второй башне, а оставшиеся $M-2^{N-1}$ перекладывания касаются пирамиды из $N-1$ диска при перемещении ее с третьей башни на вторую.

Аналогичные рассуждения дают местоположения всех оставшихся дисков.

Сложность алгоритма – O(N).

Задача F. Высокие ханойские башни

Входной файл: highhanoi.in
Выходной файл: highhanoi.out
Ограничение по времени: 2 секунды

> И снова Тибет. Каждый знает о том, что здесь находятся самые высокие в мире башни и очень большое количество дисков, которые могут быть надеты на них…

Переживая за судьбу нашего мира, жрецы храма Солнца решили усложнить задачу тибетским монахам и увеличили количество дисков. Снова монахи принялись за работу, но высокая пирамида из дисков (да еще и вы с расспросами о том, как долго длится работа) настолько деморализовала их, что они запутались, и сейчас диски расположены в каком-то непонятном порядке, но все еще согласно установленным правилам (больший диск не может находиться сверху на меньшем). Теперь монахи хотели бы собрать все диски хотя бы на какой-нибудь башне, используя как можно меньше перемещений. Помогите им определить на какой башне лучше всего собирать диски и найдите минимальное количество необходимых для этого перемещений.

Первая строка входного файла содержит количество дисков N ($0<N\le 200000$). Во второй строке записаны 3 целых числа s_1, s_2, s_3 ($0\le s_1,s_2,s_3\le N$, $s_1+s_2+s_3=N$) – количества дисков на каждой из трех башен. Строки с третьей по пятую содержат размеры дисков на каждой из башен (соответственно с первого по третий). Диски задаются снизу вверх, поэтому числа в каждой строке упорядочены по убыванию и находятся в пределах от 1 до N. Пустая башня задается пустой строкой. Все диски имеют разный размер.

В первой строке выходного файла необходимо вывести номер башни, на которой следует собирать диски, а во второй – остаток от деления минимального количества необходимых перемещений на число 1000000.

highhanoi.in	*highhanoi.out*
3 1 1 1 3 1 2	1 2
4 0 3 1 4 3 2 1	2 1

Разбор

Задача взята с центрально-европейской олимпиады по информатике 2003 года (авторы (во всяком случае, официальных исходных текстов программ-решений) – Tobias Thierer и Adrian Kuegel).

Это фактически обратная к предыдущей задаче и состоит из двух подзадач: определить номер башни и количество перекладываний.

Первая подзадача довольно простая – собирать пирамиду имеет смысл только на той башне, где исходно находится наибольший диск (диск с номером N) Предположим, что это не так – тогда в какой-то момент нам пришлось бы переместить диск N из его текущего положения на другую башню, которая должна быть пустой. Однако этот ход ничего не меняет (с точностью до перенумерации башен, мы получим ту же самую конфигурацию) и потому может быть исключен из решения.

Официальное решение второй подзадачи, предложенное на CEOI, получается в несколько этапов – сначала строится решение со сложностью $O(2^N)$, затем оно усовершенствуется до $O(N^2)$ и лишь затем получается алгоритм с количеством операций $O(N)$. При этом, окончательное решение использует рекурсию (а значит увеличиваются требования на размер стека), без которой вполне можно обойтись.

Рассмотрим диск с номером $N-1$. Если он уже находится на требуемой башне, то с ним ничего делать не нужно, а требуется лишь переместить диски с $N-2$ по 1 на эту же башню. Если же рассматриваемый диск лежит не на требуемой башне, то нужно будет переместить все диски с $N-2$ по 1 на третью башню (не ту, куда должен быть переложен диск $N-1$, и не ту, где он находится сейчас) и после этого потребуется 2^{N-2} перекладываний (одно перекладывание ($N-1$)-го диска и $2^{N-2}-1$ перекладываний для перемещения пирамиды дисков с $N-2$ по 1). Таким образом, 2^{N-2} перекладываний мы уже сразу можем учесть, и нам останется посчитать количество перекладываний для перемещения дисков с $N-2$ по 1 на третью башню.

Аналогичным образом мы рассматриваем все оставшиеся диски.

Для эффективной реализации следует создать массив, *i*-ый элемент которого будет хранить номер башни, на которой находится диск с номером *i*. Кроме того, имеет смысл выполнить предварительный просчет всех степеней 2, которые будут использоваться в программе (до *N*−2 степени). Все вычисления производятся, разумеется, по указанному в условии задачи модулю.

Задача G. Покупка

Входной файл: purchase.in
Выходной файл: purchase.out
Ограничение по времени: 1 секунда

Первая столица Советской Украины Харьков. Город, который славится своими рынками. А еще здесь в национальном университете радиоэлектроники проводится Зимняя школа по программированию…

Студент, приехавший в Харьков на Зимнюю школу по программированию в составе одной из команд, в свободный от соревнований день отправился на знаменитый рынок возле станции метро «Имени Академика Барабашова», чтобы купить себе теплую куртку. Долго он ходил по рынку от одного продавца к другому, не удовлетворяясь либо качеством, либо ценой. И лишь собираясь уже уходить, он наконец-то заметил в одном из павильонов именно то, что искал. Сторговавшись с продавцом по цене (она составила *N* гривен), студент решил расплатиться с ним таким образом, чтобы общее количество отданных банкнот и полученных в качестве сдачи было как можно меньше. Как это сделать он не знает, но благодаря недюжинным познаниям в программировании и наличию ноутбука, можно написать программу, которая определит оптимальный вариант оплаты.

В обращении находятся банкноты достоинством 1, K, K^2, …, K^M, и у студента и у продавца банкнот каждого номинала имеется в достаточном количестве.

В единственной строке входного файла находятся целые числа K, M, N ($2 \le K \le 100$, $0 \le M \le 100$, $0 \le N \le 2^{63}-1$, $N \le K^M$).

Первая строка выходного файла должна содержать два числа – количество банкнот, которые должен отдать студент продавцу, и количество банкнот, которые он должен получить от продавца в качестве сдачи. Во второй строке должны быть перечислены банкноты, которые отдает студент, а в третьей – которые получает.

purchase.in	*purchase.out*
10 3 123	6 0 100 10 10 1 1 1
2 6 61	1 2 64 1 2

Разбор

Для удобства назовем отдаваемые банкноты положительными, а получаемые – отрицательными, и докажем некоторые свойства, которыми должен обладать оптимальный вариант оплаты.

1. Банкноты одного достоинства в оптимальном решении могут либо все быть положительными, либо все отрицательными.

Действительно, если есть хотя бы одна положительная банкнота и хотя бы одна отрицательная, то они могут быть удалены из решения без потери баланса.

2. Количество банкнот одного достоинства в оптимальном решении не может превышать $K-1$.

Если есть по крайней мере K банкнот одного достоинства (допустим K^i), то они могут быть заменены одной банкнотой достоинства K^{i+1} (быть может, кроме случая, когда эти банкноты имеют максимальное достоинство ($i=M$), но мы увидим дальше, что и в этом случае утверждение остается справедливым).

3. Число M, заданное в условии задачи, является фиктивным параметром в том смысле, что оптимальное решение при выполнении ограничения $N \le K^M$ не изменится, если позволить банкнотам иметь любой номинал, являющийся степенью K.

Предположим, что благодаря увеличению количества банкнот нам удалось построить лучшее решение. Это означает, что существует в этом решении существует по крайней мере одна банкнота достоинства K^i (где

$i>M$). Рассмотрим сумму (с учетом знака) этих банкнот. Она должна делиться на K^{M+1}. Если она равна нулю, то все эти банкноты можно удалить из решения без потери баланса, что противоречит предположению об оптимальности. Если же она не равна нулю, то имеет значение не меньшее K^{M+1}. Баланс же (сумма $K^{M+1}-N$) должен достигаться за счет банкнот достоинством до K^M. Как показано в предыдущем утверждении, каждую из банкнот номиналом до K^{M-1} мы можем взять не более $K-1$ раза. Следовательно, максимальная сумма, которую можно получить за их счет, равна K^M-1. Поэтому, банкнот вида K^M потребуется для баланса не менее, чем

$\text{ceil}(((K^{M+1}-N)-(K^M-1))/K^M)\geq\text{ceil}((K^{M+1}-K^M-K^M+1)/K^M)=\text{ceil}(K-2+K^{-M})=K-1$.

Причем эти банкноты должны иметь противоположный знак. Но тогда мы можем заменить все банкноты достоинства K^i ($i>M$) с суммой K^M и $(K-1)$ банкнот достоинства K^M одной банкнотой K^M, сохранив баланс и уменьшив количество банкнот. Это опять противоречит оптимальности решения с банкнотами больше K^M.

Таким образом, мы можем решать задачу, не обращая внимания на M и считая количество достоинств неограниченным.

4. Оптимальное решение может быть найдено среди тех, у которых количество банкнот одного достоинства не превышает $K/2$ и, кроме того, нет двух подряд идущих достоинств, количество банкнот для которых равно $K/2$.

Пускай у нас есть решение, для которого количество банкнот достоинства K^i равно $c>K/2$ (то есть $c\geq(K+1)/2$) Допустим, что они положительные. Тогда их можно заменить одной положительной банкнотой K^{i+1} и $(K-c)$ отрицательными банкнотами K^i. Тогда сохранив баланс, мы уменьшим общее количество банкнот на величину $c-(1+(K-c))=2c-(1+K)\geq2\cdot(K+1)/2-(1+K)=0$. То есть, количество банкнот как минимум не увеличивается.

Теперь пусть есть решение, для которого банкнот достоинства K^i и K^{i+1} равно $K/2$. Если они имеют один знак (например, положительные), то можно заменить их одной положительной банкнотой K^{i+2}, $K/2-1$ отрицательными банкнотами K^{i+1} и $K/2$ отрицательными банкнотами K^i, не изменив общего количества банкнот. Если же банкноты K^{i+1} положительные, а K^i – отрицательные, то их можно заменить $K/2-1$ положительными банкнотами K^{i+1} и $K/2$ отрицательными банкнотами K^i, уменьшив тем самым общее количество банкнот на 1.

5. Решение, обладающее свойствами, указанными в утверждении 4, определяется для заданного N однозначно.

Доказательство будет конструктивным. Поскольку все достоинства, кроме 1, делятся на K, то количество банкнот 1 с учетом знака (то есть отрицательные должны браться со знаком минус) должно быть сравнимо с N по модулю K. Если остаток от деления N на K отличен от $K/2$, то

количество банкнот и их знак определяется однозначно. Если же остаток будет равен $K/2$, то количество банкнот по прежнему будет определяться однозначно, но их знак будет определен лишь тогда, когда станет известным количество банкнот достоинства K. Изменив соответствующим образом баланс (теперь он будет делиться на K), мы можно перейти к определению количества и знака банкнот достоинства K и т.д.

Следует отметить, что несмотря на то, что N не превосходит пределов знакового 64-разрядного целого ($2^{63}-1$), тем не менее, последнее достоинство может его все-таки превысить. Например, при $K=2$, сумма $2^{63}-1$, представляется в виде 2^{63} и сдачи 1. Вполне мыслима себе ситуация, когда будет недостаточно и беззнакового 64-разрядного числа, для того, чтобы записать степень, однако подобрать такой тест не удалось (он должен достаточно специфическим).

Оптимальный алгоритм решения данной задачи имеет сложность $O(\log_K N)$.

Задача H. С днем рождения!

Входной файл: birthday.in
Выходной файл: birthday.out
Ограничение по времени: 1 секунда

А это Амазонка, река в Южной Америке, величайшая в мире по размерам бассейна и водоносности. Здесь в одном из глухих районов бассейна реки живет племя копельпеков, сохранившее свои традиции и культурно-бытовой уклад, несмотря на европейскую колонизацию…

Племя копельпеков насчитывает K человек. Каждый год они отмечают свои дни рождения (у каждого день рождения происходит в определенный день года, и нет двух человек, которые родились бы в один день). В свой день рождения именинник приглашает всех соплеменников на торжество, на котором они

водят хороводы, курят трубку и едят мясо убитого именинником оленя или другого животного. Естественно, что когда кто-то идет на день рождения к соплеменнику, то должен принести подарок. Со времен прибытия в их края первых бледнолицых волшебников в качестве подарка у них принято дарить друг другу магические круглые предметы, которые точно знают того, кто в них смотрит, и называются бледнолицыми странным словом «зеркальце». Каждый должен принести имениннику (кроме него самого, разумеется) хотя бы одно зеркальце. Но для того, чтобы именинник был доволен и удача не отвернулась от племени, он должен получить в сумме от всех своих соплеменников непременно большее количество зеркалец, чем было подарено предыдущему имениннику в его день рождения. Каждый может подарить часть зеркалец (или даже все), которые подарили в свое время ему. Если же у кого-то не хватает зеркалец, он должен будет обращаться к колонистам, у которых есть очень много зеркалец. Но за каждое из них бледнолицый потребует значительную часть добычи и урожая, которые каждому индейцу нужны и самому (покупать же друг у друга, брать в долг или отдавать просто так зеркальца копельпеки не могут). Поэтому естественно племя стараются брать (если это вообще требуется) как можно меньше зеркалец у бледнолицых. При этом в самом начале у каждого копельпека есть некоторый запас зеркалец.

Помогите племени определить минимальное количество зеркалец, которое им придется приобрести у колонистов, за *N* дней рождения.

birthday.in	*birthday.out*	Комментарий
4 2 0 0 0 0	5	*Чтобы прийти на день рождения к первому, остальные должны приобрести по одному зеркальцу. Тогда у первого получит 3 зеркальца, и до него никто не получал больше. На день рождения второго, первый может подарить 2 зеркальца (и у него останется еще одно), а третий и четвертый должны купить еще по одному зеркальцу. Тогда второй получит 4 зеркальца, что больше, чем было подарено первому. Общее количество покупок равно 5, и меньшим быть не может.*
3 7 5 6 7	0	*Запаса оказывается достаточно, чтобы отметить 7 дней рождения без покупок.*

Первая строка входного файла содержит численность племени K и количество отмечаемых дней рождения N ($2 \le K \le 1000$, $0 \le N \le 100000$). Во второй строке записаны K целых чисел – начальные количества зеркалец у каждого копельпека (первое число соответствует индейцу, у которого будет самый первый день рождения в году). Все эти числа целые неотрицательные и не превосходят 30000.

В выходной файл выведите одно число – наименьшее количество приобретенных зеркалец.

Разбор

Ясно, что первому индейцу в его первый день рождения будет подарено $K-1$ зеркалец. Второму от всего племени не следует дарить больше, чем K зеркалец, так как K ему будет достаточно для того, чтобы ему хватило дожить до следующего своего дня рождения без покупок, а лишнее подаренное зеркало может лишь разогреть аппетит у последующих именинников. Продолжая рассуждения таким образом, мы получим, что в i-ый день рождения должно дариться $K-2+i$ зеркалец.

Рассмотрим теперь каким же образом следует проводить один конкретный день рождения. Мы знаем сколько зеркалец нужно подарить имениннику. Перенумеруем индейцев таким образом, чтобы именинник был первым, вторым был тот, у кого будет следующий день рождения, третьим – тот, у кого день рождения через раз, и так далее. Пусть у них будет соответственно a_1, a_2, a_3 и т.д. зеркалец. Сначала каждый должен отдать хотя бы одно зеркальце, из имеющихся у него. Если же у кого-то их не было, тот должен покупать.

После этого второй может дарить первому хоть все свои оставшиеся зеркальца – следующий день рождения будет у него, поэтому ему не нужен запас, а после дня рождения у него снова будет достаточно много зеркалец. Третьему имеет смысл оставить себе хотя бы одно зеркальце (чтобы обойтись без покупок в день рождения второго), остальные же он может смело отдавать. Четвертому надо оставить как минимум два. И так далее… K-ому индейцу нужно $K-2$ зеркальца, чтобы обойтись без покупок.

Если всего этого не хватило для того, чтобы удовлетворить запросы первого индейца, забираем $(K-2)$-ое зеркальце у K-го индейца (если конечно у него их столько есть). У него останется тогда $K-3$, и это значит, что ему не хватит зеркалец, чтобы поздравить всех своих соплеменников и, в конце концов, ему придется покупать зеркальце. Но случится это нескоро – лишь в день рождения $(K-1)$-го индейца. Если повезет, то N дней рождения (которые требуется обсчитать в задаче) пройдут раньше. Было бы ошибкой забирать зеркальце у кого-либо другого, поскольку тогда делать покупку придется раньше, чем на день рождения $(K-1)$-го.

Далее мы забираем ($K-3$)-ие зеркальца (они могут быть в этот момент только у ($K-1$)-го и K-го). Если и этого не хватило, забираем ($K-4$)-ые зеркальца и т.д.

Если же и после того, как все зеркальца перейдут к первому, а у остальных ничего не останется, все равно не будет хватать зеркалец (такая ситуация можно возникнуть только при $K=2$), кто-либо (совершенно неважно кто именно) идет покупать недостающие зеркальца и приносит их первому.

Описанное решение может быть реализовано в виде алгоритма со сложностью O(NK).

Задача I. Спасите астронавта

Входной файл: astronaut.in
Выходной файл: astronaut.out
Ограничение по времени: 2 секунды

Меркурий, первая планета Солнечной системы. Звездные сутки на Меркурии равны 58,65 земных суток и составляют примерно 2/3 от меркурианского года. Неудивительно, что космических путешественников, ступивших на поверхность планеты, могут ожидать довольно неприятные сюрпризы…

Астронавт, находящийся на поверхности Меркурия, встретил восход солнца. Все бы было ничего, если бы это происходило на Земле, но на Меркурии восход солнца означает, что с каждой секундой теперь поверхность будет нагреваться и плавиться. Чтобы спастись, астронавту следует как можно скорее попасть на свой корабль. Помогите ему сделать это.

Поверхность Меркурия представляется плоским прямоугольником, разбитым на $M \times N$ равных квадратных участков, температура в каждом из которых

примерно одинакова. За единицу измерения температур выбран меркурианский градус (ровно на один такой градус в секунду увеличивается температура в каждой точке поверхности). Скафандр космонавта рассчитан на диапазон температур от T_1 до T_2 меркурианских градусов включительно. В начальный момент времени астронавта находится в клетке с координатами (i_1,j_1), а его корабль – в (i_2,j_2).

За секунду астронавт может переместиться в одну из соседних клеток или может подождать эту секунду в текущей клетке. Он не может покидать границы прямоугольника. Кроме того, в любой момент времени температура поверхности в точке, где находится космонавт, не может быть больше T_2 или меньше T_1. Есть только одно исключение из этого правила – как только астронавт достигнет клетки (i_2,j_2), он сразу же забирается в свой космический корабль. Поэтому астронавт может переместиться в эту клетку из соседней в любой момент, независимо от температуры клетки (i_2,j_2).

В первой строке входного файла находятся размеры прямоугольника M, N ($1 \le M,N \le 1000$), во второй строке – числа T_1 и T_2 ($0 \le T_1 < T_2 \le 1000000$), определяющие диапазон температур для скафандра. Третья строка содержит числа i_1, j_1, i_2, j_2 ($1 \le i_1,i_2 \le M$, $1 \le j_1,j_2 \le N$) – координаты астронавта и корабля (гарантируется, что они не совпадают в начальный момент времени). В последующих M строках записано по N чисел, определяющих начальные температуры в соответствующих клетках (гарантируется, что клетка, в которой первоначально находится астронавт, имеет допустимую температуру). Все числа целые и находятся в диапазоне от 0 до 1000000.

astronaut.in	*astronaut.out*
4 3 20 30 2 2 4 3 19 17 23 10 28 25 11 25 29 12 13 10	15 ENHWWHHHHHSSSEE
2 3 10 20 1 1 1 3 15 3 15 17 14 0	-1

В выходной файл необходимо вывести минимальное время, за которое астронавт может добраться до корабля, либо число -1, если астронавт обречен. В первом случае во второй строке должно содержаться описание оптимального пути. Это описание состоит из нескольких приказов,

которые последовательно должен исполнить астронавт. Каждый приказ – это один из символов N (переместиться в соседнюю клетку на север), S (на юг), W (на запад), E (на восток) или H (ждать 1 секунду). По отношению к клетке (i,j) соседней северной является клетка $(i-1,j)$, южной – $(i+1,j)$, западной – $(i,j-1)$, восточной – $(i,j+1)$. Приказы не следует разделять пробелами или переводами строк.

Разбор

Поверхность планеты можно представить в виде ориентированного графа, вершинами которого являются клетки, а ребрами переходы из одной клетки в соседнюю. Тогда задача сводится к задаче нахождения кратчайшего пути и решается с помощью алгоритма Дейкстры. Единственное, что следует отметить – веса ребер (время, необходимое для перехода из одной клетки в другую) нам заранее неизвестны и определяются лишь тогда, когда мы знаем уже длину пути в соответствующую клетку.

Допустим мы в момент времени l попали в клетку с координатами с начальной температурой T_b. Тогда в соседнюю клетку с начальной температурой T_e мы можем попасть лишь в такой момент времени k, который удовлетворяет условиям:

$k \geq l+1$

$T_1 \leq T_b+k \leq T_2$

$T_1+1 \leq T_e+k \leq T_2$

Отметим в последнем неравенстве +1 слева, поскольку при входе в очередную клетку в ней температура еще не достигает значения T_e+k.

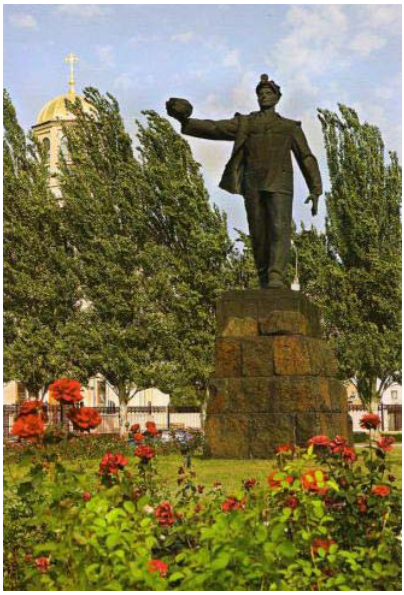
Стандартная реализация алгоритма Дейкстры имеет сложность $O(V^2)$, где V – количество вершин. Поскольку вершин (клеток) $V=MN$ у нас может быть достаточно большим, это не приемлемо. Если же для хранения множества вершин с еще неокончательными оценками длины пути использовать очередь с приоритетами (реализуемые с помощью кучи) количество операций станет $O(E \log V)$, где E – количество ребер. В нашем случае $E \sim 4V$, поэтому такая реализация алгоритма Дейкстры будет работать гораздо быстрее.

Задача J. Тоннель

Входной файл: tunnel.in
Выходной файл: tunnel.out
Ограничение по времени: 1 секунда

А это Донецк. Город миллиона роз, из которых сейчас осталось гораздо меньше. Родной город

футбольного клуба «Шахтер», ежегодно планирующего выиграть Лигу Чемпионов. Однако в последнее время Донецк трагедиями на угольных шахтах…



Шахтер, работавший в одной из горных выработок шахты Трудовская, получил серьезную травму. Можно было бы вызвать медицинскую бригаду, если бы в очередной раз не произошло плановое отключение электроэнергии, поэтому врачи не могут спуститься в шахту. Однако в соседнем забое трудится товарищ травмированного, который может оказать первую медицинскую помощь. Для этого второму шахтеру придется прорубить тоннель в породе между двумя выработками, но поскольку этот процесс трудоемким и требует большого количества усилий и времени, тоннель этот должен быть как можно меньшей длины. Найдите эту длину.

Каждую выработку, равно как и тоннель, будем считать прямолинейным отрезком в трехмерном евклидовом пространстве, пренебрегая их поперечными размерами.

В первой строке входного файла даны координаты начала x_{1b}, y_{1b}, z_{1b} и конца x_{1e}, y_{1e}, z_{1e} первой выработки, во второй аналогичным образом задаются концы второй выработки x_{2b}, y_{2b}, z_{2b}, x_{2e}, y_{2e}, z_{2e}. Все числа – вещественные, имеют не более двух знаков после десятичной точки и не превосходят по абсолютной величине 100.

В выходной файл необходимо вывести одно число – длину минимально возможного тоннеля между двумя выработками с точностью не менее 10^{-6}.

tunnel.in	*tunnel.out*
1 1 -1 1 -1 1 -1 1 1 -1 -1 -1	2.000000
1 2 3 4 5 6 0 0 0 4 6 8	0.000000

Разбор

Ясно, что тоннель также должен быть отрезком, концы которого лежат на заданных отрезках (любую кривую, соединяющую две точки, можно всегда заменить отрезком, соединяющим те же точки и не имеющим не большую длину).

Приведем одно из возможных решений задачи. Пусть нам заданы отрезки AB и CD. Возьмем точку X_1 на отрезке AB и точку X_2 на отрезке CD. Тогда вектор AX_1 коллинеарен AB и значит может быть записать в виде $AX_1 = t_1 AB$, где $0 \le t_1 \le 1$ (поскольку векторы сонаправлены и длина AX_1 не превышает AB). Аналогично $CX_2 = t_2 CD$, где $0 \le t_2 \le 1$.

Тогда вектор X_1X_2 может быть выражен как

$X_1X_2 = X_1A + AC + CX_2 = -t_1 AB + AC + t_2 CD$.

Возьмем функцию $f(t_1,t_2)$ равную квадрату длины вектора X_1X_2 (знаком «·» обозначено скалярное произведение):

$f(t_1,t_2) = t_1^2 AB^2 + t_2^2 CD^2 + AC^2 - 2t_1t_2\, AB \cdot CD - 2t_1\, AB \cdot AC + 2t_2\, AC \cdot CD$

Тогда наша задача сводится к нахождению наименьшего значения этой функции на квадрате $0 \le t_1,t_2 \le 1$. Это значение будет достигаться либо в точке локального минимума (поскольку функция выпукла (вниз), то локальный минимум является глобальным для функции, другое дело, что он может находиться вне квадрата $0 \le t_1,t_2 \le 1$), либо на одной из границ ($t_1=0$, $t_1=1$, $t_2=0$ или $t_2=1$).

Точку локального минимума $f(t_1,t_2)$ легко найти, если приравнять нулю частные производные по t_1 или t_2. Получится система из двух линейных уравнений – если ее определитель равен нулю, то минимумы образуют прямую (задаваемую любым из уравнений). Пересекает ли эта прямая квадрат или нет, но наибольшее значение в этом случае следует искать на границе квадрата $0 \le t_1,t_2 \le 1$. Так же нужно переходить к границе, если получившаяся точка минимума будет лежать вне квадрата.

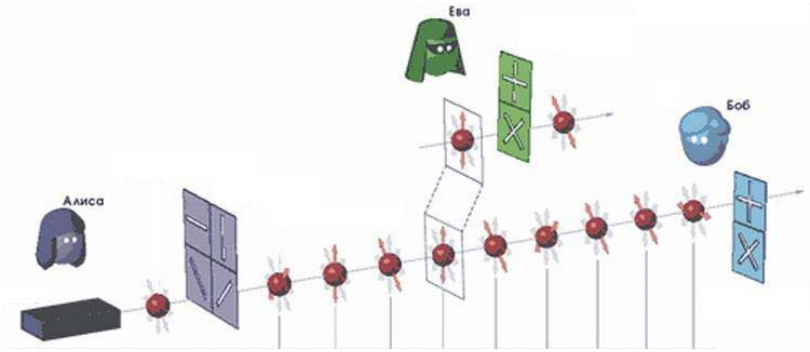
На границе (с фиксированным значением t_1 или t_2) наша функция становится функцией одной переменной, минимум которой найти еще проще.

Сложность алгоритма – O(1).

Задача K. Шифрование

Входной файл: ciphering.in
Выходной файл: ciphering.out
Ограничение по времени: 1 секунда

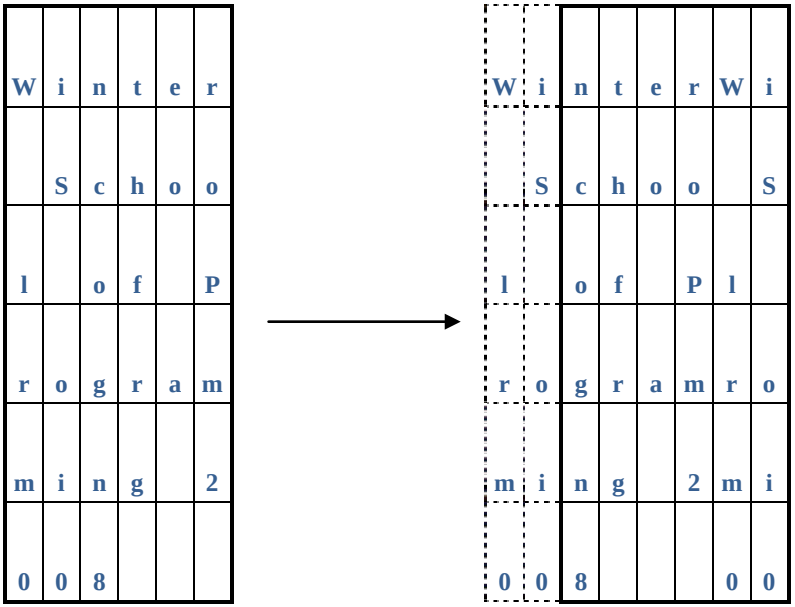
Соединенные Штаты Америки, страна, в которой родился Билла Гейтс и создал операционную систему Windows. А еще здесь, если верить книгам по криптографии, живут Алиса, Боб и Ева. Всю свою сознательную жизнь Алиса шифрует все новыми и новыми способами сообщения и отправляет их Бобу, Боб получает и расшифровывает, а Ева перехватывает сообщения Алисы…



Алиса придумала новый способ шифрования сообщений и написала для него программу. Сообщение шифруется следующим образом – в матрицу размера $N \times N$ (достаточно большую) построчно записываются символы сообщения (то есть сначала заполняется первая строка слева направо, затем аналогичным образом вторая и т.д.). Незаполненные элементы матрицы заполняются пробелами. И затем выполняет несколько команд, из следующего списка, которые переставляют элементы матрицы:

1) L i, где целое число i лежит в пределах от 1 до N-1 – элементы матрицы циклически сдвигаются на i позиций влево.

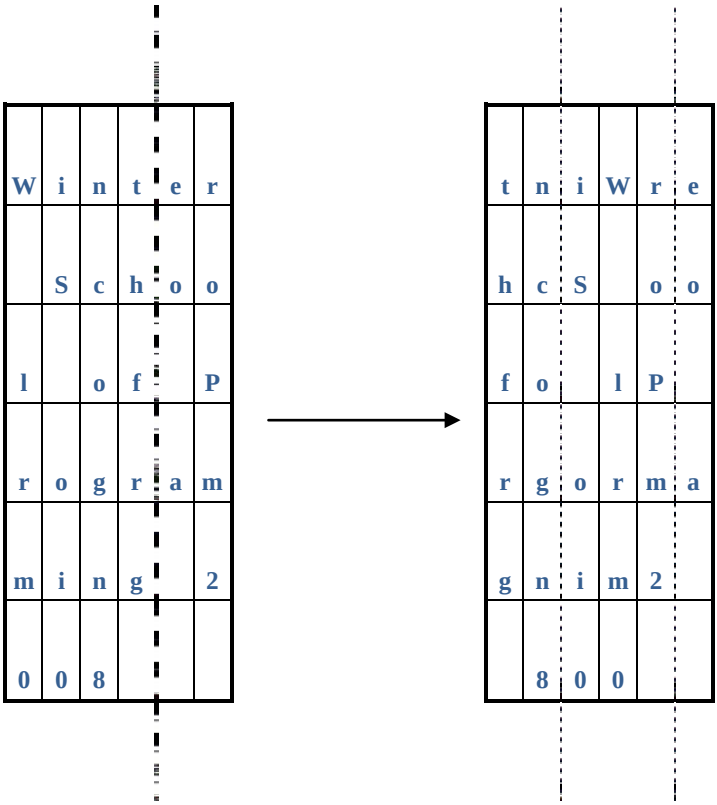
Например, при выполнении команды L 2 элементы переставляются следующим образом.



Аналогичным образом выполняются команды R *i*, U *i*, D *i*, обозначающие циклические сдвиги на *i* позиций вправо, вверх и вниз соответственно.

2) V *i*, где целое число *i* лежит в пределах от 1 до *N*/2. Если *i*=*N*/2, то элементы матрицы отражаются симметрично относительно прямой, разделяющей столбцы с номерами *N*/2 и *N*/2+1. Если же *i*<*N*/2, то матрица делится на две части прямой, разделяющей столбцы с номерами 2*i* и 2*i*+1, первая из которых отражается симметрично относительно прямой, разделяющей столбцы с номерами *i* и *i*+1, а вторая – относительно прямой между столбцами *N*/2+*i* и *N*/2+*i* +1 (при четном *N*) или прямой проходящей вертикально через столбец с номером (*N*+1)/2+*i* (при нечетном *N*).

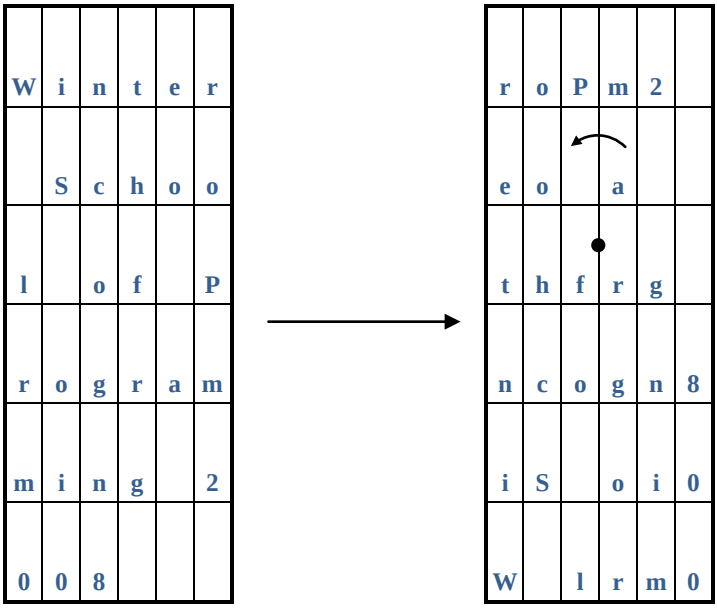
Например, при выполнении команды V 2 элементы переставляются следующим образом.



Аналогичным образом выполняется команда H *i* (эта команда выполняется так, как указано в определении команды V с заменой столбцов на строки).

3) P *angle*, где *angle* является одним из чисел -270, -180, -90, 0, 90, 180 или 270 – матрица поворачивается на указанный угол в градусах против часовой стрелки относительно своего центра.

Например, при выполнении команды P 90 элементы переставляются следующим образом.



4) T – элементы матрицы симметрично отражаются относительно главной диагонали (транспонирование).

После того, как все команды будут выполнены, символы построчно выписываются из матрицы и получается зашифрованное сообщение, которое Алиса посылает Бобу.

Боб не умеет работать с матрицами, но у него есть написанная Алисой программа для шифрования сообщений. Он надеется, что, запуская несколько раз подряд эту программу, ему удастся расшифровать любое сообщение Алисы. Но поможет ли это и если да, то сколько раз нужно запускать программу, Боб не знает.

Первая строка входного файла содержит размер матрицы N ($1 \le N \le 10^9$) и количество M команд перестановок элементов матрицы в программе Алисы ($0 \le M \le 10^5$). В последующих M строках записаны сами команды в указанном в условии задачи формате.

В выходной файл необходимо вывести наименьшее количество раз, которое должен запускать Боб программу шифрования, чтобы получить сообщение Алисы. Если же расшифровать сообщение не удастся, выведите число −1.

ciphering.in	*ciphering.out*
6 1 L 2	2
3 3 P 90 T P 90	1

Разбор

Отметим, что наименьшее количество операций может быть равно 0 (в том случае, когда программа Алисы в конечном итоге приводит к тому же самому сообщению).

Для решения задачи можно заполнить матрицу размера $N \times N$ различными значениями (например, числами от 1 до N^2). Произвести с ней один раз весь набор команд, заданных в файле, и затем выполнять этот набор команд до тех пор, пока матрица не совпадет с начальным значением. В результате мы получим алгоритм со сложностью $O(LMN^2)$, где L – ответ задачи. При заданных в условии ограничениях, это будет невероятное большое число. Кроме того, в случае невозможности расшифровки, такой алгоритм просто зациклится. К счастью, как мы увидим дальше, расшифровать сообщение всегда возможно.

Уменьшить сложность можно следующим образом. После выполнения M команд, мы получим некоторую перестановку p чисел от 1 до N^2, и в дальнейшем вместо повторного выполнения всего набора команд достаточно выполнить всего лишь эту самую перестановку p. В результате мы выполним $O((L+M)N^2)$ операций, что по-прежнему достаточно много.

Нетрудно заметить, что задача эквивалентна такой – найти наименьшее неотрицательное L, такое, что $p^{L+1}=e$, где e – тождественная перестановка. Такая степень всегда существует (а значит ответ всегда отличен ~~от~~1) и может быть определена как наименьшее общее кратное длин циклов перестановки. Разложить перестановку длины N^2 на циклы можно за $O(N^2)$. Для нахождения НОК двух чисел (с помощью алгоритма Евклида) порядка $O(\log N)$ операций. Но нам еще необходимо затратить время $O(MN^2)$ для нахождения самой перестановки. А потому общая сложность алгоритма будет определяться величиной $O((M+\log N)N^2)$.

Заметим, что каждая команда определяет некоторое преобразование матрицы. Удобно его записать не для самой матрицы, а для позиций элементов. Так, например, при преобразовании L k элемент, находившийся в позиции (i,j), перемещается в позицию $(i,j-p)$ (все операции выполняются по модулю N). Все преобразования как нетрудно видеть являются аффинными (и даже сохраняющими метрику в пространстве Z_n^2),

а значит можно каждому из них поставить в соответствие матрицу размера 3×3 над полем целых чисел Z_n такого вида

$$\begin{pmatrix} 1 & 0 & 0 \\ p_x & k_{xx} & k_{xy} \\ p_y & k_{yx} & k_{yy} \end{pmatrix}.$$

Действие преобразования будет соответствовать умножению вектора $(1,i,j)^T$ слева на соответствующую матрицу.

Элементарным преобразованиям (командам) соответствуют такие матрицы:

L p	$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -p & 0 & 1 \end{pmatrix}$
R p	$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ p & 0 & 1 \end{pmatrix}$
U p	$\begin{pmatrix} 1 & 0 & 0 \\ -p & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$
D p	$\begin{pmatrix} 1 & 0 & 0 \\ p & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$
V p	$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 2p & 0 & -1 \end{pmatrix}$
H p	$\begin{pmatrix} 1 & 0 & 0 \\ 2p & -1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$

P 0	$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$
P 90 P -270	$\begin{pmatrix} 1 & 0 & 0 \\ -1 & 0 & -1 \\ 0 & 1 & 0 \end{pmatrix}$
P 180 P -180	$\begin{pmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -1 \end{pmatrix}$
P 270 P -90	$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ -1 & -1 & 0 \end{pmatrix}$
T	$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix}$

Матрицы указанного вида образуют группу. Следовательно, преобразованию для всего набора команд соответствует матрица того же вида и равная произведению матриц, соответствующих элементарным преобразованиям, и оно может быть найдено за O(M) операций. Чтобы найти наименьшую степень (L+1) этого преобразования, дающую тождественное, потребуется выполнить L умножений матрицы на саму себя, то есть O(L) операций. Общая сложность алгоритма будет составлять O(M+L). Поскольку, величина L достигать значения 2N, данный алгоритм так же может не уложиться в отведенное время.

Можно заметить, что подматрица $\begin{pmatrix} k_{xx} & k_{xy} \\ k_{yx} & k_{yy} \end{pmatrix}$ результирующего преобразования определяет вращение (возможно несобственное, то есть с симметрией) плоскости Z_n^2 на один из углов 0, 90, 180, 270, поэтому видов

данной подматрицы существует всего 8: $\begin{pmatrix}1 & 0\\0 & 1\end{pmatrix}$, $\begin{pmatrix}1 & 0\\0 & -1\end{pmatrix}$, $\begin{pmatrix}-1 & 0\\0 & 1\end{pmatrix}$, $\begin{pmatrix}-1 & 0\\0 & -1\end{pmatrix}$, $\begin{pmatrix}0 & 1\\1 & 0\end{pmatrix}$, $\begin{pmatrix}0 & -1\\1 & 0\end{pmatrix}$, $\begin{pmatrix}0 & 1\\-1 & 0\end{pmatrix}$, $\begin{pmatrix}0 & -1\\-1 & 0\end{pmatrix}$.

Для каждого из видов подматрицы вращения можно явным образом вычислить нужный нам показатель степени L, дающей единичную матрицу. В некоторых случаях потребуется нахождения наибольшего общего делителя, поэтому алгоритм будет иметь сложность $O(M+\log N)$.

Важное замечание – случаи $N=1$ и $N=2$ в последних решениях требуют отдельного рассмотрения, поскольку для них $-1\equiv 1 \pmod N$.

День третий (16.02.08 г.).

Контест Александра Рыбака.

Об авторе…

Студент 6 курса Киевского Политехнического Института, Физико-технический факультет. Занимается математикой, большое внимание уделяет комбинаторике и алгоритмическим задачам.

Достижения.

Дважды (2004, 2006) занимал первое место на Международной Математической олимпиаде для студентов. В 2005 году занял второе место на этой же олимпиаде.

Пять раз (2002-2006) участвовал в Региональной олимпиаде ACM по программированию (юговосточноевропейский регион). В 2006 году занял певрое место (в команде с Кириллом Веденским и Владимиром Гигиняком).

Два раза (2003, 2007) участвовал во Всемирной олимпиаде ACM по программированию. (В 2003 году: в команде с Антоном Меллитом и Владимиром Ткачуком, заняли 30 место из 72, в 2007 году: в команде с Кириллом Веденским и Владимиром Гигиняком, заняли 26 место из 88.)

Задачи и разборы.

Материалы прилагаются в электронном виде на диске.

День четвертый (17.02.08 г.).

Контест Евгения Билецкого.

Материалы прилагаются в электронном виде на диске.

День пятый (18.02.08 г.)

Контест Андрея Лопатина.

Об авторе …

Лопатин Андрей Сергеевич.

Старший преподаватель кафедры системного программирования математико-механического факультета СпбГУ.

Серебрянный призер IOI в 1997 и 1998.

Член команды СПбГУ чемпиона мира финалов ACM 2000 и 2001.

Финалист TCCC Marathon Match 2007.

Тренер команд программистов СпбГУ.

Тренер школьной сборной России по информатике.

Задачи и разборы.

Задача A. Экзамен по вождению

Имя входного файла:	exam.in
Имя выходного файла:	exam.out
Ограничение по времени:	1 секунда
Ограничение по памяти:	256 мегабайт

Царь Вася решил улучшить сообщение между городами своей страны. Вместо того, чтобы строить новые дороги, он взялся обучить

своих граждан водить автомобиль. Недолго думая, Вася решил, что экзамен по вождению будет состоять в проезде через несколько параллельных ворот.

Чтобы составить план экзамена, Вася решил изобразить ворота на координатной плоскости в виде отрезков. Для простоты он решил, что все они будут вертикальными.

Жители Васиной страны не любят ничего нового, и, в частности, им не нравится ездить на автомобиле. Больше всего они боятся поворачивать. Поэтому жители согласны ехать во время экзамена между воротами только по прямой. Чтобы как-то пройти Васин экзамен, они смирились с необходимостью поворачивать непосредственно при проезде через ворота.

С точки зрения жителей Васиной страны опасность поворота — это величина угла, на который надо повернуть. Опасность всего пути — это сумма опасностей поворотов (в радианах), сделанных на нём. Поскольку жителей много, а Вася — один, ему не удалось сохранить в тайне план предстоящего экзамена. Считая, что автомобиль представляет из себя точку на плоскости, помогите бедным людям найти самый безопасный путь через ворота.

Формат входного файла

В первой строке входного файла находится единственное натуральное число $N(2 \le N \le 100)$. количество ворот на экзамене. Далее следуют N строк, по три целых числа x_i, a_i, b_i в каждой

($-10000 \le x_i \le 10000$, $-10000 \le a_i \le b_i \le 10000$) i-ые ворота представляют собой отрезок между точками (x_i,a_i) и (x_i,b_i).

Ворота на экзамене следует пересекать в порядке появления их во входном файле. Известно, что числа *Xi* идут по возрастанию.

Формат выходного файла

В первой строке выходного файла выведите вещественное число — опасность самого безопасного пути через ворота. Во второй строке выведите JV вещественных чисел — координаты точек (по оси *y),* в которых автомобиль должен пересекать ворота согласно самому безопасному пути. Вещественные числа требуется выводить не менее, чем с шестью точными знаками после десятичной точки.

Если таких путей несколько, выведите любой из них.

Считайте, что автомобиль может безопасно пересечь первые и последние ворота в любой их точке с любым направлением. При движении автомобиль может касаться края ворот.

Примеры

exam.in	exam.out
2 1 1 2 2 3 4	0 1.5 3.5
3 1 1 2 2 3 4 4 1 2	1.2490457723982544 2 3 2

Разбор задачи «Экзамен по вождению»

Главная идея решения — все изломы пути находятся на границах ворот. Это ясно из того, что если у нас есть излом, то мы его можем «распрямлять», пока не распрямим или не упремся в границу. При этом суммарный излом может лишь только уменьшиться. После этого наблюдения решение уже не представляет сложности. Пусть $a[v_1][v_2]$ — ответ на задачу, если мы закончили в вершине v_1, а последний излом был в вершине v_2. Тогда пересчитывается он просто — перебираем, куда мог пойти путь после вершины v_1. Итого мы получим решение за $O(N^3)$, что вполне нам подходит.

Задача B. Елки

Имя входного файла:	firs.in
Имя выходного файла:	firs.out
Ограничение по времени:	1 секунда
Ограничение по памяти:	256 мегабайт

Граница между царствами Пети и Васи имеет вид отрезка прямой линии длиной $N - 1$ метр. Царь Вася для соблюдения секретности распорядился

вдоль границы высадить N пушистых ёлок (на расстоянии одного метра друг от друга). Он думает, что благодаря этому агенты царя Пети не смогут наблюдать за его страной. Для ухода за ёлками он нанял садовника, который каждое утро проходил вдоль всех ёлок, выбирал наименее пушистую (то есть с наименьшим количеством иголок) и опрыскивал её специальным средством. (Если таких ёлок несколько, то он выбирал первую). От этого средства количество иголок на всех ёлках в радиусе 1 метра (т.е. от одной до трёх ёлок) удваивалось. Однако царь Петя решил противодействовать ему и нанял другого садовника. Он каждый вечер действовал по той же схеме, что и садовник Васи (в целях конспирации, конечно), однако средство у него было другое. От этого средства все ёлки в радиусе 1 метра погибали.

Вас нанял министр финансов царя Пети, чтобы узнать, через сколько дней умрут все ёлки. Напишите программу, которая скажет ему это.

Формат входного файла

В первой строке задано число деревьев N ($2 \le N \le 100$). Во второй строке задано N целых чисел a_i ($1 \le a_i \le 100000$) количество иголок на ёлках (в том порядке, в котором они растут). Все данные приведены по состоянию на первое утро, до прохода Васиного садовника.

Формат выходного файла

Выведите количество дней, в течение которых на границе остаётся хотя бы одна ёлка.

Примеры

firs.in	firs.out
3 3 2 2	1
3 2 2 3	2

Разбор задачи «Ёлки»

Решение задачи состоит в моделировании процесса, описанного в условии. Ясно, что после каждого дня количество ёлок уменьшится хотя бы на единицу; поэтому общее число промоделированных дней не превзойдёт исходного количества ёлок. Осталось только научиться быстро находить ёлку, пушистость которой минимальна, а при равенстве пушистостей — минимальную по номеру. Два возможных решения, позволяющие промоделировать один день за O(log$n$) операций — это использование двоичной кучи и построение дерева отрезков. Остановимся на первом способе как на наиболее естественном.

Наряду с пушистостью каждой ёлки будем хранить минимальную из пушистостей первой и второй ёлок, минимальную из пушистостей третьей и четвёртой и так далее (таких вспомогательных чисел будет n/2). Далее, по этим числам получим минимальную из пушистостей первых четырёх ёлок, минимальную из пушистостей ёлок 5, 6, 7, 8, и так далее (этих чисел будет уже n/4). Продолжив построение таким образом, мы в конце концов дойдём до уровня, на котором хранится одно число — минимальная изо всех пушистостей; при этом всего уровней будет log$n$, а на k-*ом* из построенных уровней каждое число отвечает за отрезок из 2^k ёлок. Теперь, чтобы найти, какая же ёлка имеет эту минимальную пушистость, следует спуститься на предыдущий уровень и посмотреть, какое из двух чисел на нём равно общему минимуму, затем от него ещё на уровень ниже и так далее, пока мы не окажемся на уровне, где каждое число отвечает только за одну ёлку. Когда же мы изменим какую-то из пушистостей, это изменит не более чем log$n$ хранимых чисел, поскольку изменённая ёлка содержится ровно в одном отрезке каждого уровня.

Подобную структуру данных удобно хранить как двоичную кучу. Число, отвечающее за минимум пушистостей всех ёлок, будем хранить в a[1]. Числа предыдущего уровня, отвечающие за левую и правую половины всего множества ёлок, запишем в a[2] и a[3], и так далее. Наконец, числа, соответствующие индивидуальным ёлкам, положим в $a[2^d]$, $a[2^d+1]$, ..., где d, = log$n$,. Условимся обозначать несуществующие ёлки большой константой C. Наконец, дополним наши ёлки

фиктивными (тоже с пушистостью *C)* так, чтобы слева от первой и справа от последней была по крайней мере одна ёлка и чтобы общее число ёлок равнялось 2^d. Теперь пересчитывать минимумы при изменении пушистости можно по простой формуле a [i] = min{a[2 • i], a[2 • i + 1]}, а после нахождения наименее пушистой ёлки *j* изменять пушистость надо у ёлок *j — 1, j* и *j* + 1 независимо от их существования. Процесс прекращается, когда a[≥] *C*.

Следует также отметить, что при удвоении пушистости одной ёлки несколько раз пушистость может оказаться слишком большой, чтобы поместиться в целочисленный тип. Этого можно избежать, используя вещественный тип для хранения пушистостей; особенности хранения вещественных чисел таковы, что при удвоении числа потери точности не происходит, и все пушистости будут храниться и сравниваться без погрешностей.

Задача C. Эвкалипты

Имя входного файла:	eucalypt. in
Имя выходного файла:	eucalypt. out
Ограничение по времени:	1 секунда
Ограничение по памяти:	256 мегабайт

В стране царя Пети возле границы растёт сад из эвкалиптов. Опасаясь того, что Вася захочет этот сад погубить (ведь Вася ещё не забыл про свои любимые ёлочки, и что с ними стало, тоже помнит), Петя решил обнести сад оградой. Естественно, тут же встрял его казначей с фразой, что денег мало (страны-то были бедные...). Царь уже хотел сказать, что надо просто построить забор минимальной длины, однако его военный советник сообщил, что, в целях наилучшей охраняемости, забор должен выглядеть, как многоугольник со сторонами, параллельными осям координат.

Теперь Петино задание выглядит так — постройте забор минимальной длины таким образом, чтобы все деревья были внутри

забора, а забор был бы несамопересекающимся и несамокасающимся многоугольником со сторонами, параллельными осям координат. Выполните это задание. Считайте, что эвкалипт может расти на границе забора.

Формат входного файла

В первой строке содержится единственное целое число — *N* ($2 \le N \le 100000$) — количество деревьев в саду. В следующих JV строках содержится описание деревьев — каждое состоит из двух целых чисел x_i, y_i ($-10^9 \le x_i \le 10^9$, $-10^9 \le y_i \le 10^9$)— координат точки, в которой растёт *г-е* дерево. Все деревья растут в различных точках. Как минимум две ж-координаты и две у-координаты деревьев различны.

Формат выходного файла

Выведите одно вещественное число с точностью не менее шести знаков после десятичной точки — минимальную длину забора.

Пример

eucalypt.in	eucalypt.out
4 10 01 11 00	4.0

Разбор задачи «Эвкалипты»

Пусть у нас есть такой забор. Посчитаем отдельно длину отрезков, идущих влево, вправо, наверх и вниз. Пусть наименьшая координата по оси *OX* у наших деревьев — это *x,* а наибольшая — *X.* Тогда сумма отрезков, идущих влево не меньше, чем *X — x,* так как отрезки остальных типов нас не могут сдвигать влево. Аналогично отрезков всех остальных типов. Тогда получим, что длина периметра не меньше, *2(X — x + У — у).* Такую длину можно получить, просто нарисовав минимальный прямоугольник.

Задача D. Компьютерные игры

Имя входного файла:	games.in
Имя выходного файла:	games.out
Ограничение по времени:	2 секунды
Ограничение по памяти:	256 мегабайт

Вася — начинающий хакер. Недавно он написал программу, которая сохраняет журнал всех нажатий клавиш на компьютере. Мечтая поскорее опробовать ее в деле, он запустил ее перед уходом в школу, чтобы узнать все пароли и секретные данные своего младшего брата Пети.

Придя из школы, Вася обнаружил, что, судя по журналу, Петя действительно использовал компьютер, но вряд ли он вводил какие-либо пароли. Скорее всего, Петя просто запустил какую-то игру, и немного в нее поиграл. Однако, программа Васи не записывает в журнал щелчки мыши, поэтому Вася не может с ходу определить, что это была за игра. Вася решил написать программу, которая бы это выяснила.

Васе известно, какими клавишами ведется управление в каждой из игр. Однако, его задание усложняется тем, что Петя мог иногда промахиваться мимо нужных клавиш управления. Вася решил принять за рабочую гипотезу утверждение, что Петя промахнулся не более чем в 10% нажатий клавиш.

Вашей задачей будет повторить достижение Васи, написав программу, которая определяет, во что мог играть Петя.

Формат входного файла

В первой строке входного файла задано N — количество игр, установленных на компьютере *N* ($1 \le N \le 10000$). Далее следуют JV строк, в которых записаны название каждой из игр и клавиши, используемые в игре для управления. Название — это непустая строка из не более 20 больших и маленьких латинских букв. Описание клавиш — это непустая строка, отделенная от названия одним пробелом, и состоящая не более чем из 26 различных маленьких латинских букв. Названия игр во входном файле различны.

В последней строке входного файла содержится журнал нажатий клавиш — строка, состоящая из L маленьких латинских букв ($1 \le L \le 100\ 000$).

Формат выходного файла

Выведите в выходной файл названия всех игр, в которые мог играть Петя, если известно, что он промахнулся не более чем в 10% нажатий. Названия нужно выводить в том порядке, в котором они даны во входном файле. Если Петя не мог играть ни в какую из игр, выведите N0 SOLUTION.

Примеры

games.in	games.out
4 Doom awsd Warcraft ahpmsbf Civilization rmic Quake awsdzc adwdqdsaawwzdswaawws	Doom Quake
1 Doom awsd programmacrobeginend	NO SOLUTION

Разбор задачи «Компьютерные игры»

Для каждой игры нам надо узнать, мог ли в неё играть Петя. Простейший алгоритм решения этой задачи состоит в следующем. Для каждой нажатой клавиши проверяем, является ли она допустимой для данной игры, и подсчитываем таким образом количество нажатых допустимых клавиш. Такое решение работает за O(N * L) при условии, что мы умеем определять допустимость клавиши за O(1) (это можно сделать, если для каждой игры завести boolean массив от 1 до 26, в котором хранить, допустима ли соответствующая клавиша). Но это решение может быть ускорено. А именно, заметим, что допустимость нажатия одной и той же клавиши, при таком подходе мы проверяем

много раз, а если заранее подсчитать, сколько раз встречалась каждая клавиша (естественно это сделать за один проход строки нажатых клавиш), то получаем решение, работающее за O(N * 26 +L)

Задача E. «Дороги»

Имя входного файла: roads.in
Имя выходного файла: roads.out
Ограничение по времени: 1 секунда
Ограничение по памяти: 256 мегабайт

В стране царя Васи есть N городов. И, как обычно, некоторые из них соединены друг с другом двусторонними дорогами. Всё было замечательно, но в какой-то момент Вася внезапно заподозрил, что из столицы (где Вася и обитает, как и должно царю) не до всех городов можно доехать в карете (карета может ездить только по дорогам). И тогда царь топнул ногой и приказал составить план по строительству новых дорог, чтобы исправить это досадное упущение. Тут подошёл министр финансов и сказал, что денег в казне мало. Вася, не особо долго думая, сказал, что авторы планов, в которых дорог надо будет построить больше, чем минимальное необходимое количество, будут лишены головы. Того же, кто предложит самый красивый план, ждёт награда.

В результате Васе сдали много различных планов. Дураков у него в подданных не было, поэтому все планы были разными и содержали минимальное количество дорог, причём оказалось, что все возможные планы с минимальным количеством дорог были придуманы. Таким образом, у Васи обнаружилось много-много бумажек, на каждой из которых был изображён вариант. Их оказалось слишком много (во всяком случае, так показалось царю) и он решил делать так — брать ровно по K вариантов и откидывать их. Этот процесс продолжался до тех пор, пока вариантов не стало меньше, чем K.

Царю соседнего государства Пете для осуществления дипломатического давления на Васю оказалось очень важно узнать, сколько вариантов осталось у Васи (такая вот сложная штука

политика...) Он попросил Вас сказать ему их количество. Так как количество вариантов оказалось довольно большим, то Вам пришлось написать программу.

Формат входного файла

В первой строке содержится 3 целых числа N, M и K ($1 \le N \le 1000$, $0 \le M \le 100\ 000$, $1 \le K \le 10^9$) — соответственно количество городов в стране, количество дорог и число K из условия. Далее следуют M строк с описаниями дорог. Описание дороги состоит из двух целых чисел a и b ($1 \le a, b \le N$) — номеров городов, соединённых этой дорогой. Никакая дорога не соединяет город сам с собой. Любые два города соединены не более чем одной дорогой.

Формат выходного файла

Выведите одно целое число — количество оставшихся вариантов.

Примеры

roads.in	roads.out
3 1 2 1 2	0
3 0 1000000	3
2 1 2 1 2	1

Разбор задачи «Дороги»

Построим граф G, вершины в котором будут соответствовать городам в стране царя Васи, и две вершины будут соединены ребром тогда и только тогда, когда соответствующие им города соединены дорогой. Тогда условие задачи можно будет пере-формулироватль следующим образом. Найти количество способов дополнить данный граф до связного, добавив при этом наименьшее возможное число рёбер, и вывести это количество по модулю K. Пусть в G имеется $к$ компонент связности, мощности которых равны oi, a_2,..., $a_к$. Тогда чтобы

сделать граф *G* связным, нужно будет добавить не менее *к* — \ ребра. С другой стороны, легко видеть, что *к* — \ ребра достаточно, чтобы сделать граф связным. При этом очевидно, что искомое количество способов зависит только от чисел oi, a_2,..., $a_к$ и не зависит от внутренней структуры компоненты связности. Поэтому искомое количество можем обозначить через *f({ai,* a_2,..., $a_к$*}).* Докажем индукцией по *к,* что

$$f(\{a_1, a_2, ..., a_k\}) = a_1 a_2 ... ak(a_1 + a_2 + ... + ak)^{k-2}$$

База при *к* = 1 очевидна. Предположим, что мы доказали (1) для всех *к* ^ *ко* и докажем её для k =k_0 + 1. Рассмотрим некоторый способ дополнить граф до связного и рассмотрим (k_0 + 1)-ю компоненту связности. Пусть S $\subset$ {1, …, k_0} - в ТОЧНОСТИ непустое множество индексов компонент связности, с которыми новым ребром соединена (k_0+1)-я компонента. Такое соединение могло быть произведено

$$(a_{k0+1})^{|S|} * \prod_{i \in S} a_i$$

способами. Причём компоненты с индексами из *S* уже будут соединены между собой, поэтому получившийся граф можно будет дополнить до связного

$$f(\{ai \mid 1 \le i \le k_0, i \notin S\} \bigcup \{\sum_{i \in S} a_i\})$$

способами. Таким образом получаем для / следующую формулу:

$$f(\{a_1, a_2, ..., ak_0 +_1\}) = \sum_{\substack{S \subset \{1,...,k_0\} \\ S \neq \theta}} (a_{k_0+1})^{|S|} * \prod_{i \notin S} a_i * f(\{a_i \mid 1 \le i \le k_0, i \notin S\} \bigcup \{\sum_{i \notin S} a_i\})$$

Преобразуем её, сгруппировав слагаемые с одинаковой \S\, и воспользовавшись предположением индукции:

$$f(\{a_1,a_2,...,ak_0+_1\}) = \sum_{\substack{S\subset\{1,...,k_0\}\\ S\neq\theta}}((a_{k_0+1})^{|s|} * \prod_{i\notin S} a_i * f(\{a_i \mid 1\le i\le k_0, i\notin S\}\bigcup\{\sum_{i\notin S} a_i\})) =$$

$$= \sum_{l=1}^{k_0}((a_{k_0+1})^l \sum_{\substack{S\subset\{1,...,k_0\}\\ |S|=l}} (\prod_{i\in S} a_i * f(\{a_i \mid 1\le i\le k_0, i\notin S\}\bigcup\{\sum_{i\in S} a_i\}))) =$$

$$= \sum_{l=1}^{k_0}((a_{k_0+1})^l \sum_{\substack{S\subset\{1,...,k_0\}\\ |S|=l}} (\prod_{i=1}^{k_0} a_i * \sum_{i\in S} a_i * (\sum_{i=l}^{k_0} a_i)^{k0-l-1})) =$$

$$= \sum_{l=1}^{k_0}((a_{k_0+1})^l * \prod_{i=1}^{k_0} a_i * (\sum_{i=1}^{k_0} a_i)^{k0-l-1} * \sum_{\substack{S\subset\{1,...,k_0\}\\ |S|=l}} \sum_{i\in S} a_i)$$

Выражение

$$\sum_{\substack{S\subset\{1,...,k_0\}\\ |S|=l}} \sum_{i\in S} a_i$$

симметрично относительно $a_1,a_2,\ldots, a_{k_0}$ и содержит $C_{k_0}^l * l$ слагаемых, поэтому каждое a_i входит в эту сумму $C_{k_0}^l * l / k_0 = C_{k_0-1}^{l-1}$ раз. Поэтому

$$\sum_{\substack{S\subset\{1,...,k_0\}\\ |S|=l}} \sum_{i\in S} a_i = C_{k_0-1}^{l-1} \sum_{i=1}^{k_0} a_i$$

Но тогда

$$f(\{a_1, a_2, ..., a_{k_0+1}\}) = \sum_{l=1}^{k_0} (C_{k_0-1}^{l-1} * (a_{k+1}) * \prod_{i=1}^{k_0} a_i * (\sum_{i=1}^{k_0} a_i)^{k_0-1}) =$$

$$\prod_{i=1}^{k_0+1} a_i * \sum_{l=0}^{k_0-1} (C_{k_0-1}^{l} * (a_{k+1})^l * (\sum_{i=1}^{k_0} a_i)^{k_0-l-1}) = \prod_{i=1}^{k_0+1} a_i * (\sum_{i=1}^{k_0+1} a_i)^{k_0-1}$$

Последний переход - формула бинома Ньютона. Таким образом формула (1) доказана для $k=k_0+1$, а значит, по методу мат. индукции верна для любого натурального k.

Таким образом, благодаря формуле (1), задача свелась к нахождению количества и мощностей компонент связности графа G. Это можно сделать, например, при помощи поиска в глубину.

Задача F. Дороги-2

Имя входного файла:	roads2. in
Имя выходного файла:	roads2. out
Ограничение по времени:	1 секунда
Ограничение по памяти:	256 мегабайт

После того, как Вы написали Пете программу, царь Вася решил предпринять ответные шаги. Вася был очень решительным царем, поэтому уже через два часа начальник разведки докладывал информацию о системе дорог государства Пети. Ему удалось выяснить, что все дороги в этом царстве односторонние. Кроме того, каждый город имеет секретный номер, который ему узнать не удалось. Зато ему удалось узнать, что все города пронумерованы по порядку, начиная от 1. А также, что каждая дорога ведёт из города с меньшим секретным номером в город с большим. В целях оказания Пете дипломатического отпора, Васе нужно узнать секретный номер, который имеет столица государства Пети. Конечно, может оказаться

так, что однозначно его не определить, поэтому он хочет найти все возможные варианты. Личный программист царя Васи не смог справиться с этой задачей, поэтому он решил дать Вам шанс искупить свою вину и помочь ему.

Формат входного файла

Вам будет дана карта дорог, составленная разведчиками. У каждого города будет свой номер, данный разведкой при изучении этого города. В разведданных столица имеет номер 1.

В первой строке даны два целых числа N и M ($1 \le N \le 100\ 000$, $0 \le M \le 100\ 000$). Далее в M строках находятся описания дорог — два целых числа a и b ($1 \le a, b \le N$) — соответственно начало и конец дороги. Между двумя городами проходит не более одной дороги.

Формат выходного файла

В первой строке выведите количество возможных секретных номеров столицы. Во второй через пробел выведите в порядке возрастания сами номера. Гарантируется, что существует хотя бы одна возможная нумерация городов.

Примеры

roads2.in	roads2.out
2 0	2 1 2
2 1 2 1	1 2
3 2 2 1 2 3	2 2 3

Разбор задачи «Дороги-2»

Так как при «секретной» нумерации получается, что дороги идут в направлении возрастания номеров, то «секретная» нумерация — одна из

возможных топологичских сортировок графа дорог. По условию гарантируется существоование хотя бы одной топологической сортировки. Если мы рассмотрим любую вершину, то она обязательна должна идти после всех вершин, из которых она достиживма и перед всеми, которые из неё достижимы. Если же позиций для неё несолько, то мы можем ставить её на любую из них.

Если мы посчитаем два числа B — количество вершин перед ней и A — после неё, то всего позиций для неё будет $N — B — A$: $B + 1$, $B + 2$,..., $N — A$.

Решение будет следующим:

1. Создаём представление прямого и обратного графа списком рёбер из каждой вершины. Здесь head[num] - указатель на впервое ребро из данной вершины, a next [num] - на следущее за ним. inl [num] - количество уже назначенных рёбер.

```
void insert(int num, int f, int s) {
        next[num][inl[num]] = head[num][f];
        head[num][f] = inl[num];
        end[num][inl[num]] = s;
        inl[num]++;
}

...
for(i=0; i < m; i++) {
        scanf("%d%d", &a, &b);
        a--, b--;
        insert(0, a, b);
        insert(1, b, a);
}
```

2. Считаем количества вершин достижимых из нашей вершины и тех, из которых она достижима. Последние вершины - это те, которые достижимы из нашей в обратном графе. Делаем это поиском в глубину, возвращающим число вершин, в которых он побывал.

```
int dfs(int v, int num)  { int r = 1, e; mark[num][v] = 1;



for( e = head[num][v];  e  != -1; e = next[num][e]   )
if( ! mark[num] [end[num][e]] )
r += dfs(end[num][e], num);
return r;
}
. . .
a = dfs(0, 0)-l;
b = dfs(0, 0-1);
```

3. Выодим ответ:

```
printf("%d\n",  n-a-b);
for(i = 0; i < n-a-b; i ++)
printf("%,d",  b+i+1);
```

Суммарно решение будет выглядеть так:

```
#include <stdio.h>
#include <memory.h>

#define maxn 100000

int n,  m;
int head[2][maxn],  next[2][maxn], end[2][maxn],inl[2];
int mark[2][maxn];

void insert(int num, int f, int s){
        next[num][inl[num]] = head[num][f];
        head[num][f] = inl[num];
        end[num][inl[num]] = s;
        inl[num]++;
        }

int dfs(int v, int num)
        int r = 1, e;
```

```
        mark[num][v] = 1;

        for( e = head[num][v]; e != -1; e = next[num][e])
         if(!mark[num][end[num][e]])
                r += dfs(end[num][e],num);
                return r;
}

int main(void) {
        int i, a, b;

        freopen("roads2.in", "r", stdin);
        freopen("roads2.out", "w", stdout);

        memset(head, -1, sizeof(head));
        scanf("%d%d", &n, &m);

        for( i = 0; i < m; i++ ) {
        scanf ("%d%d", &a, &b);
        a--, b--;
        insert(0, a, b);
        insert(1, b, a);
}
a = dfs(O, 0)-l;
b = dfs(O, 1)-1;

printf("%d\n", n-a-b);
for( i = 0; i < n-a-b; i++ )
printf("%d ", b+i+1);
return 0;
}
```

Задача G. Музыкальная дорожка

Имя входного файла: mustrack.in
Имя выходного файла: mustrack.out
Ограничение по времени: 1 секунда
Ограничение по памяти: 256 мегабайт

В эвкалиптовом лесу царя Пети, в развлекательном парке, установили новый аттракцион «Музыкальная дорожка». Он представляет собой зацикленную дорожку из *n* ступенек, каждая из

которых издаёт звук определённой частоты и силы. Ступеньки занумерованы числами от 1 до *n*. Каждый желающий покупает билет, после чего делает *n* прыжков с одним и тем же интервалом, начиная с первой ступеньки и продолжая в направлении возрастания нумерации.

И, как и на всех других аттракционах, на нём взяли за привычку кататься по разу в день все звери леса. Всего в лесу *к* зверей. Интересными оказались предпочтения зверей: каждый день они случайным образом перераспределяют между собой первые *к* простых чисел и используют эти числа как интервалы для «Музыкальной дорожки».

Инспекция по слежению за загрязнением окружающей среды очень обеспокоена уровнем звукового загрязнения местности. Поэтому сотрудники инспекции попросили Вас написать программу, подсчитывающую суммарное число издаваемых за день децибел. Так как ответ может быть очень большим, то все вычисления необходимо провести по модулю *m*.

Формат входного файла

Первая строка входного файла содержит три целых числа *n, к* и *m* — количество ступенек

($1 \le n \le 200000$), количество зверей ($1 \le к \le 10^6$) и модуль ($2 \le m \le 10^9$).

Вторая строка содержит *n* чисел a_i ($0 \le a_i \le 10^9$), где a_i – кол-во децибел, издаваемых ступенькой номер *i*.

Формат выходного файла

В выходной файл выведите единственное число — количество децибел, издаваемых за день, взятое по модулю *m*.

Примеры

mustrack.in	mustrack.out

3 1 1000 1 2 3	6
5 2 29 1 2 3 4 5	1

Разбор задачи «Музыкальная дорожка»

Для дальнейших нужд будем нумеровать ступеньки с 0. Если участник использует число *p,* то он будет прыгать по ступенькам со следущими номерами:

$$\{0, p \bmod n, 2p \bmod n, 3p \bmod n, \ldots, (n-1)p \bmod n\}$$

Если *p* не делит *n,* то среди этих остатков по модулю *n* встречаются всевозможные, поэтому обладатель интервала *p* прыгнет по разу на каждую ступеньку и произведёт звук, равный суммарному числу звуков, издаваемых всеми ступеньками.

Для делителей же ситуация будет несколько иная: если $n = pk$, тогда в ряду $\{0,p, 2p, 3p,\ldots, (n-1)p\}$, будут по *p* раз встречаться числа $\{0,p, 2p, 3p,\ldots, (k-1)p\}$

Понятно, что делители *n* не больше миллиона. Так как простых чисел до 200,000 не больше 100,000, то нам достаточно найти только их, чтобы учесть все возможные делители.

В свете этого решение будет следующим:

1. С помощью метода решета Эратосфена находим простые до *n*.

```
sq = ((int) sqrt(n*1.0) ) + 1;

for( i = 2; i <= sq; i++ ) {
if( not_prime[i] ) continue;
for( j = i*i; j <= n; j += i ) not_prime[j] = 1;
  }
```

```
}
```

2. Собираем их вместе.

```
for( number_of_prime = 0 ,   i = 2 ;   i < = n ;   i ++)

if( !not_prime[i] )
prime[number_of_prime++]   = i;
```

3. Вычисляем сумму всех по модулю *m*.

```
for( i = 0; i < n; i++)
sum += a[i];

sum %= m;
```

4. Далее вычисляем ответ для всех простых не больших *n*.

Задача H. Самолёт

Имя входного файла: plane.in
Имя выходного файла: plane.out
Ограничение по времени: 1 секунда
Ограничение по памяти: 256 мегабайт

У царя Пети есть личный самолёт с личным пилотом, и царь частенько летает на нём из столицы в свою любимую резиденцию. Чтобы сократить время полёта, пилот выбирает кратчайший маршрут между столицей и резиденцией.

Однако, летая в резиденцию и обратно, Петя каждый раз испытывает некоторое беспокойство. Дело в том, что разведка сообщила, что на северном полюсе расположена ракетная база царя Васи, и пока неизвестно, какова дальность пуска ракет класса «земля-воздух», находящихся на этой базе.

Будем считать, что планета, на которой живёт Петя, имеет форму шара, её поверхность — сфера, столица и резиденция — две точки на этой сфере, а маршрут полёта — это кратчайший путь по сфере между этими двумя точками. Петя хочет выяснить, в каком месте он

приближается к северному полюсу на минимальное расстояние при полёте.

Поскольку царь не знает, чем руководствуется пилот при выборе маршрута, кроме его длины, он решил, что если оптимальных маршрутов несколько, то надо найти ближайшую к северному полюсу точку из всех точек всех возможных кратчайших маршрутов. А поскольку Петя сам не очень-то понимает, как это сделать, он поручил эту задачу Вам.

Формат входного файла

В первой строке входного файла записаны через пробел широта и долгота столицы, а во второй — широта и долгота любимой резиденции Пети.

Широта задаётся в формате *d m s c* — градусы, минуты, секунды и полушарие. $0 \le d \le 90$, $0 \le m \le 59$, $0 \le s \le 59$, а **c** — это символ N для северного полушария и S для южного; широта не может быть больше 90 градусов.

Долгота задаётся в формате *d m s c* — градусы, минуты, секунды и полушарие. $0 \le d \le 180$, $0 \le m \le 59$, $0 \le s \le 59$, а **c** — это символ W для западного полушария и E для восточного; долгота не может быть больше 180 градусов.

Гарантируется, что столица и резиденция находятся в различных точках сферы.

Формат выходного файла

Выведите в первой строке широту и долготу искомой точки через пробел с точностью до секунды. Формат совпадает с форматом входных данных. Если оптимальных ответов несколько, разрешается вывести любой.

Примеры

plane.in	plane.out
2000S1000E 2000S11000W	2000S1000E
123456N142445W 234553N152134W	234553N152134W
5000N9000E 5000N3000W	671422N3000E

Разбор задачи «Самолёт»

Будем считать поверхность планеты сферой радиуса 1 с центром в начале координат. Пусть $A(x_i,y_i, z_i)$ — столица, $B(x_2, y_2, z_2)$ ~ резиденция. Тогда кратчайший путь между столицей и резиденцией лежит в плоскости, проходящей через *A, B и* начало координат. Если точки *A и B* симметричны относительно начала координат, то таких плоскостей бесконечно много. В частности, найдётся плоскость, проходящая через северный полюс. Поэтому в этом случае ответом является сам северный полюс.

Пусть теперь *A* и *B* не симметричны относительно начала координат. Тогда такая плоскость *P* единственна, и уравнение этой плоскости имеет вид $ax + by + cz = 0$, где $a = y_1z_2 — z_1y_2$, $b = x_2z_1 — x_1z_2$, $c = x_1y_2 — x_2y_1$. Плоскость *P* пересекает сферу по окружности *S*. Найдём точку *C* с максимальной координатой по оси *Oz*, что равносильно тому, что она ближайшая к северному полюсу, на окружности *S*.

Если точки *A и B* лежат на экваторе, то ответом является любая точка кратчайшего пути. В частности, подходят точки *A* и *B*. Если же хотя бы одна из точек *A* или *B* не лежит на экваторе, то такая точка *C* определена однознозначно, и её можно искать, например, методом множителей Лагранжа.

Для этого рассмотрим функцию

$$L(x,y,z) = z-\mu/2(x^2 + y^2 + z^2-l) + \lambda(ax + by + cz).$$

Чтобы точка *C(x, y, z)* была точкой экстремума функции *L* необходимо, чтобы она удовлетворяла следующей системе:

$$\begin{cases} -\mu x + \lambda a = 0 \\ -\mu y + \lambda b = 0 \\ 1 - \mu z + \lambda c = 0 \\ x^2 + y^2 + z^2 = 1 \\ ax + by + cz = 0 \end{cases}$$

Выражая *x, y и z из* первых трёх уравнений и подставляя в пятое находим, что

$$\lambda = -\frac{c}{a^2 + b^2 + c^2}$$

(Следует заметить, что так как *A* или *B* не лежит на экваторе, то $a^2 + b^2 + c^2 > 0$.) Затем подставляя их в четвёртое уравнение системы находим, что

$$\mu^2 = -\frac{a^2 + b^2}{(a^2 + b^2 + c^2)^2}$$

Теперь из первых трёх уравнений системы легко найти *x, y и z*. Причём, так как нас интересует точка с наибольшей $z = (\lambda c + 1)/\mu$ то μ следует взять положительным. Теперь если точка *C* лежит на кратчайшем пути, то она и является ответом. А если не лежит, то ответом является точка *A* или *B,* в зависимости от того, какая из ближе к северному полюсу.

Задача I. Бильярд

Имя входного файла: pool.in
Имя выходного файла: pool.out
Ограничение по времени: 1 секунды
Ограничение по памяти: 256 мегабайт

Одна крупная компания разрабатывает робота для игры в бильярд. Вашей группе поручено важное задание в рамках этого проекта — написать программу, которая будет определять результат пущенного роботом шара.

Бильярд представляет собой прямоугольный стол m x n сантиметров, на котором расположены K шаров радиуса R сантиметров каждый. Вашей программе будет передано расположение этих шаров на столе, номер катящегося шара A и его направление движения (dx, dy), а также число L, обозначающее максимальное расстояние в сантиметрах, которое может прокатиться шар.

Вашей программе требуется определить, какое из трех событий произойдет раньше: шар остановится, шар коснется другого шара или шар коснется бортика.

Формат входного файла

В первой строке входного файла заданы шесть целых чисел m, n, K, R, L, A ($5 \le m \le 1\,000$, $5 \le n \le 1000$, $1 \le K \le 100$, $1 \le R \le 20$, $1 \le L \le 1000$, $1 \le A \le K$). Во второй строке задаются два целых числа dx и dy, по модулю не превосходящие 100, обозначающие направление, в котором движется шар. Эти два числа обозначают, на сколько бы сдвинулся шар через некоторое время, если бы на плоскости не было никаких препятствий. Хотя бы одно из этих чисел отлично от нуля. Последние K строк файла содержат целые координаты шаров x_i и y_i ($0 \le x_i \le m$, $0 \le y_i \le n$).

Гарантируется, что шары в начальном положении не касаются друг друга и не касаются бортика.

Формат выходного файла

В выходной файл выведите одно из следующих трех слов, соответствующих событию, которое произошло раньше:

- BOARD, если шар врезался в бортик.
- TOUCH, если шар коснулся другого шара.
- STOP, если шар остановился (прокатился *L* сантиметров без препятствий).

Гарантируется, что одновременно с первым событием не произойдет никакого другого.

Примеры

pool.in	pool.out
15 15 2 1 10 1 1 1 2 2 11 11	STOP
15 15 2 1 11 1 1 1 2 2 11 11	TOUCH
15 15 2 1 10 1 1 -1 2 2 11 11	BOARD
15 15 2 1 10 1 1 0 2 2 6 4	TOUCH

Разбор задачи «Бильярд»

Общий ход решения задачи таков. Пусть для определённости за одну секунду шар смещается на вектор *(dx, dy).* Будем считать, что при столкновении с другим объектом шар продолжает катиться с

неизменной скоростью. Рассмотрим луч, по которому движется шар, и отметим на нём момент времени, в который шар коснётся края доски, а также все моменты столкновения с другими шарами. Выберем минимальный из этих моментов. Если он больше, чем время, требуемое для того, чтобы шар прокатился *L* сантиметров, выведем «STOP». В противном лее случае выведем «BOARD» или «TOUCH» в зависимости от того, какое событие случилось раньше.

Выяснить, когда шар достигнет границы, просто. Для этого, например, молено пересечь луч с прямыми $x = r$, $x = m - r$, $y = r$ и $y = n$ — r, а затем выбрать наиболее ранний из моментов пересечения.

Чуть сложнее узнать, когда шар впервые соприкоснётся с другим шаром. Найдём сначала расстояние от центра неподвижного шара до прямой, по которой движется наш шар. Эта прямая имеет уравнение $a*x+b*y = c$, где $a = dy$, $b = -dx$, $c = a*x_o+b*y_o$, а $A = (x_Q, y_o)$ — начальные координаты движущегося шара. Расстояние d от этой прямой до точки $B = (x_1, y_1)$ будет равно $\frac{|(a*x^2+b*y^2)-c|}{v}$, где $v = \sqrt{dx^2 + dy^2}$ — скорость движения шара. Если это расстояние больше, чем 2 • г, то шары в процессе движения не соприкоснутся. В противном случае выясним, в какой момент времени они впервые соприкоснулись. Момент времени t, в который центры шаров ближе всего друг к другу, равен скалярному произведению вектора (dx, dy) на вектор $(x_1 — x_0, y_1 - y_0)$, делённому на квадрат скорости v. Рассмотрим точку C, в которой шар оказался в этот момент времени, и точку D, в которой произошло касание. Заметим, что треугольник $ABCD$ прямоугольный, и $|CD| = \sqrt{|BD|^2 — |BC|^2} = \sqrt{(2 • r)^2 — (d • v)^2}$. Таким образом, момент касания получается из t вычитанием величины $|CD|/v$. Теперь, если полученный момент времени лежит между нулём и –,

L/v шары успеют коснуться прежде, чем движущийся шар пройдёт расстояние L.

Задача J. Баянический квадрат

Имя входного файла: square.in
Имя выходного файла: square.out
Ограничение по времени: 2 секунды
Ограничение по памяти: 256 мегабайт

Будем рассматривать квадраты размера *(n* + 1) х *(n* + 1), клетки которых нумеруются от (0, 0) до (п, *n)* (на рисунке изображен такой квадрат при *n* = 2):

	0	1	2
0	(0,0)	(1,0)	(2,0)
1	(0,1)	(1,1)	(2,1)
2	(0,2)	(1,2)	(2,2)

В клетках этого квадрата расставляются вещественные числа от —10 до 10. Квадрат с расставле-ными в нем числами назовем *баяническим,* если для каждой его внутренней (то есть имеющей ровно четырех соседей) клетки верно, что значение в ней равно среднему арифметическому значений этих соседей, т.е. справедлива формула $C_{xy} = (C_{x-1,y} + C_{x}+_{1,y} + C_{x,y+1} + C_{x,y-1})/4$.

К примеру, квадрат

1	2	5	6
2	3	4	5
5	4	3	2
6	5	2	1

является баяническим. Напишите программу, которая бы по известным числам в граничных клетках баянического квадрата строила бы сам квадрат (гарантируется, что решение существует и единственно) .

Формат входного файла

В первой строке входного файла задано число *n* ($2 \le n \le 50$). В последующих строках идут значения чисел на границе квадрата в порядке обхода по часовой стрелке, начиная с клетки (0, 0).

Например, для n = 2 числа будут идти в таком порядке: (0,0) (1,0) (2,0) (2,1) (2,2) (1,2) (0,2) (0,1). Все числа разделены произвольным количеством пробелов и/или переводов строки.

Формат выходного файла

В выходном файле должен содержаться искомый баянический квадрат, по n +1 числу на строку. Каждое число необходимо выводить с пятью десятичными знаками после запятой. Числа в выходном файле расположены в том же порядке, что и в квадратах на рисунке. Строки расположены в порядке возрастания значения y. Соседние числа в строке отделяются друг от друга одним пробелом.

Пример

square.in	square.out
3 1 2 5 6 5 2 1 2 5 6 5 2	1.00000 2.00000 5.00000 6.00000 2.00000 3.00000 4.00000 5.00000 5.00000 4.00000 3.00000 2.00000 6.00000 5.00000 2.00000 1.00000

Разбор задачи «Баянический квадрат»

Рассмотрим преобразование нашего квадрата, при котором значения во всех внутренних клетках одновременно заменяются на среднее арифметическое соседей. Ясно, что если это преобразование применить к баяническому квадрату, то значения в его клетках не изменятся. Если же квадрат не являлся баяническим, и при этом известно, что баянический квадрат с такими числами на границе существует и единственен, то после этого преобразования квадрат стал ближе к баяническому в следующем смысле. Рассмотрим клетки, в которых разность между значением в клетке нашего квадрата и значением в соответствующей клетке баянического квадрата максимальна. Тогда после нашего преобразования количество клеток с такой разностью уменьшилось; в частности, если клетка с максимальной разностью была одна, то и сама максимальная разность уменьшилась. Действительно, если пусть в некоторой клетке разность максимальна, а

в какой-то из соседних — не максимальна; очевидно, что если квадрат не баянический, то максимальная разность больше нуля, и такая клетка существует. Легко показать, что при замене значения в этой клетке на среднее арифметическое значений в её соседях эта разность уменьшится, что и требовалось доказать.

Таким образом, решение задачи состоит в том, чтобы сначала заполнить внутренние клетки квадрата произвольными числами, а затем применить наше преобразование достаточно много раз. Завершать работу алгоритма можно либо после некоторого фиксированного количества итераций, либо когда значения в клетках следующего полученного квадрата будут отличаться от значений в клетках предыдущего не более, чем на некоторую очень малую величину. Сократить количество итераций можно, выбрав подходящие начальные значения для внутренних клеток — например, нули или среднее значение чисел на границе. При заданных ограничениях на размер квадрата и требуемой точности достаточно было сделать несколько тысяч итераций, чтобы получить правильный ответ.

Можно также доказать, что для любых значений на границе баянический квадрат существует и единственен; однако, поскольку это гарантировалось в условии задачи, на соревновании делать этого не требовалось.

Задача K. Танк

Имя входного файла:	tank. in
Имя выходного файла:	tank.out
Ограничение по времени:	1 секунда
Ограничение по памяти:	256 мегабайт

После очередного инцидента цари Вася и Петя решили перейти от дипломатических игр и диверсионных операций к военным действиям. Жалея свои владения, войну они развернули на территории соседней нейтральной страны. Военная карта этой страны имеет вид

прямоугольника *M* x *N*, разбитого на квадраты 1 x 1, и сухопутные войска могут перемещаться только между квадратами, соседними по стороне. Эта страна граничит со страной Васи с севера и со страной Пети с юга. Каждый квадрат либо содержит важный населённый пункт или объект народного хозяйства, либо не имеет стратегического значения; каждый значимый квадрат либо контролируется Петей или Васей, либо сохраняет нейтралитет.

Со стороны владений Васи в страну вторгается универсальный танк-вездеход. Его задача — захватить как можно больше нейтральных значимых квадратов, после чего прибыть в один из уже захваченных до него квадратов для получения дальнейших инструкций. Танк-вездеход — штука мощная, и даже просто проехав по территории нейтрального квадрата, он тут же захватывает его. С другой стороны, одному танку-вездеходу не по силам не то что захватить квадрат, контролируемый войсками царя Пети, а даже просто по нему проехать. Танк не может покидать пределов карты.

Положение осложняется тем, что в армии царя Васи бытуют своеобразные понятия о воинской доблести и чести. Во-первых, ни один солдат, а уж тем более командир танка, не позволит себе отступать. Это значит, что, побывав в некотором квадрате карты, танк-вездеход не может ни вернуться в него позже, ни перейти в квадрат, находящийся ближе к родному северу, чем этот. Во-вторых, командир танка считает позорным заходить на уже захваченные территории, пока не решит, что его задача выполнена.

Помогите командиру танка составить маршрут, который позволит ему захватить как можно больше нейтральных значимых квадратов, не вступая при этом на захваченные территории, и после этого прибыть в квадрат, уже захваченный до него войсками царя Васи.

Формат входного файла

В первой строке входного файла заданы два числа M и N ($1 \le M \le 100$, $1 \le N \le 100$) — размеры зоны боевых действий. Далее следуют N строк по *M* символов в каждой, задающие карту; i-ый символ *j-ой.* из этих строк говорит о том, что находится в квадрате *(i, j).* Значения символов таковы:

- P — Квадрат значимый и контролируется Петей.
- V — Квадрат значимый и контролируется Васей.
- N — Квадрат значимый и сохраняет нейтралитет.
- T — В этом квадрате начинает своё вторжение танк-вездеход.

Квадрат не является значимым.

- . (точка) — В квадрате нет важных объектов или населённых

пунктов.

Гарантируется, что буква T присутствует во входном файле ровно один раз и находится в первой (самой северной) строке карты.

Формат выходного файла

В первой строке выходного файла выведите *L* — длину маршрута и *S* — количество захваченных танком-вездеходом квадратов через пробел. В следующей строке выведите *L* символов — маршрут перемещения танка. Каждый символ обозначает сторону света (E — восток, S — юг, W — запад) и указывает, что нужно переместиться на один квадрат в эту сторону. Если маршрутов с оптимальным *S* несколько, можно вывести любой из них. Если же выполнить задание невозможно, выведите в выходной файл единственное число - 1.

Разбор задачи «Танк»

Решать задачу будем методом динамического программирования. Пусть a[клетка, направление] — максимальный ответ, если мы из клетки «клетка» пошли в направлении «направление». Тогда пересчитать такую функцию ничего не стоит — надо перебрать все направления, в которых можно пойти. При этом надо следить, чтобы не пойти в обратную сторону (это запрещено по условию). Удобнее всего в данном случае эту динамику реализовать рекурсией с запоминанием, в силу того, что иначе клетки надо перебирать в странном порядке. Однако и такой вариант тоже возможен.

Примеры

tank.in	tank.out
43 .T.N NVNP .NVN	52 WSSEE
61 T.P..V	-1
43 T. NNNN .V.	94 EESWWWSEE
34 T.V .VN PPP	20 SE

День шестой (19.02.08 г.).

Контест Андрея Станкевича.

Об авторе…

Станкевич Андрей Сергеевич

Старший преподаватель СПбГУ ИТМО. Тренер команд СПбГУ ИТМО на чемпионате мира по программированию. Председатель жюри Всероссийской командной олимпиады по программированию, Санкт-Петербургской городской олимпиады по программированию. Член жюри NEERC, Всероссийской олимпиады по программированию, учебно-тренировочных сборов школьников России к международной олимпиаде, Петрозаводских зимних и летних учебно-тренировочных сборов.

Старший преподаватель параллели А Летней компьютерной школы (www.lksh.ru)

Преподаватель курса алгоритмов в Академии современного программирования (www.amse.ru). Автор множества задач NEERC, РОИ, других олимпиад, автор 30 контестов для Петрозаводских сборов.

Оосновные достижения на олимпиадах как участника:

- 2 место на NEERC в 2000;
- 4 место (серебряная медаль) на финале 2000;
- 3 место (золотая медаль) на финале 2001;
- 6 место на TopCoder Open 2006;

- 4 место на TopCoder Collegiate Challenge 2006;
- 3 место на Google Code Jam 2006;

Основные достижения команд:

- чемпионы NEERC в 2001, 2003, 2004, 2007 годах;
- 2 место на NEERC в 2005, 2006;
- чемпионы мира 2004 года;
- золотая медаль на финале чемпионата мира в 2003, 2005, 2007 годах.

Задачи и разборы

Задача A. Brackets

Имя входного файла:	brackets.in
Имя выходного файла:	brackets.out
Ограничение по времени:	2 seconds
Ограничение по памяти:	256 megabytes

Regular brackets sequence is the sequence of *2n* characters each of which is either " (" (opening bracket), or ")" (closing bracket), such that each prefix contains no more closing brackets than opening ones, and the whole sequence contains *n* opening and *n* closing brackets. For example, "", "((()))", "()()()", "(0)0" are the regular brackets sequences, but "())(()" is not, because its prefix "())" contains more closing than opening brackets, neither is "((", because it contains more opening brackets than closing.

For each opening bracket you can find the *corresponding* closing bracket — the one following it, such that there is a regular brackets sequence between them.

The generalization of the regular brackets sequence if the regular brackets sequence with κ types of brackets. Let each bracket have its *type* — an integer number from 1 to *k*. The sequence of opening and closing brackets with types is regular if it is a regular brackets sequence and the corresponding closing bracket for each opening bracket has the same type as the opening bracket itself. For example, "$(_1(_2)_2)_1(_1)_1$" is the regular brackets sequence with two types of brackets, but "$(_1(_2)_1)_2(_1)_1$" is not.

If you introduce some order on *2k* typed brackets, you can consider *lexicographical* order on all regular brackets sequences of length *2n*. You are given *n*, *k*, the order on typed brackets and the regular brackets sequence. Find the next regular brackets sequence in the lexicographical order.

Формат входного файла

The first line of the input file contains t — the number of tests ($1 \le t \le 10000$). The description of t test cases follows.

The first line of each description contains two integer numbers: n and κ ($1 \le n \le 100\,000$, $1 \le \kappa \le 10\,000$). The second line contains $2k$ integer numbers — the permutation of the set $\{-k,-(k-1),..., -2, -1,1,2,..., k\}$. It lists the brackets from the least to the greatest in the given order. Positive number i means the opening bracket of the i-th type, negative number $-i$ means the closing bracket of the i-th type. The third line contains $2n$ integer numbers from $-\kappa$ to κ (except 0) and describes the regular brackets sequence.

The sum of n and the sum of κ in all test cases don't exceed 100 000.

Формат выходного файла

For each test case output $2n$ integer numbers — the next regular brackets sequence after the one from the input file in lexicographical order. If the sequence in the input file is the last one, output the first one.

Пример

brackets.in	brackets.out
2 3 2 1 -1 2 -2 1 2 -2 -1 1 -1 3 2 1 -1 2 -2 1 2 -2 -1 2 -2	1 2 -2 -1 2 -2 1 2 -2 2 -2 -1

Разбор

Решение этой задачи стандартно для задач такого класса. Просматривая последовательность с конца, найдем самый последний элемент, который можно увеличить, и увеличим его минимальным

возможным образом. После этого дополним последовательность минимальным возможным «хвостом».

Рассмотрим теперь технические особенности для этой задачи.

Закрывающую скобку можно заменить только на открывающую. При этом эта открывающая скобка должна быть больше заданной закрывающей. Заранее подсчитав для каждой закрывающей скобки следующую после нее в лексикографическом порядке открывающую, мы можем проверить, можно ли увеличить заданную скобку. Дополнительно отметим, что превратить закрывающую скобку в

открывающую можно только если после нее ранее была хотя бы одна открывающая (иначе не хватит длины последовательности закрыть все скобки).

Открывающую скобку можно заменить либо на большую открывающую, либо на закрывающую, которая соответствует лежащей на вершине стека при просмотре последовательности с начала до позиции, на которой стоит рассматриваемая скобка (эти значения следует предподсчитать). На большую открывающую всегда можно заменить, а на закрывающую можно заменить только в случае, когда она больше в лексикографическом порядке заменяемой скобки. Наконец, если возможны оба варианта, следует выбрать меньший.

Теперь рассмотрим, как получить минимальных «хвост». Этот же метод надо применить и для получения минимальной в лексикографическом порядке скобочной последовательности, если исходная последовательность была последней. Каждый раз, ставя очередную скобку, мы имеем два варианта: либо поставить минимальную в лексикографическом порядке открывающую скобку, либо поставить закрывающую, соответствующую лежащей на вершине стека открывающей. Из этих двух вариантов следует выбрать меньший, за исключением случаев, когда один из них невозможен (открывающую нельзя поставить, если остатка длины последовательности не хватит, чтобы ее завершить, а закрывающую — если стек пуст).

Задача B. Car Wash

Имя входного файла:	**carwash. in**
Имя выходного файла:	**carwash. out**
Ограничение по времени:	2 seconds
Ограничение по памяти:	256 megabytes

Ben is the owner of a car wash. Ben offers car washing and dry-cleaning of the car compartment. His wash is located in the capital of England, and he often serves clients from government. Recently Ben has received an order of washing and dry-cleaning of n cars. And this order must be executed as fast as possible!

After preliminary investigation, Ben found out that the i-th car can be washed in a_i, minutes and cleaned in b_i, minutes. Washing and cleaning are performed at different buildings, so they cannot be performed simultaneously for the same car. The order of the two operations for a car is irrelevant, neither is the order of processing the

cars. Ben's best workers Tom and Jerry were called to perform the order. Tom will wash the cars, and Jerry will dry-clean their compartments. Each worker can work with one car at a moment, and due to quality requirements, it is not allowed to switch from one car to another until the one is ready (this condition is independent for each worker, different workers can process cars in any order).

Help Ben to find out how the cars should be processed by the workers, so that all cars were finished as soon as possible.

Формат входного файла

The first line of the input file contains n ($1 < n < 10\,000$). The following n lines contain two integer numbers each: a_i, and b_i ($1 \le a_i, b_i \le 10^5$).

Формат выходного файла

The first line of the output file must contain t — the number of minutes after the process has started

when all cars will be washed and dry-cleaned. You must minimize this number.

The following n lines must contain two integer numbers each — the number of minutes after the start of the process when the washing and the dry-cleaning of the corresponding car must start, respectively.

If there are several optimal solutions, output any one.

Пример

carwash.in	carwash.out
6 10 6 7 9 3 8 1 2 12 7 6 6	39 11 26 4 17 0 7 3 15 27 0 21 32

Разбор

Ясно, что минимальное время обработки не меньше, чем каждая из величин $\sum_{i=1}^{n} a_i$, $\sum_{i=1}^{n} b_i$ и $\max_{i=1}^{n}(a_i + b_i)$. Покажем, что можно обработать все машины за время, равное максимуму из этих величин.

Если максимум достигается на $a_i + b_i$ для некоторого i, то это значит, что $a_i \geq \sum_{j \neq i} b_j$ и значит все машины можно почистить за время, пока идет мойка i-й машины, и, аналогично, $b_i \geq \sum_{j \neq i} a_j$ и значит все машины можно помыть, пока идет чистка i-й машины. Следовательно в этом случае оптимального результата действительно можно достичь.

Пусть теперь максимум достигается на $\sum_{i=1}^{n} a_i$ либо $\sum_{i=1}^{n} b_i$ (то есть очень «длинной» машины нет). Пусть, не ограничивая общности, максимум достигается на $\sum_{i=1}^{n} a_i$.

Решим сначала задачу для случая, когда есть всего три машины. Пусть мойка машин идет в следующем порядке: первая, вторая, третья, а чистка — вторая, третья, первая. При этом мойка идет без перерывов, а чистка второй и третьей — с начала подряд, а первая так, чтобы чистка завершилась в тот момент, когда закончится мойка третьей. Конфликт может возникнуть только из-за второй или третьей машины, с первой проблем быть не может (иначе $a_1 + b_1$ было бы больше суммы всех a_i). Проблемы могут возникнуть, если либо $b_2 > a_1$, либо $b_2 + b_3 > a_1 + a_2$. Если одно из этих двух условий выполняется, выполним «циклический сдвиг» номеров машин, сделав вторую машину первой, третью — второй, а первую — третьей. Тогда чтобы при новой нумерации машин не получилось их обработать указанным образом, должно выполняться (в исходной нумерации) либо $b_3 > a_2$, либо $b_3 + b_1 > a_2 + a_3$. Аналогично, сделав еще один сдвиг, получим, что если и так нельзя обработать машины, то либо $b_1 > a_3$, либо $b_1 + b_2 > a_3 + a_1$. Но одновременно хотя бы по одному неравенству из каждой пары выполняться не может (это противоречит тому, что сумма a_i больше суммы b_i). Значит есть хотя бы один способ корректно упорядочить машины.

Осталось решить задачу для случая более чем трех машин. Пусть есть хотя бы четыре машины. Тогда можно свести задачу к меньшей следующим образом: выберем две машины с минимальными суммами $a_i + b_i$ и $a_j + b_j$, соответственно, и объединим их в одну с временем мойки $a_i + a_j$ и $b_i + b_j$ Заметим, что если машин хотя бы четыре, то

после такого объединения максимум из $\sum_{i=1}^{n} a_i$ $\sum_{i=1}^{n} b_i$, и $\max_{i=1}^{n}(a_i + b_i)$ по прежнему достигается на $\sum_{i=1}^{n} a_i$. Значит, решив задачу для получившегося меньшего набора, получим решение исходной задачи.

Задача C. Painting Cottages

Имя входного файла: cottages.in
Имя выходного файла: cottages.out
Ограничение по времени: 2 seconds
Ограничение по памяти: 256 megabytes

The new cottage settlement is organized near the capital of Flatland. The construction company that is building the settlement has decided to paint some cottages pink and others — light blue. However, they cannot decide which cottages must be painted which color.

The director of the company claims that the painting is *nice* if there is at least one pink cottage, at least one light blue cottage, and it is possible to draw a straight line in such a way that pink cottages are at one side of the line, and light blue cottages are at the other side of the line (and no cottage is on the line itself). The main architect objects that there are several possible nice paintings.

Help them to find out how many different nice paintings are there.

Формат входного файла

The first line of the input file contains n — the number of the cottages ($1 \le n \le 300$). The following n lines contain the coordinates of the cottages — each line contains two integer numbers x_i and y_i ($-10^4 \le x_i, y_i \le 10^4$).

Формат выходного файла

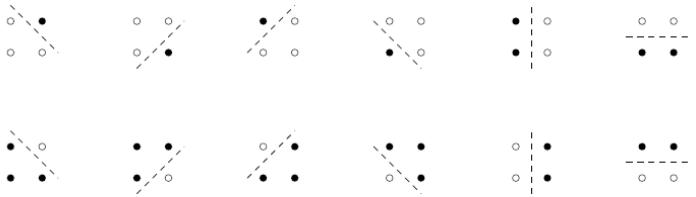
Output one integer number — the number of different nice paintings of the cottages.

Пример

cottages.in	cottages.out

4 0 0 1 0 1 1 0 1	12

The possible nice paintings are shown on the following picture.



Разбор

Увеличим все точки, превратив их в круги малого радиуса. Радиус выберем настолько маленьким, чтобы никакие три круга не имели общей касательной, которая является внутренней для двух из них.

Рассмотрим теперь прямую, разбивающую множество кругов на два. Будем поворачивать ее до тех пор, пока она не упрется в один из кругов, и затем поворачивать ее дальше, сохраняя касание с этим кругом пока она не упрется еще в один. Получится, что для разбиения множества на два достаточно рассматривать общие внутренние касательные кругов, не пересекающие никаких других. Заметим, что такая касательная существует у кругов, если они получены из точек, на отрезке между которыми нет другой точки. Обозначим количество таких пар точек как *x*.

Отметим, что с одной стороны каждое разбиение получается таким образом из двух пар точек (мы можем исходно вращать прямую в одну или в другую сторону). С другой стороны, каждой паре точек соответствует две внутренних касательных соответствующих кругов, а также разбиения, отличающиеся раскраской частей, считаются различными. Следовательно ответ равен x*2*2/2=2x.

Задача D. Dinner Problem

Имя входного файла: **dinner.in**
Имя выходного файла: **dinner.out**
Ограничение по времени: 2 seconds
Ограничение по памяти: 256 megabytes

A group of κ students from Cooking University living in the campus decided that each day of the semester one of them will prepare the dinner for the whole company. The semester lasts for n days.

In sake of fairness they decided that each of the students must prepare the dinner at least once during the semester. Now they wonder how many ways are there to plan the semester — to decide for each day which student would make a dinner that day. Help them to find that out.

Формат входного файла

The input file contains two integer numbers κ and n **(1** $\leq \kappa \leq n \leq$ 100).

Формат выходного файла

Output one number — the number of ways.

Пример

dinner.in	dinner.out
2 3	6

There are six ways: (1,1,2), (1,2,1), (2,1,1), (1,2,2), (2,1,2), (2,2,1).

Разбор

Используем динамическое программирование. Обозначим как *a[i][j]* количество последовательностей длины i, в которых встречается *j* различных чисел. Тогда a[0][0] = 1, а для $i > 0$ выполнено

a[i][j] = ja[i -1][j] + (к - j + l)a[i - l][j - 1]. Первое слагаемое соответствует тому, что мы ставим на позицию *i* число, которое встречалось раньше *(j* вариантов), а второе — что мы ставим на эту позицию новое число, которое ранее не встречалось.

Отметим, также, что в этой задаче ответ может быть большим, поэтому при реализации необходима длинная арифметика.

Задача E. Financial Software

Имя входного файла: financial.in
Имя выходного файла: financial.out
Ограничение по времени: 2 seconds
Ограничение по памяти: 256 megabytes

Jim is writing financial software. He specializes in various financial instruments. The last financial instrument he is writing is history analysis of arbitrage with highly volatile stock.

The idea of the analysis is the following: given a quote of the stock for n consecutive time periods, select a subsequence of the quotes such that any two selected quotes that are adjacent in the selected subsequence are at least κ points apart. For example, if the quotes are 1014,1024,1034,1045,1030, 998 and

$\kappa = 15$, the valid subsequence is 1014,1034, 998.

The selected subsequence must be as long as possible. In the example above the selected subsequence is not optimal, selecting 1014,1045,1030, 998 is better. Given a sequence of quotes, find the longest valid subsequence that can be selected.

Формат входного файла

The first line of the input file contains n ($1 \le n \le 100\ 000$) — the number of quotes, and κ ($1 \le \kappa \le 10^9$). The second line contains n integer numbers — the given sequence of quotes (all quotes are between 1 and 10^9, inclusive).

Формат выходного файла

The first line of the output file must contain l — the length of the optimal subsequence. The second line must contain l integer numbers — the elements of the subsequence.

Пример

financial.in	financial.out
6 15 1014 1024 1034 1045 1030 998	4 1014 1045 1030 998

Разбор

Обозначим элементы исходной последовательности как a[1], a[2],... , *a[n]*.

Рассмотрим, сначала, решение задачи за $O(n^2)$. Используем динамическое программирование, обозначим как l[i] максимальную длину допустимой последовательности, заканчивающуюся в г-м элементе. Тогда l[i] = max{l[j]|j < i,a[j] ≤ a[i] - к или a[j] ≥ a[i] + к} + 1. Максимум в пустом множестве считаем равным 0.

Ответом на задачу будет последовательность, соответствующая максимуму по всем возможным значениям l[i].

Для ускорения решения применим дерево отрезков. Нам надо найти максимум среди значений, меньших либо больших некоторого числа.

Построим дерево отрезков на максимум над всеми возможными значениями котировок, будем хранить в нем длину максимальной последовательности, которая заканчивается в этой котировке. Тогда можно вычислить l[i] за O(log n) следующим образом. Найдем двоичным поиском в массиве возможных котировок максимальное значение *a[j]*, не большее

A[i]-k. После этого сделаем запрос в дереве отрезков между минимальным и этим значением — это соответствует максимальной длине последовательности, заканчивающейся на число меньшее a[i], которой можно добавить a[i] в конец, сохранив ее допустимость.

Аналогично можно найти максимальную длину последовательности, заканчивающейся на число большее a[i], которой можно добавить a[i] в конец.

Вычислив l[i] как максимум из двух полученных значений, увеличенный на единицу, обновляем соответствующее значение в дереве отрезков.

Получаем решение за O(n log n).

Задача F. Minima

Имя входного файла: minima. in
Имя выходного файла: minima.out
Ограничение по времени: 3 seconds
Ограничение по памяти: 256 megabytes

You are given an array x[1... *n*] and a number *m.* For all *i* from 1 to *n* - *m* + 1 find the minimum among x[i], x[i + 1],... , x[i + m - 1] and return the sum of those minima.

Формат входного файла

The first line of the input file contains three integer numbers: *n, m* and к ($1 \le n \le 30\,000\,000$, $1 \le m \le n$, $2 \le к \le \min(n, 1000)$). The second line of the input file contains three integer numbers: a, b and c ($-2^{31} \le a,b,c \le 2^{31} - 1$). The third line of the input file contains к integer numbers: x[1],x[2],...,x[k] ($-2^{31} \le x[i] \le 2^{31} - 1$)

Пример

minima.in	minima.out

10 3 2 1 1 0 0 1	33
1000000 15 5 283471207 23947205 3 17625384 939393931 1838388 912740247 290470294	-1879262596173354

The rest of the array is calculated using the following formula: x[i] = f(a * x[i - 2] + b * x[i - 1] + c). Here f(y) returns such number $-2^{31} \le z \le 2^{31} - 1$ that y - z is divisible by 2^{32}.

Формат выходного файла

Print one integer number — the sum of minima of all subarrays of length *m* of the given array.

Разбор

Задачу можно свести к следующей: реализовать структуру данных типа очередь, которая поддерживает следующие три операции: добавить число в конец, извлечь число из начала и найти минимум в очереди.

Мы приведем два способа реализовать такую структуру данных, в обоих случаях все три операции будут выполняться за амортизированное O(1).

Первый способ. Рассмотрим аналогичную задачу со стеком вместо очереди. Тогда реализовать все три операции за O(1) не представляет труда: будем вместе со стеком хранить стек минимумов, на *i-й* позиции будет храниться минимум в стеке от этой позиции до дна стека. Тогда при добавлении мы кладем на вершину стека минимумов минимум из добавляемого числа и числа на вершине стека минимумов, при извлечении извлекаем также и число с вершины стека минимумов, а минимум в стеке — это просто число на вершине стека минимумов.

Как известно, очередь можно реализовать с использованием двух стеков с амортизированным временем всех операций O(1). Поэтому можно свести задачу с очередью к задаче со стеком.

Второй способ. На этот раз работаем непосредственно с очередью. Будем хранить следующую информацию: p_1— указатель на минимум в очереди, p_2 указатель на минимальный элемент после p_1-го, p_3 — указатель на следующий элемент после p_2-го, и т. д. Хранить эти указатели будем в деке.

При добавлении элемента извлекаем из конца дека элементы, пока они больше добавляемого, и затем добавляем в конец дека указатель на добавленный элемент. При извлечении элемента из головы очереди, если удаляемый элемент не минимальный, то просто удаляем его, а если

минимальный, то также удаляем указатель на него из головы дека. Наконец, минимальный элемент в очереди находится в позиции, на которую указывает указатель в голове дека. Заметим, что все операции реализуются за O(1), кроме удаления элементов из конца дека, но поскольку каждый элемент будет оттуда удален не более одного раза, то амортизированное время всех операций O(1).

Задача G. Move to Front

Имя входного файла: **mtf. in**
Имя выходного файла: **mtf.out**
Ограничение по времени: 2 seconds
Ограничение по памяти: 256 megabytes

Move-to-Front is a method of transforming sequences of positive integer numbers, that is used in some compression algorithms, such as Burrows-Wheeler transform.

Initially all positive integer numbers are organized as an ordered list *L* in their natural order. Consider a sequence a_1, a_2, …, a_n of positive integer numbers. It is encoded as a sequence b_1, b_2, …, b_n in the following way. Let the part of the sequence from a_1 to a_{i-1} be encoded. The position of a_i in the current list *L* is considered. It is assigned to b_i, and a_i is moved to the beginning of the list *L*.

For example, the sequence 3, 3, 3, 2, 2, 2, 2, 2, 3, 1, 3, 3, 2 is encoded as 3, 1, 1, 3, 1, 1, 1, 1, 2, 3, 2, 1, 3.

You are given a sequence a_1, a_2, …, a_n, you must encode it using Move-to-Front method, and output the resulting sequence b_1, b_2, …, b_n.

Формат входного файла

The first line of the input file contains integer number n ($1 \le n \le 100\,000$). The second line contains n integer numbers a_i, ranging from 1 to 10^9.

Формат выходного файла

Output n integer numbers — the sequence b_1, b_2, …, b_n.

Пример

mtf.in	mtf.out
13 3 3 3 2 2 2 2 2 3 1 3 3 2	3 1 1 3 1 1 1 1 2 3 2 1 3

Разбор

Задача на активное использование структур данных.

Рассмотрим отдельно два случая: очередное число уже встречалось, и очередное число ранее не встречалось.

Сначала изучим второй случай. Пусть во входной последовательности встретилось число x, которое ранее не встречалось. Тогда в выходную последовательность надо вывести число $x+a$ где a — количество чисел, больших x, которые встречались ранее. Для нахождения a можно использовать, например, дерево отрезков или сбалансированное дерево.

Теперь рассмотрим как реализовать случай, когда встретившееся число уже встречалось ранее. Рассмотрим два способа.

Первый способ. Используем сбалансированное дерево, причем не используем ключи, а сохраняем фиксированный порядок на ключах (соответствующий порядку в списке), поддерживая размеры поддеревьев. Дополнительно храним ссылки для каждого числа, в какой вершине сбалансированного дерева оно находится (например, с помощью хеш-таблицы). При появлении числа находим количество чисел перед ним в дереве (используя информацию о размерах поддеревьев), затем удаляем соответствующую вершину из дерева и добавляем ее в начало.

Второй способ. Используем при хранении списка дерево отрезков. Храним массив ключей и для каждого ключа помечаем, не был ли он уже удален. На этих пометках (0 или 1) построим дерево отрезков на сумму. Аналогично первому способу храним для каждого числа указатель на его текущее положение в массиве. Когда во входных данных встречается число, делаем запрос о сумме на префиксе до этой позиции (не включая). После этого заменяем пометку этой позиции на 0 (удаляем число из списка) и перемещаем число в начало списка (в последнюю свободную позицию), устанавливая этой позиции пометку в 1. Так же поступаем и с новыми числами, которые появляются в списке.

Задача H. TV Show

Имя входного файла:	**show. in**
Имя выходного файла:	**show.out**
Ограничение по времени:	2 seconds
Ограничение по памяти:	256 megabytes

Recently various TV shows started to gain popularity among young people who like to test their luck**.** One such show is running on ZhTV channel in China**.**

The show proceeds as follows. There is a special game board which has the form of a rectangle with *n* rows and *n* columns. The cells on the main diagonal of the board contain one hieroglyph each. There are also *m* phrases each of which contains *n* hieroglyphs. For each *i* there are at most two phrases such that their i-th hieroglyph is equal to the hieroglyph written in the i-th row of the board.

The player has to put phrases to the rows of the board in such way that if the phrase is put to the i-th row, the i-th hieroglyph of the phrase must coincide with the hieroglyph written in the corresponding cell. Also hieroglyphs in each column must be distinct.

For each phrase put to the board the player gets some small prize, and she gets a super prize if she puts a phrase to every row of the board. Help the player to determine whether she can win the super prize.

Формат входного файла

We will denote hieroglyphs with integer numbers from 1 to 10^9.

The first line of the input file contains *n* — the number of rows on the board ($1 \le n \le 200$). The next line contains *n* integer numbers — hieroglyphs written on the diagonal.

The third line contains an integer number *m* ($n \le m \le 2n$) followed by *m* lines. Each line contains *n* integer numbers — hieroglyphs of the corresponding phrase. For each *i* there are at most two phrases such that their i-th hieroglyph is equal to the hieroglyph written in the i-th row of the board.

Формат выходного файла

If the super prize can be won, print "YES" at the first line of the output file. The second line must contain *n* integer numbers — the phrases to put to the board, from top to bottom. Phrases are numbered from 1 to *m* in order they are given in the input file.

If the super prize cannot be won, print "NO" at the first line of the output file.

Пример

show.in	show.out

4 1 2 3 4 7 1 5 2 5 1 5 4 1 2 3 3 1 3 2 4 1 2 2 2 2 3 3 3 3 5 4 5 4	YES 2 5 6 7
4 1 2 3 4 7 1 5 2 5 1 3 4 1 2 3 3 1 3 2 4 1 2 2 2 2 3 3 3 3 5 4 5 4	NO

Разбор

Покажем, как свести эту задачу к известной задаче об удовлетворимости булевой формулы в

2-КНФ.

Сначала поставим слова в те строки, в которые можно поставить ровно одну фразу. После этого множество фраз, которые можно поставить в некоторые другие строки может тоже уменьшится до одной фразы из-за ограниченияна запрет повторять иероглифы в столбцах. Будем продолжать

этот процесс, пока либо не найдется строки, в которую нельзя поставить ни одной фразы, либо не получится, что в каждую строку можно поставить по две фразы.

В первом случае ответ, очевидно, нет.

Пусть в каждую из оставшихся строк можно поставить по две фразы. Сопоставим каждой строке булеву переменную x_i, пусть $x_i = 0$ соответствует одной из двух фраз, а $x_i = 1$ — другой. Заметим, что если мы поставили одну из фраз, то в других строках возможно останется лишь по одной возможной фразе. С точки зрения булевых переменных эти можно записать в виде $x_i \to x_j$ (либо с отрицаниями, например, $x_i \to \overline{x}_j$, и т. п.) Записав все такие условия и соединив их конъюнкцией, получим булеву формулу в 2-КНФ, алгоритм проверки которой на совместность широко известен.

Например, его можно найти, используя запрос «2-SAT» в Google или Wikipedia.

Задача I. Hard Test

Имя входного файла:	test.in
Имя выходного файла:	test.out
Ограничение по времени:	2 seconds
Ограничение по памяти:	256 megabytes

Andrew is having a hard time preparing his 239-th contest for Petrozavodsk. This time the solution to the problem is based on Dijkstra algorithm and Andrew wants to prepare the hard test for the algorithm.

The Dijkstra algorithm is used to find the shortest path from a source vertex to all other vertices in a graph. The algorithm acts as follows. Let *G* be a weight directed graph with vertex set *V*, edge set *E* and weight function w: $E \to R^+$. Let all vertices be reachable from vertex *s*. The algorithm uses a set of vertices *U*, first initialized as empty. Each vertex is labeled with either an integer number, or with $+\infty$. Initially all vertices are labeled with $+\infty$, and the vertex *s* is labeled with 0. Denote the label of vertex v as d[v].

A step of the algorithm is the following: the vertex with the minimal label that doesn't belong to *U* is selected. Let this vertex be *u*. The vertex *u* is added to the set *U*, and each edge $uv \in 6$ *E* is *relaxed*. The relaxation replaces *d[v]* with min(d[v], d[u] +w(uv)). The algorithm is over when all vertices belong to *U*. If the label of the vertex *v* has changed, the relaxation is said to be *active*.

Now Andrew would like to create a graph with *n* vertices and *m* edges, such that the Dijkstra algorithm makes as many active relaxations as possible. Help him to create such graph. To avoid nondeterminism, each time when selecting a vertex with minimal label among vertices that are not in *U* there must be exactly one vertex with the minimal label.

Формат входного файла

The first line of the input file contains two integer numbers: n and m — the number of vertices and the number of edges in the graph Andrew would like to create ($4 \le n \le 1000$, $n - 1 \le m \le n^2/5$).

Формат выходного файла

Output m lines — the edges of the graph. Each line must contain three integer numbers: the beginning of the edge, the end of the edge and the weight of the edge. All weights must be non-negative and must not exceed 10^6. All vertices must be reachable from vertex 1. If Dijkstra algorithm is run with $s = 1$ there must be maximal possible number of active relaxations among all graphs with n vertices and m edges. There must be no loops and no parallel edges.

Пример

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

Разбор

Задача на творческий поиск. Построим полный ациклический граф, который удовлетворяет всем требованиям: из вершины i в вершину j, где $j > i$ направим ребро веса $j — i — 1$. Простая проверка показывает, что этот граф удовлетворяет требованиям.

Чтобы получить граф с фиксированным числом ребер достаточно удалить из построенного графа $n(n — 1)/2 — m$ ребер.

Задача J. Travel Agency

Имя входного файла: travel.in
Имя выходного файла: travel.out

Ограничение по времени: 2 seconds
Ограничение по памяти: 256 megabytes

Anthony is working in the intergalaxy travel agency. He often meets the requests to find a path from one planet to another using available spaceship routes. Unfortunately, spaceships have only limited number of routes, so passengers usually have to make connections at intermediate planets.

Anthony noticed that some planets are used as intermediate more often than other. He decided to make an investigation — for each planet *A* he would like to know how many pairs of distinct planets *(B, C)* are there, such that any path from planet *B* to planet *C* visits planet *A.* Help him!

Формат входного файла

The first line of the input file contains two integer numbers: *n* and *m* — the number of planets and the number of spaceship routes, respectively ($2 \le n \le 20\,000$, $1 \le m \le 200\,000$). The following *m* lines describe spaceship routes. Each route is bidirectional, and is described with the numbers of planets it connects. It is possible to get from any planet to any other.

Формат выходного файла

Output *n* integer numbers — for each planet *A* output the number of pairs of distinct planets such that any path from one to another goes through *A.*

Пример

travel.in	travel.out
7 9 1 2 1 3 1 4 1 5 1 6 1 7 2 3 4 5 6 7	18 6 6 6 6 6 6

Разбор

Решим сначала задачу для дерева. Подвесим дерево, и заметим, что ответ для вершины, у которой поддеревья имеют размеры $s_1, s_2, \ldots, s_k$, а

количество вершин вне поддерева которой есть $s_0 = n - \sum_i s_i - 1$, равен $\sum_{0 \le i < j \le k} s_i s_j$

Преобразовывая формулу, получаем $((S_0+S_1+\ldots+S_K)^2 - (s_0^2 + s_1^2 + \ldots + s_k^2))/2$. Отметим, что эта величина может быть посчитана для всех вершин дерева за суммарное время O(n).

Теперь осталось свести задачу для произвольного графа к задаче для дерева. Для этого найдем точки сочленения и блоки (см., например, книгу Кормена и других) и построим дерево блоков-точек сочленения. После этого ответ для всех вершин, кроме точек сочленения, равен *n* — 1, а для точек сочленения может быть посчитан с помощью алгоритма для дерева (следует лишь заметить, что у вершин, соответствующих блокам, при подсчете размеров поддеревьев, будет вес, равный количеству вершин — не точек сочленения в блоке).