**Замена тегов**

**Описание**

В языке xml у каждого открывающего тега должен быть свой закрывающий. Последовательность xml-тегов является называется **правильной**, если она может быть построена по следующим правилам:

пустая строка является правильной последовательностью xml-тегов;

если A - правильная последовательность xml-тегов, то <tag>A</tag> - тоже правильная последовательность xml-тегов, где <b>tag</b> - некоторый тег языка xml

если A и B - правильные последовательности xml-тегов, то AB - тоже правильная последовательность xml-тегов.

Примеры правильных последовательностей xml-тегов - "", "<a><b></b><c></c></a>", "<tag1></tag1><tag2><tag3></tag3></tag2><tag4></tag4>" ,"<x1></x1><x2><a1></a1><s1><a1></a1><a1></a1></s1></x2>".

Рассмотрим последовательность xml-тегов, содержащую только теги <a> и <b>. Пусть разрешается выполнять следующие операции: заменить открывающий тег <b> на произвольное число открывающих тегов <a> и заменить закрывающий тег </b> на произвольное количество закрывающих тегов </a>. Разрешается при замене создавать ноль тегов, то есть просто удалять соответствующий </b> тег.

**Требуется** с использованием указанных операций получить из заданной строки минимальную по длине правильную последовательность xml-тегов, состоящую только из тегов и .

**Формат входного файла (tags.in)**

Входной файл содержит одну строку, состоящую только из тегов <a>, </a>, <b>, </b>. Длина строки не превышает 2000 символов.

**Формат выходного файла (tags.out)**

Выведите в выходной файл минимальную по длине правильную xml-последовательность из тегов <a> и </a>, которую можно получить из заданной строки описанными операциями. Если решений несколько, выведите любое. Если из данной строки нельзя получить ни одной правильной xml-последовательности, выведите в выходной файл слово **Impossible**

**Examples**

|  |  |
| --- | --- |
| **Input** | **Output** |
| <b></a><a></a></a><a></b><a></a></b>  | <a><a></a><a></a></a><a></a><a></a>  |
| <b></b>  |   |
| <a></a></a>  | Impossible  |

**Заклинание**

**Описание**

Во вселенной Ядоснури магия очень хорошо изучена. Известно как составлять заклинания, как они должны звучать, и какая у них сила. Кстати, сила заклинания зависит от того кто и как его произносит, но еще она зависит от базовой силы вложенных в него слов.

Жительница этого мира – Сулико, учиться в школе и изучает магию. Она хорошо запомнила все доступные для её уровня N магических слов и M магических ядер. И даже может объяснить все шесть правил составления заклинаний. Они просты.

Правило «Предшествования» Одно магическое слово предшествует другому, если второе слово начинается с одной, двух или трех букв, которыми заканчивается первое слово. Например, слово «aarkh» предшествует слову «khroo».

Правило «Цепочек». Если из магических слов можно построить цепочку так, что каждое i-ое слово предшествует (i+1)-ому, то получается магическая фраза. При этом открывающим словом называется первое слово во фразе, а закрывающим – последнее. Количество слов в цепочке не меньше 1.

Правило «Определенности силы». Магическая сила слова равна количеству вхождений магических ядер в него.

Правило «Пустоты». Магическая фраза может быть заклинанием или пустой фразой. Если есть две магические фразы, с одинаковыми открывающим и закрывающим словами, то та фраза, в которой магическая сила слов больше, – пустая. Фраза является заклинанием, если не существует другой магической фразы с меньшей суммарной силой слов и такими же открывающим и закрывающим словами.

Правило «Ограниченности магии». Каждое слово может предшествовать не более чем 15 другим словам. Магическое ядро не может содержать в себе другие магические ядра. В заклинании не может быть использовано одно и то же магическое слово дважды.

Правило «Большое складывается из малого». Базовая сила заклинания определяется как сумма магических сил слов в него входящих. При этом для пустых магических фраз она, очевидно, равна нулю. Однако и сила некоторых заклинаний может оказаться равной нулю.

Вчера Сулико изучала заклинание «Порыв Ветра», но она не записала его и вспомнить тоже не может. Зато она помнит, что его базовая сила равна P.

В мире Ядоснури нет компьютеров, и Сулико просит Вас помочь ей восстановить заклинание «Порыв Ветра». **Требуется** написать программу, которая найдет все заклинания с базовой силой P, а какое из них искомое, Сулико будет разбираться сама.

**Формат входного файла (spell.in)**

Первая строка содержит три целых числа N, M и P (3≤N≤1500, 2≤M≤10, 1≤P≤100000) разделенные одиночными пробелами. В каждой из следующих N строк, содержится по одному магическому слову. В каждой из следующих M строк, содержится по одному магическому ядру. Каждое магическое слово и магическое ядро представляет собой строку, содержащую только маленькие буквы латинского алфавита. Длина любого магического слова больше 3, но меньше 51. Длина любого магического ядра больше 1, но меньше 21. Все слова между собой различны, также как и ядра между собой.

**Формат выходного файла (spell.out)**

В первой строке число найденных заклинаний с базовой силой равной P. В каждой следующей строке по одному заклинанию, представляющему собой последовательность магических слов, разделенных одиночными пробелами. Если количество ответов (заклинаний) превышает 100, то вывести надо 100 строк (заклинаний) лексикографически меньшие всех остальных.

**Examples**

|  |  |
| --- | --- |
| **Input** | **Output** |
| 4 2 9makeelephantthatattractiveea  | 1make elephant that attractive  |
| 5 3 3aarkhkhrookrohohrabohokrakhora  | 4aarkh hokraaarkh hrabokhroohokra aarkh  |

### Выдача кредита

**Описание**

За несколько лет один банк накопил кредитную статистику, содержащую информацию о 100 000 его клиентов (данные — настоящие!). Она хранится в файлах [input\_stat.txt и input\_task.txt](http://staff.mmcs.sfedu.ru/~sguda/credit.7z) в следующем формате: каждому клиенту соответствует одна строчка, в которой заданы отношение ежемесячной выплаты по кредиту к ежемесячному доходу клиента (debtRatio), ежемесячный доход клиента (salary, $), его возраст (age, лет) и факт грубой просрочки платежа по кредиту (trouble, 0 — все нормально, 1 — была серьезная просрочка). В файле input\_task.txt столбец trouble отсутствует.

**Требуется** составить программу, которая на основе статистики в input\_stat.txt, как можно точнее предсказывает, будет ли новый клиент платить проценты по кредиту вовремя. Для проверки качества предсказаний требуется применить программу к клиентам из файла input\_task.txt; и найти среди них 3470 клиентов, которые с наибольшей вероятностью просрочат платежи по кредиту.

На сайт нужно загрузить файл output.txt, содержащий 3470 номеров строк тех клиентов из input\_task.txt, которые с наибольшей вероятностью просрочат платеж. Именно столько клиентов из 50 000 в input\_task.txt на самом деле будут испытывать затруднения при выплате кредита. Нумерация строк в файле начинается с 1. Клиенту номер 1 соответствует номер строки 2, клиенту 2 - строки 3, ... В output.txt укажите каждый номер на отдельной строчке.

**Критерии оценки**

100 баллов, если среди найденных клиентов будет более 460 правильных, 40 баллов — более 420 правильных, 20 баллов — более 300 правильных.

**Компилятор**

**Описание**

Упрощенная модель процессора имеет 4 регистра R0,R1,R2,R3 для хранения чисел с плавающей точкой и подсоединена к оперативной памяти, содержащей данные для вычисления. Процессор поддерживает команды:

LOAD mem Ri - загрузка числа из оперативной памяти по адресу mem в регистр Ri
STORE Ri mem - сохранение регистра Ri в оперативную память по адресу mem
ADD Ri Rj Rk - суммирование чисел в регистрах Ri и Rj, запись результата в Rk
SUB Ri Rj Rk - вычитание: Rk := Ri - Rj
MUL Ri Rj Rk - умножение: Rk := Ri \* Rj
DIV Ri Rj Rk - деление: Rk := Ri / Rj
MOV Ri Rj - копирование содержимого регистра Ri в регистр Rj
SET Const Rj - помещение числа Const в регистр Rj

Все инструкции кроме LOAD и STORE выполняются за 1 такт работы процессора. LOAD и STORE выполняются за 20 тактов.

**Требуется** откомпилировать (транслировать в ассемблер) следующей фрагмент программы на паскале:

for i:=2 to N-1 do
         a[i] = (a[i-2] + 1) / (a[i-1] + 1) + (a[i-1] + 1) / (a[i-2] + 1) + 0.2\*(a[i-1] + a[i-2]);

так, чтобы она работала как можно меньшее число тактов. Число N явно задано на этапе компиляции. В начале работы программы все регистры содержат нулевые значения, массив a с элементами a[0] … a[N-1] хранится в ячейках оперативной памяти с адресами 0...N-1, переменная i может быть оптимизирована. Откомпилированная программа должна быть эквивалентна исходной, т.е. после ее исполнения ячейки памяти с адресами 0..N-1 должны содержать правильные данные. Содержимое других ячеек можно менять как угодно. Объем оперативной памяти: 10000 ячеек.

**Формат входного файла compiler.in**

Входной файл содержит единственное число N (3≤N≤100).

**Формат выходного файла compiler.out**

Выходной файл должен содержать инструкции процессору (каждая – на своей строчке).

**Отладка**

По ссылке http://staff.mmcs.sfedu.ru/~sguda/debugger.7z можно скачать «отладчик» для системы команд процессора.

debugger.exe N program.asm verbose
verbose = 1 – для пошагового вывода инструкций и содержимого регистров
verbose = 0 - статистика

**Examples**

|  |  |
| --- | --- |
| **Input** | **Output** |
| 3  | LOAD 0 R0LOAD 1 R1SET 1 R2ADD R0 R2 R0ADD R1 R2 R1DIV R0 R1 R2DIV R1 R0 R3LOAD 0 R0LOAD 1 R1ADD R0 R1 R0SET 0.2 R1MUL R0 R1 R0ADD R0 R2 R0ADD R0 R3 R0STORE R0 2  |