Станкевич Андрей Сергеевич

РАЗБОР ЗАДАЧ ФИНАЛА МЕЖДУНАРОДНОГО СТУДЕНЧЕСКОГО ЧЕМПИОНАТА МИРА 2003 ГОДА ПО ПРОГРАММИРОВАНИЮ ACM

ЗАДАЧА A. СТРОИТЕЛЬСТВО МОСТОВ

Городской совет Нового Альтонвиля планирует построить несколько пешеходных мостов, соединив ими небоскребы в деловом центре города, чтобы люди могли перемещаться из одного здания в другое, не выходя на улицу. Ваша задача – написать программу, которая помогла бы определить оптимальную конфигурацию мостов.

Деловой центр Нового Альтновиля представляет прямоугольник, разбитый на квадратные плащадки. Каждое здание занимает одну или несколько смежных площадок. Два здания, которые имеют общую точку (достаточно касания углами), считаются связанными, и их не требуется соединять мостами. Мосты можно прокладывать только над линиями – границами площадок. Каждый мост должен представлять собой отрезок прямой линии и должен соединять ровно два здания.

Для заданного набора зданий вы должны определить минимальное количество мостов, которое требуется, чтобы соединить все здания. Если это невозможно, требуется найти решение, которое минимизирует количество не связанных групп зданий. Среди возможных решений с одним и тем же количеством мостов требуется найти решение, суммарная длина мостов в котором минимальна. Два моста могут пересекаться, но в таком случае они должны быть проложены на разной высоте, и пешеход не может перейти с одного моста на другой по середине мостов.

На рисунке 1 показаны несколько конфигураций зданий. Город 1 содержит пять зданий, которые могут быть соединены с ис-

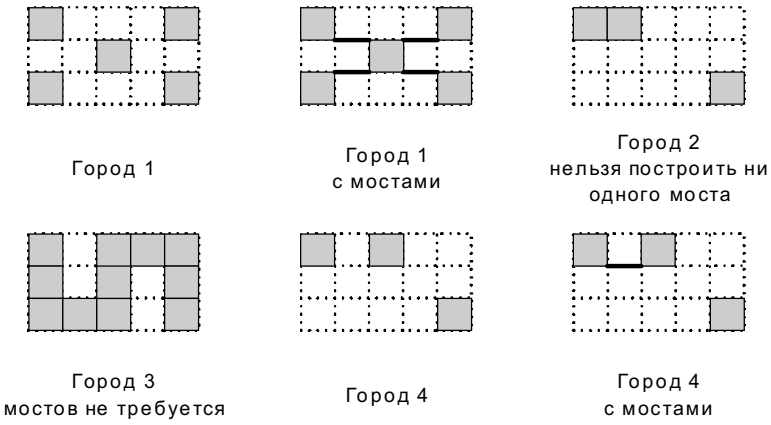


Рисунок 1.

пользованием четырех мостов с суммарной длиной в 4 единицы. В городе 2 нельзя построить ни одного моста, поскольку никакие два здания не находятся на одной линии сетки. В городе 3 мостов не требуется, поскольку в нем имеется только одно здание. В городе 4 лучшее, что может быть достигнуто, – это две не связанные группы зданий, в одной из которых два здания соединены мостом длины один.

Входные данные

Входной файл содержит описания нескольких прямоугольных городов.Описание каждого города начинается с двух чисел r и c, представляющих собой размер города в направлении с севера на юг и с запада на восток, соответственно ($1 \le r \le 50$, $1 \le c \le 50$). Затем следует ровно r строк, каждая из которых содержит ровно по c символов. Символ «#» означает, что соответствующая клетка занята зданием, а символ «.» – что она свободна.

Последняя строка входного файла содержит два нуля.

Выходные данные

Для каждого города выведите в выходной файл две или три строки, как показано в примере. Первая строка должна содержать номер города. Если в городе менее двух зданий, вторая строка должна содержать фразу «No bridges are needed.» Если в городе имеется не менее двух зданий, но ни одного моста построить нельзя, вторая строка должна содержать фразу «No bridges are possible.» В противном случае вторая строка должна содержать фразу «N bridges of total length L», где N – количество мостов, которое требуется построить, а L – их суммарная длина в оптимальном решении (если $N = 1$, то вместо слова «bridges» следует использовать слово «bridge»). Если решение оставляет две или более группы зданий не связанными, то третья строка вывода для данного города должна содержать количество не связанных групп зданий.

Выводите пустую строку между ответами на тестовые примеры. Используйте форматирование, приведенное в примере.

Пример

Входной файл	Выходной файл
3 5 #...# ..#.. #...# 3 5 ##... ....# 3 5 #.### #.#.# ###.# 3 5 #.#.. ....# 0 0	City 1 4 bridges of total length 4 City 2 No bridges are possible. 2 disconnected groups City 3 No bridges are needed. City 4 1 bridge of total length 1 2 disconnected groups

Решение.

Заметим сначала, что условия, которые накладываются на набор мостов, приведены в условии в некотором смысле в порядке «убывания значимости». А именно, критерий, приведенный последним, в соответствии с которым требуется построить множество мостов такое, что их суммарная длина минимальна, и определяет суть решения задачи. А именно, задача, которую требуется решить, сводится к поиску минимального остовного дерева в графе.

Будем однако анализировать требования последовательно.

Для каждых двух зданий определим, можно ли их связать с помощью моста. Построим граф, вершинами которого будут здания. Свяжем две вершины ребром, если здания, им соответствующие, можно соединить мостом. Ясно, что общее число не связанных групп зданий, которые образуются после строительства мостов, равно числу компонент связности построенного графа.

Пусть мы решили построить некоторое множество мостов. Скажем в этом случае, что мы *выбрали* соответствующие этим мостам ребра в построенном графе. Тогда условие, что общее количество построенных мостов должно быть минимально, соответствует в нашей модели утверждению, что в

каждой компоненте связности множество выбранных ребер образует дерево, называемое также *остовным деревом*. Поскольку дерево с n вершинами содержит $n - 1$ ребро, то общее число выбранных ребер равно $n - k$, где k – число компонент связности графа.

Обратимся теперь к основному условию. Взвесим наш граф, назначив каждому ребру в качестве веса минимальную возможную длину моста, которым можно соединить здания, соответствующие вершинам, которые это ребро соединяет. Ясно, что, поскольку построение остовных деревьев в компонентах связности графа независимо, чтобы минимизировать общую длину мостов, надо по отдельности минимизировать суммарную длину мостов в каждой компоненте связности. Таким образом, нам требуется в каждой компоненте связности решить задачу о нахождении остовного дерева минимального веса, или *минимального остовного дерева.*

Известно много алгоритмов, позволяющих решить данную задачу. Все они в той или иной мере могут быть отнесены к категории так называемых *жадных* алгоритмов, поскольку на каждом шаге добавляют к остовному дереву минимальное по весу ребро в некотором классе, следуя, таким образом, принципу локальной оптимизации. Рассмотрим два наиболее известных эффективых алгоритма построения минимального остовного дерева, известных как алгоритмы Прима и Краскала соответственно.

Для обоснования алгоритмов нам потребуется понятие разреза. Разрезом называют такое множество C ребер графа, что вершины графа можно разбить на два непересекающихся множества S и T таких, что ребро принадлежит C тогда и только тогда, когда один его конец принадлежит S, а другой – T (то есть в C вошли все ребра, соединяющие S и T, и никакие другие). Оба алгоритма (как и большинство других алгоритмов построения минимального остовного дерева) используют следующую теорему, которую часто называют теоремой о разрезе и минимальном остовном дереве.

Пусть выбрано некоторое множество S ребер графа, которое является подмножеством множества ребер некоторого минимального остовного дерева (то есть множество S можно дополнить до некоторого минимального остовного дерева). Назовем ребро $e \notin S$ безопасным для множества S, если $S \cup \{e\}$ также является подмножеством множества ребер некоторого минимального остовного дерева нашего графа (то есть можно добавить e в множество S, и полученное множество также можно будет дополнить до некоторого минимального остовного дерева). Пусть C – разрез, такой что $C \cup S = \varnothing$ и e – одно из минимальных по весу ребер в C. Тогда e безопасно для S.

Доказательство этой теоремы можно найти, к примеру, в [1].

Оба алгоритма построения остовного дерева действуют в соответствии с этой теоремой – они начинают с пустого множества, которое, без сомнения, можно дополнить до минимального остовного дерева и поочередно добавляют в текущее множество минимальное ребро из некоторого разреза. Различие в алгоритмах заключается в том, какой разрез используется.

Алгоритм Прима в качестве разреза использует множество ребер, один из концов которых принадлежит множеству вершин, достижимых по текущему остовному дереву из первой вершины, а другой – нет. Так, сначала это множество ребер, инцидентных первой вершине. Минимальное из них по теореме безопасно для пустого множества, добавим его в наше остовное дерево. Теперь множество вершин, достижимых из первой вершины, состоит из двух вершин. Соответственно, наш разрез – это множество ребер, инцидентных одной из них, но не инцидентных другой. Найдем теперь в нем минимальное ребро. Продолжая таким образом, постепенно построим минимальное остовное дерево. Действительно, по приведенной теореме, наше множество ребер в любой момент является подмножеством ребер некоторого минимального остовного дерева. Каждый шаг добавляет в него по ребру, когда ребер будет $n - 1$, где n – количество вершин в графе, наше множество ребер будут стягивать n вершин, то есть будет остовным деревом нашего графа.

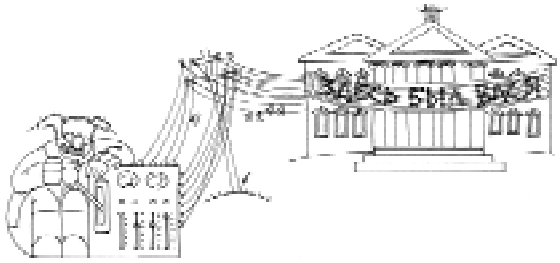
Алгоритм Краскала внешне использует более нетривиальный разрез, но при этом оказывается даже проще алгоритма

Прима. Пусть S – множество ребер, которое мы дополняем до минимального остовного дерева. Найдем минимальное ребро e в графе, концы которого не соединены путем в S. Тогда существует разрез, содержащий e и не пересекающийся с S (достаточно взять все ребра, не входящие в S и найти в нем минимальное по включению множество, содержащее e, такое, что при его удалении из графа он распадается на компоненты связности). Ясно, что в этот разрез не входят ребра, концы которых соединены путем в S, значит, e минимально и в этом разрезе, а тогда оно безопасно для S. Отметим, что нам не требуется искать сам этот разрез, достаточно его существования. Отсюда получаем следующий алгоритм. Начнем с пустого множества. Найдем минимальное ребро в графе. Добавим его в S. Найдем минимальное ребро в графе, концы которого не соединены путем в S. Добавим его в S. Продолжаем так, пока не получится минимальное остовное дерево.

Итак, для решения задачи осталось построить вспомогательный граф. Оказывается, что процесс построения этого графа можно удивительным образом совместить с процессом построения минимальных остовных деревьев во всех его будущих компонентах связности с помощью алгоритма Краскала! Действительно, алгоритму Краскала для работы требуется список ребер в возрастающем порядке их веса. Будем действовать следующим образом. Рассмотрим все точки с целыми координатами, лежащие на границах зданий. Из каждой из этих точек начнем виртуально строить мосты во всех возможных направлениях (то есть в таких направлениях, чтобы мосты не пересекались со зданием, от которого они строятся). Сначала построим мосты длины 1. Затем удлиним их до длины 2 и т. д. При этом, как только мост доходит до некоторого здания, проверяем, не связаны ли эти здания уже построенными мостами. Если нет, то свяжем их вновь построенным мостом и добавим его в остовное дерево.

Ясно, что приведенный алгоритм является как раз реализацией алгоритма Краскала для описанного графа.

ЗАДАЧА B. ЛАМПОЧКИ



Новый голливудский театр Коллизей Культуры и Кино[1] украшен гигантской гирляндой, управляемой компьютером, в которой используются тысячи лампочек. Каждый ряд лампочек в гирлянде управляется набором переключателей, которые контролируются с использованием специальной компьютерной программы. К сожалению, электрики установили неверный набор переключателей, а сегодня вечером состоится открытие театра. Ваша задача – написать программу, которая бы заставила переключатели работать корректно.

Ряд лампочек в гирлянде состоит из n лампочек и контролируется n переключателями. Лампочки и переключатели занумерованы слева направо от 1 до n. Каждая лампочка может быть либо включена, либо выключена. Каждый тестовый пример в этой задаче будет содержать начальную и желаемую конечную конфигурации лампочек.

Исходно предполагалось, что каждый переключатель будет контролировать одну лампочку. Однако ошибка в электронике привела к тому, что каждый переключатель, помимо своей лампочки, также изменяет состояние двух (или одной, если основная лампочка находится с краю) соседних с ней, как показано на рисунке 2. Так, самый левый переключатель ($i = 1$) изменяет состояние двух самых левых лампочек (1 и 2), а самый правый ($i = n$) – двух самых правых ($n - 1$ и n). Каждый из оставшихся переключателей ($1 < i < n$) изменяет состояние трех лампочек с номерами $i - 1$, i и $i + 1$. Исключение составляет единственный случай, когда имеется ровно одна лампочка и один переключатель. В этом случае этот переключатель контролирует эту лампочку. Таким образом, если, к примеру, лампочка

[1] Atheneum of Culture and Movies, ACM.

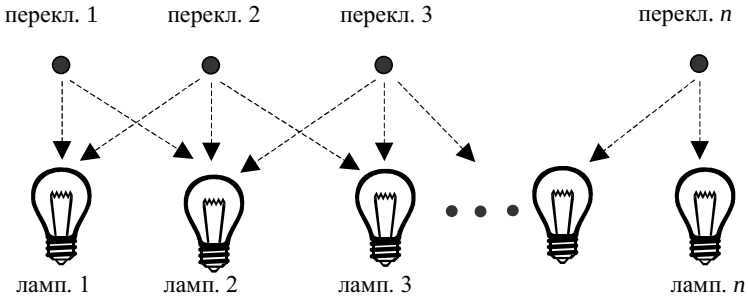


Рисунок 2.

1 включена, а лампочка 2 выключена, то при нажатии на переключатель 1 лампочка 1 выключится, а лампочка 2 включится. Определим минимальную стоимость перехода из одной конфигурации к другой как минимальное количество переключателей, которое требуется нажать, чтобы получить из начальной конфигурации конечную.

Можно представить состояние ряда лампочек как двоичное число, где 0 означает, что лампочка выключена, а 1 – что лампочка включена. Так, в частности, двоичное число 01100 представляет собой ряд из пяти лампочек, среди которых вторая и третья включены, а остальные выключены. Можно превратить эту конфигурацию в конфигурацию 10000, нажав последовательно на переключатели 1, 4 и 5, но дешевле сделать это, просто изменив состояние переключателя 2.

Напишите программу, которая определит, на какие переключатели следует нажать, чтобы перевести начальную конфигурацию лампочек в конченую за минимальную стоимость. В некоторых случаях подобное может оказаться невозможным. Отметим, что для компактности представления двоичные числа записываются как десятичные, то есть вместо 01100 и 10000 используются числа 12 и 16, соответственно.

Входные данные

Входной файл содержит несколько тестовых примеров. Каждый тестовый пример занимает одну строку. Строка содержит два неотрицательных целых десятичных числа, по крайней мере, одно из которых положительно и каждое из которых содержит не более 100 цифр. Двоичная запись первого числа представляет собой начальную конфигурацию, а второго – конечную.

Чтобы избежать проблем с ведущими нулями, будем предполагать, что первая лампочка либо в начальной, либо в конечной конфигурации включена. Во входном файле не будет пробелов в начале или конце строки, числа записаны без ведущих нулей и разделены ровно одним пробелом.

Последняя строка входного файла содержит два нуля.

Выходные данные

Для каждого тестового примера выведите в выходной файл одну строку. Она должна содержать номер тестового примера и десятичное число, двоичное представление которого кодирует набор переключателей, которые требуется нажать, чтобы из начальной конфигурации получить конечную. В двоичном представлении этого числа самый правый (наименее значимый) бит должен соответствовать переключателю с номером *n*, 1 означает, что переключатель следует нажать, а 0 – что нет. Если решения не существует, то вместо этого числа следует вывести слово «impossible». Если имеется более одного решения, выведите то, которое представляется меньшим десятичным числом.

Разделяйте вывод для тестовых примеров пустой строкой. Используйте формат, приведенный в примере.

Пример

Входной файл	Выходной файл
12 16 1 1 3 0 30 5 7038312 7427958190 4253404109 657546225	Case Number 1: 8 Case Number 2: 0 Case Number 3: 1 Case Number 4: 10 Case Number 5: 2805591535 Case Number 6: impossible

Решение.

Решение этой задачи не представляет особой сложности. Сначала заметим, что случаи, когда имеется ровно одна или ровно две лампочки, можно обработать отдельно, поэтому не будем на них останавливаться – в них несложно рассмотреть все случаи. Итак, пусть дано, по крайней мере, три лампочки.

Первая подзадача, которая возникает в процессе решения задачи, заключается в чтении ввода. Как указано в условии, ввод может содержать десятичные числа длиной до 100 цифр, что соответствует примерно 350 двоичным. Такое число не помещается в стандартный целочисленный тип данных языков программирования, доступных для решения задач, поэтому следует разобрать число вручную. Считав его как строку, преобразуем его в двоичную запись. По этой записи несложно восстановить конфигурацию лапочек. Небольшая дополнительная проблема заключается в том, что количество лампочек заранее неизвестно, а определяется как длина двоичной записи более длинного из чисел, встречающихся во входном файле. Впрочем, эта проблема легко решается.

Итак, перейдем к следующему этапу решения задачи – собственно поиску минимальной по стоимости последовательности операций, переводящих множство лампочек из одной конфигурации в другую. Оказывается, существует не более двух возможных последовательностей, выполняющих нашу задачу. Как следствие, обе эти последовательности можно проанализировать и выбрать среди них оптимальную.

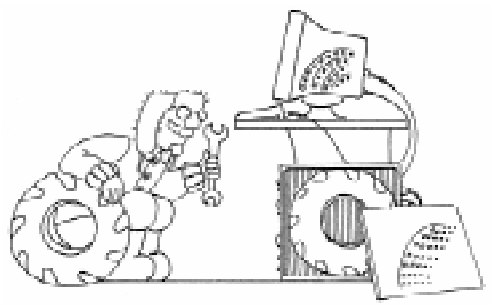
Несложный анализ показывает, что единственный произвол, который имеется при выборе набора переключателей для решения задачи, заключается в следующем: использовать ли первый переключатель. Обозначим начальное состояние i-ой лампочки как $a[i]$, ее конечное состояние как $b[i]$, а решение об использовании i-го переключателя как $s[i]$. Соответственно, для лампочки будем полагать, что соответствующее значение равно 1, если она включена, и 0 – в противном случае, а для переключателя будем полагать $s[i] = 1$, если он используется и 0 – в противном случае. Обозначим через n общее количество лампочек.

Итак, пусть мы выбрали значение для $s[1]$. Заметим, что должно выполняться $b[1] = a[1]$ xor $s[1]$ xor $s[2]$. Поскольку $a[1]$, $s[1]$ и $b[1]$ известны, а $s[2]$ выражается из указанного выражения с помощью равенства $s[2] = s[1]$ xor $a[1]$ xor $b[1]$, то $s[2]$ полностью определяется значением $s[1]$. Аналогично, $s[3]$ полностью определяется $s[1]$ и $s[2]$: $s[3] = s[1]$ xor $s[2]$ xor $b[2]$ xor $a[2]$. Вообще, при любом $1 < i < n$ выполняется равенство $s[i + 1] = s[i - 1]$ xor $s[i - 2]$ xor $a[i]$ xor $b[i]$. Таким образом, выбрав значение $s[1]$, мы полностью восстанавливаем необходимость использования остальных переключателей. Отметим, что пока не использовались значения $a[n]$ и $b[n]$. Они требуются для проверки корректности построенной последовательности. А именно, по построению, последовательность правильно устанавливает все лампочки, кроме последней. Правильность установки последней следует проверить отдельно, она устанавливается правильно, если выполняется равенство $b[n] = a[n]$ xor $s[n - 1]$ xor $s[n]$.

Построив две возможные последовательности переключателей, выбираем среди них допустимые, а среди последних ту, которая имеет меньшую стоимость. После этого осталось превратить последовательность в десятичное число методом, аналогичным тому, которым десятичное число было превращено в двочиное при чтении входных данных. Заметим, что требование, чтобы при существовании двух решений использовалось то, которое представляется меньшим десятичным числом, несложно отследить на этапе выбора минимума – действительно, из двух возможных решений одно имеет старший бит равным единице, а другое – нулю. Ясно, что при равенстве следует выбрать решение с нулевым старшим битом.

ЗАДАЧА C. ШИНА

Последие разработки в области конфигурируемых микропроцессоров используют одну шину, которая определенным об-



разом проложена по чипу. Компоненты процессора, которые могут быть в произвольной части чипа, присоединены к *точкам входа* шины и таким образом могут общаться между собой.

Определенные исследования показали, что удобным является прокладка шины в форме рекурсивно определяемой «SZ»-кривой, также известной как «S-образная кривая Пеано». Два примера такой кривой приведены на рисунке 3. Каждая кривая рисуется на единичном квадрате. Кривая порядка 1, изображенная слева, по форме напоминает букву «S» и состоит из сегментов, соединяющих точки (0,0), (1,0), (1,0.5), (0,0.5), (0,1) и (1,1) в указанном порядке. Каждая горизонтальная чать «S» или «Z» в кривой вдвое длиннее, чем каждая вертикальная часть. Для кривой порядка 1 длина вертикальной части (обозначим ее как *len*), есть 0.5.

Кривая порядка 2, изображенная справа, состоит из 9 уменьшенных копий кривых порядка 1 (четыре из которых были зеркально отражены относительно вертикальной оси, в результате чего они превратились в «Z»-кривые). Эти копии соединены сегментами длины *len*, как показано пунктирными линиями. Поскольку ширина и высота кривой порядка 2 есть $8 \times len$, а кривая изображается на единичном квадрате, то для кривой порядка 2 имеет место $len = 0.125$.

Кривая порядка 3 состоит из 9 уменьшенных копий кривой порядка 2 (четыре из которых зеркально отражены относительно вертикальной оси), соединенных сегментами так же, как это сделано в кривой второго порядка. Кривые высших порядков получаются тем же способом. Точки входа, к которым присоединяются компоненты процессора, расположены вдоль кривой каждые *len* единиц. Первая точка входа расположена в точке (0, 0), а последняя – в точке (1, 1). Всего на кривой порядка k имеется 9^k точек входа, а общая длина шины – $(9^k - 1) \times len$.

Ваша цель – написать программу, которая бы определяла, какое расстояние проходит сигнал между двумя компонентами процессора. Каждая компонента задается как пара координат x, y, $(0 \le x \le 1, 0 \le y \le 1)$, где x – расстояние от левой стороны чипа до этой компоненты, а y – расстояние от нижней стороны чипа. Каждая компонента присоединяется к точке входа, ближайшей к ней, с помощью отрезка прямой. Если имеется несколько точек входа, находящихся на равном расстоянии от компоненты, то используется точка входа с меньшей координатой x и меньшей координатой y. Общее расстояние, проходимое сигналом между двумя компонентами, есть сумма длин отрезков, соединяющих компоненты с шиной, и длина шины между двумя точками входа, к которым присоединяются компоненты. Например, расстояние между компонентами, находящимися в точках (0.5, 0.25) и (1.0, 0.875) на чипе, выполненном с использованием кривой первого порядка, равно 3.8750 единицам.

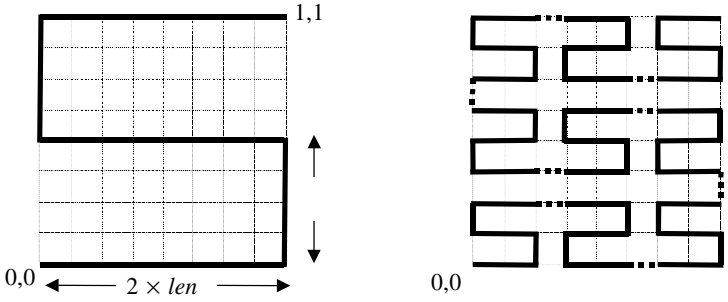


Рисунок 3.

Входные данные

Входной файл содержит несколько тестовых примеров. Для каждого тестового примера входные данные состоят из целого числа, задающего поря-

док SZ-кривой, используемой в шине (порядок не превышает 8), и затем четырех вещественных чисел x_1, y_1, x_2, y_2, задающих координаты компонент процессора, которые следует соединить. Несмотря на то, что обычно при создании процессоров каждая компонента должна иметь собственную уникальную точку присоединения к шине, ваша программа не должна использовать этот факт.

Последняя строка входного файла содержит единственное число 0.

Выходные данные

Для каждого тестового примера выведите его номер и затем расстояние между компонентами процессора, притом что они соединены указанным способом. Расстояние следует выводить с точностью до 4 знаков после запятой.

Используйте формат, подобный использованному в примере. Оставляйте пустую строку между выводом для различных тестовых примеров.

Пример

Входной файл	Выходной файл
1 0.5 .25 1 .875 1 0 0 1 1 2 .3 .3 .7 .7 2 0 0 1 1 0	Case 1. Distance is 3.8750 Case 2. Distance is 4.0000 Case 3. Distance is 8.1414 Case 4. Distance is 10.0000

Решение.

Прежде всего выполним следующую операцию: умножим координаты всех точек, встречающихся во входных данных, на $9^k - 1$, где k – порядок кривой во входных данных. После этой операции отрезок длины len станет отрезком единичной длины, а все точки входа будут соответствовать точкам с целыми координатами, лежащим внутри квадрата со стороной $9^k - 1$. Теперь, найдя расстояние между точками вдоль увеличенной кривой, для получения исходного расстояния достаточно уменьшить ответ в $9^k - 1$ раз.

Сначала найдем координаты точки входа, к которой следует подключить заданную точку. Для точки с координатами (x, y) возможными кандидатами, в порядке предпочтения, являются $(\lfloor x \rfloor, \lfloor y \rfloor)$, $(\lfloor x \rfloor, \lceil y \rceil)$, $(\lceil x \rceil, \lfloor y \rfloor)$ и $(\lceil x \rceil, \lceil y \rceil)$. Среди них, в соответствии с условием, следует выбрать ближайшую точку к исходной, а если таких несколько, то следующую раньше в указанном списке.

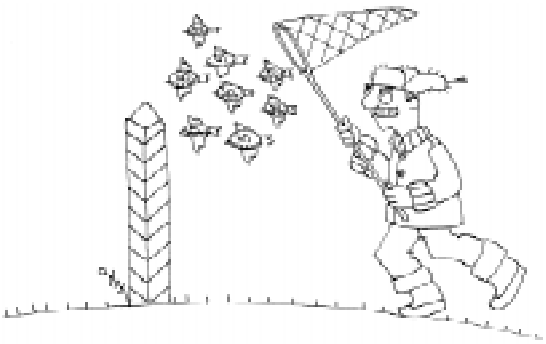
Посмотрим теперь, как определить расстояние вдоль кривой между двумя точкам входа. Для простоты введем понятие кривой нулевого порядка, представляющей собой одну точку плоскости. Расстояние между двумя точками кривой нулевого порядка определить несложно – оно равно нулю.

Рассмотрим теперь, как можно свести задачу к задаче для кривой более низкого порядка. Скажем, что кривая разбита на сегменты, соответствующие копиям кривой предыдущего порядка, из которых она составлена. Соответственно, для кривой, лежащей в квадрате $\left[0..9^{k-1}\right]\times\left[0..9^{k-1}\right]$ сегменты для i и j, принимающих значения от 0 до 2 располагаются в квадратах вида $\left[9^{k-1}\cdot 3i..9^{k-1}\cdot(3i+3)-1\right]\times$ $\times\left[9^{k}\cdot 3j..9^{k-1}\cdot(3j+3)-1\right]$. Рассмотрим две наших точки входа. Если они лежат внутри одного сегмента, то достаточно вычислить расстояние между ними на кривой меньшего порядка, лежащей внутри этого сегмента (чтобы использовать рекурсивный подход, надо сделать параллельный перенос плоскости и, возможно, отражение относительно вертикальной оси). В противном случае сделаем следующее. Занумеруем сегменты от 0 до 8 в порядке их следования вдоль кривой. Пусть одна из точек лежит в i-ом сегменте, а другая – в j-ом, причем $i < j$. Тогда расстоянием между точками вне этих сегментов есть $9^{k-1}(j-i)+1$, а внутри этих сегментов следует решить задачу о расстоянии от заданной точки до последней (или первой, соответственно) вдоль кривой точки входа внутри этого сегмента. Таким образом, задача полностью решена.

Отметим, что эта задача была одной из самых сложных технически задач на чемпионате, немногие команды с ней справи-

лись. В частности, одной из сложностей этой задачи является внешне очевидная, но неверная идея, что длина *len* убывает как степень двойки (первые два порядка соответствуют 1/2 и 1/8), несмотря на то, что жюри явно подчеркивает в условии, что длины отрезков связаны со степенями девятки.

ЗАДАЧА D. ЕВРОДИФФУЗИЯ



1 января 2002 года двенадцать европейских стран отказались от своих национальных валют в пользу новой валюты евро. Таким образом, больше не используются франки, марки, лиры, гульдены, кроны, ... – только евро, во всей еврозоне. Одни и те же банкноты используются во всех странах. А как насчет монет? Здесь ситуация другая: каждая страна имеет определенное право выпускать свои собственные монеты евро.

1 января 2002 года единственными монетами евро в Париже были французские монеты. Вскоре первые нефранцузские монеты евро появились в Париже. Рано или поздно можно ожидать, что все типы монет равномерно распределятся по всем двенадцати странам. (На самом деле это не совсем верно. Все страны продолжают штамповать и распределять монеты своего формата. Так что даже в случае стабилизации ситуации в Берлине будет больше немецких монет). Итак, как скоро первые финские или ирландские монеты будут в обращении на юге Италии? Сколько времени пройдет, прежде чем монеты всех видов будут доступны во всех странах?

Ваша задача – написать программу, которая будет симулировать распространение монет евро по Европе с использованием сильно упрощенной модели. Остановим наш взгляд на монетах только одного номинала. Пусть Европейские города представляют собой точки на прямоугольной сетке. Каждый город может иметь до четырех соседей (по одному на севере, востоке, юге и западе). Каждый город принадлежит некоторой стране, которая представляет собой прямоугольную часть плоскости. Рисунок 4 демонстрирует пример карты с тремя странами и 28 городами. Граф стран связен, но страны могут граничить с «белыми пятнами», которые соответствуют морям или, скажем, странам, не вошедшим в еврозону, таким как Швейцария или Дания. Изначально каждый город имеет один миллион монет вида той страны, которой он принадлежит. Каждый день некоторая часть монет, в зависимости от баланса этого города, перемещается в каждый из соседних городов. А именно, в каждый из соседних городов перемещается по одной монете на каждую полную тысячу монет каждого вида.

Скажем, что город *насыщен*, если в нем имеется, по крайней мере, одна монета каждого вида. Страна *насыщена*, если каждый ее город насыщен. Ваша программа должна определить, через какое время каждая страна окажется насыщена.

Входные данные

Входной файл состоит из нескольких тестовых примеров. Первая строка каждого тестового примера содержит количество стран ($1 \le c \le 20$). Следующие с строк имеют формат *name xl yl xh yh*, где *name* – название страны, слово, не длиннее 25 символов; *xl*, *yl* – координаты западного из самых

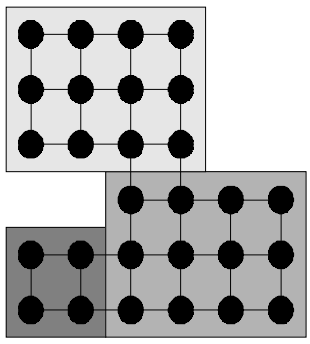


Рисунок 4.

южных городов этой страны, а *xh*, *yh* – координаты самого восточного из самых северных городов ($1 \le xl \le xh \le 10$, $1 \le yl \le yh \le 10$).

Последняя строка входного файла содержит единственное число 0.

Выходные данные

Для каждого тестового примера выведите строку, содержащую номер тестового примера, после которой выведите для каждой страны ее название и количество дней до насыщения этой страны. Упорядочите страны по возрастанию количества дней, требуемых для их насыщения. Если две страны имеют одно и то же необходимое количество дней, упорядочите их в алфавитном порядке их названий.

Следуйте формату, приведенному в примере.

Пример

Входной файл	Выходной файл
3 France 1 4 4 6 Spain 3 1 6 3 Portugal 1 1 2 2 1 Luxembourg 1 1 1 1 2 Netherlands 1 3 2 4 Belgium 1 1 2 2 0	Case Number 1 Spain 382 Portugal 416 France 1325 Case Number 2 Luxembourg 0 Case Number 3 Belgium 2 Netherlands 2

Решение.

Это одна из самых простых задач на чемпионате. Для ее решения требуется просто проэмулировать указанный процесс.

Отметим, что типичной ошибкой в подобных задачах, особенно у начинающих программистов, является перезапись исходных данных. Так, если хранить количество евро во всех точках плоскости в некотором массиве и в нем же проводить обновления количества монет по итогам очередного дня, результат, разумеется, может отличаться от правильного. Как всегда в таких случаях, следует завести вспомогательный массив и записывать новое количество монет в городе в него, после чего перенести информацию из вспомогательного массива в основной. В остальном решение этой задачи не представляет сложности.

ЗАДАЧА E. ЗАТЫКАЯ ДЫРЫ

Можно ли накрыть круглую дыру квадратной крышкой? Можно, если крышка достаточно большая. Разумеется, это будет не точное совпадение, но тем не менее дыру можно закрыть полностью.

Ассоциация Изготовителей Крышек[2] состоит из нескольких компаний, которые изготавливают крышки для всех типов дыр – люков, дыр в асфальте, колодцев, окопов, траншей, входов в пещеры, дыр на задних дворах, вырытах собаками, чтобы закопать кости, и многих других. Ассоциация хочет получить программу, которая бы определяла, можно ли с использованием заданной крышки полностью закрыть заданную дыру. В настоящее время их интересуют исключительно крышки и дыры, которые представляют собой прямоугольные многоугольники (то есть многоугольники, все внутренние углы которых равны 90 или 270 градусам). Более того, и дыра, и крышка уже установлены относительно координатных осей, и не допускается их вращение друг относительно друга, только параллельный перенос.

Входные данные

Входной файл содержит несколько описаний крышек и дыр. Первая линия каждого описания содержит два целых числа *h*

[2] Association of Cover Manufacturers, ACM.

и c ($4 \le h \le 50$, $4 \le c \le 50$) – количество вершин у многоугольника, соответствующего дыре, и многоугольника, соответствующего крышке, соответственно. Каждая из следующих h строк содержит по два числа x и y, которые представляют собой координаты вершин многоуольника дыры в порядке их обхода по многоугольнику. Следующие c строк содержат аналогичное описание крышки. Оба многоугольника прямоугольные, стороны многоугольников параллельны осям координат. Оба многоугольника имеют положительную площадь и не пересекаются.

Последняя строка входного файла содержит два нуля.

Выходные данные

Для каждого примера выведите сначала его номер. Если дыра может быть полностью накрыта крышкой, выведите «Yes», в противном случае выведите «No». Помните, что крышка может выходить за границы дыры, но вся дыра должна быть накрыта. Следуйте формату вывода, приведенному в примере.

Пример

Входной файл	Выходной файл
4 4 0 0 0 10 10 10 10 0 0 0 0 20 20 20 20 0 4 6 0 0 0 10 10 10 10 0 0 0 0 10 10 10 10 1 9 1 9 0 0 0	Hole 1: Yes Hole 2: No

Решение.

Эта задача – пожалуй, самая сложная задача на чемпионате. Отметим, впрочем, что на последнем чемпионате мира, в отличие, скажем, от предыдущего года, не было откровенно безнадежных задач, в частности, каждую задачу решила хотя бы одна команда.

Решение этой задачи можно разделить на две независимые части. Первая из них состоит в выборе всех возможных относительных положений крышки и дыры. Вторая заключается в том, чтобы при заданном взаимном расположении крышки и дыры установить, покрывается ли дыра крышкой. Опишем решение каждой из указанных подзадач.

Первая подзадача решается достаточно просто. Верно следующее утверждение: проведем через каждую сторону дыры и крышки прямую. Следует перебирать лишь такие взаимные положения крышки и дыры, при которых существует пара совпадающих горизонтальных прямых для сторон крышки и дыры, соответственно, а также существует аналогичная пара совпадающих вертикальных прямых. Общее число как горизонтальных, так и вертикальных сторон и крышки и дыры не превышает 25, следовательно, общее число взаимных положений не превышает 25^4.

Более сложно решить вторую подзадачу. Отметим сначала неверные подходы к ее решению. Так, например, недостаточно покрытия всех вершин, или всех середин сторон дыры. Правда, оказывается достаточно, чтобы все стороны дыры были полностью покрыты – в силу односвязности крышки, при этом будет полностью покрыта и ее внутренность. Таким образом, задача свелась к следующей: проверить, что заданный отрезок полностью покрывается заданным многоугольником. Решим ее.

Пусть, например, рассматриваемый отрезок горизонтален. Рассмотрим части, на которые он разбивается вертикальными прямыми, проходящими через стороны крышки. Достаточно, чтобы середина каждой из указанных частей была покрыта. Таким об-

разом, осталось научиться проверять, что точка принадлежит многоугольнику. Для этого известно много различных способов.

Один из способов заключается в следующем: проведем из заданной точки в произвольном направлении луч. Тогда, если этот луч не проходит через вершины многоугольника, то точка лежит внутри многоугольника, если луч пересекается со сторонами многоугольника нечетное число раз. Добиться того, чтобы луч не проходил через вершины многоугольника, можно различными способами. Например, можно выбрать случайным образом направление луча и, если он прошел через вершину многоугольника, выбрать направление заново.

Другой способ заключается в том, чтобы сложить ориентированные углы, под которыми из заданной точки видны стороны многоугольника, при обходе его по или против часовой стрелки. Если сумма этих углов равна 0, то точка лежит вне многоугольника, в противном случае – внутри него. При этом подходе отдельно следует рассмотреть случай, когда точка лежит на границе многоугольника.

Литература.

1. *Кормен Т., Лейзерсон Ч., Ривест Р.* Алгоритмы: построение и анализ. М.: МЦНМО, 2000.

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