Содержание

Задача A. Добавление и удаление точек [0.6 sec, 256 mb]

Не все $N^2 \log N$ одинаковы полезны...

Какая-то лекция

У вас в каждый момент есть мультимножество A точек на плоскости.
Нужно научиться обрабатывать запросы трех типов:

- Добавить точку в мультимножество A
- Удалить точку из мультимножества A
- Вычислить $\sum\limits_{p \in A} \max\limits_{q \in A} distance(p, q)$.

Формат входных данных

Число запросов N ($1 \leqslant N \leqslant 3000$). Далее N строк, описывающие запросы, точный формат смотрите в примере. Координаты точек — целые число от 0 до 3000. Точки могут совпадать. Запрос *удалить точку* должен удалять ровно одну точку (гарантируется, что такая точка в мультимножестве на момент запроса есть).

Формат выходных данных

После каждой операции с множеством выводите текущую сумму максимальных расстояний. Абсолютная погрешность не должна превышать 10^{-6}.

Пример

adddel.in	adddel.out
6 + 0 0 + 5 5 + 5 0 - 5 5 - 5 0 - 0 0	0.00000000000000000000 14.14213562373095100000 19.14213562373095100000 10.00000000000000000000 0.00000000000000000000 0.00000000000000000000

Задача B. Самая дальняя [1.5 sec, 256 mb]

Даны N точек на плоскости, нужно уметь обрабатывать следующие запросы:

- `get a b` — возвращает максимум по всем точкам величины $ax + by$.
- `add x y` — добавить точку в множество.

Формат входных данных

Число N ($1 \leqslant N \leqslant 10^5$) и N точек. Далее число M ($1 \leqslant M \leqslant 10^5$) — количество запросов и собственно запросы. Формат запросов можно посмотреть в примере. Все координаты точек и числа `a`, `b` — целые числа, по модулю не превосходящие 10^9.

Формат выходных данных

На каждый запрос вида `get` выведите одно целое число — максимум величины $ax + by$.

Пример

mostfar.in	mostfar.out
3 0 0 1 0 0 1 10 get 1 1 get -1 -1 get 1 -1 get -1 1 add 2 2 add -2 -2 get 1 1 get -1 -1 get 1 -1 get -1 1	1 0 1 1 4 4 1 1

Задача C. Мосты в дереве [2 sec, 256 mb]

Уже декабрь, скоро... Скоро сессия!

Дед Мороз

Мальчик Серёжа почти закончил университет. Осталось только сдать последнюю сессию и защитить диплом. Сессия, прямо скажем, — это не проблема: Серёжа уже сдавал её в прошлом году, а вот диплом — задача непростая.

Серёжа пару лет назад уже выбрал тему диплома: это «Dynamic 2-Connectivity Problem». В такой задаче присутствует динамически меняющийся граф, и нужно быстро отвечать на запрос «количество мостов в графе в текущий момент времени».

Серёжа придумал несколько решений. В одном из них, наиболее простом для реализации, возникла одна несколько неприятная подзадача. Серёжа написал код, но тот оказался сложным и длинным...

И тут ему стало интересно, а вдруг есть более простая реализация? Более короткая и не менее быстрая. Для скорейшего получения ответа на столь любопытный вопрос решено было дать задачу на ближайший чемпионат родного университета.

До предзащиты осталось всего несколько месяцев, и Серёже придется реализовать ещё много забавных алгоритмов. Времени все меньше, а если он не успеет, то не защитится, и ему придется восстанавливаться на пятый курс. Опять.

Помогите, пожалуйста, Серёже в нелёгком деле написания диплома, и попробуйте сразиться с этой неприятной подзадачей.

Напомним, что *дерево* — неориентированный связный граф без циклов. *Мостом* называется ребро неориентированного графа, при удалении которого количество компонент связности этого графа увеличивается.

Дано дерево `YourTree` из N ($2 \leqslant N \leqslant 100\,000$) вершин. Ваша задача — подсчитывать количество мостов в графах, полученных добавлением к дереву `YourTree` наборов из K_i рёбер.

Формат входных данных

В первой строке ввода записаны целые числа N и M ($2 \leqslant N \leqslant 100\,000, 1 \leqslant M \leqslant 100\,000$) — количество вершин в дереве и количество запросов.

Во второй строке находится описание дерева `YourTree`: последовательность из $N-1$ целого числа p_2, p_3, ..., p_N, задающая рёбра $(2, p_2)$, $(3, p_3)$, ..., (N, p_N). Соседние числа разделены пробелами. Гарантируется, что $p_i < i$, а также что эти рёбра образуют дерево.

Далее идут описания M запросов, по одному на строке. Запрос с номером i описывается неотрицательным целым числом K_i и K_i парами чисел от 1 до N — рёбрами, которые добавляются к дереву при построении графа.

Сумма K_i по всем запросам не превосходит 100 000.

Заметим, что в полученных графах допустимы петли и кратные рёбра. Помните, что кратные рёбра не могут являться мостами.

Формат выходных данных

В ответ на каждый запрос выведите одно целое число — количество мостов в графе, полученном добавлением к дереву `YourTree` заданного набора рёбер.

Пример

bridges2.in	bridges2.out
7 8 1 1 2 2 3 3 1 4 5 3 4 5 6 7 3 2 1 5 6 1 1 1 1 3 6 2 4 3 2 7 1 5 1 3 1 2 1 3 1 6	4 0 2 6 5 2 4 3

Задача D. Distance Sum [2.5 sec, 256 mb]

На некоторой карте обозначены n городов и $n-1$ дорога, соединяющая эти города таким образом, что полученный граф является деревом. Города занумерованы последовательными целыми числами от 1 до n.

Город 1 является корнем дерева; обозначим для каждого $i > 1$ город, являющийся предком города i, за p_i, а расстояние между городами p_i и i за d_i.

Snuke хочет для каждого $1 \leqslant k \leqslant n$ вычислить наименьшую сумму расстояний от некоторого города до городов $1, \ldots, k$:

$$\min_{1 \leq v \leq n} \{\sum_{i=1}^{k} dist(i, v)\} \tag{1}$$

Здесь $dist(u, v)$ обозначает расстояние между городами u и v.

Формат входных данных

Первая строка входа содержит одно целое число n ($1 \leqslant n \leqslant 2 \cdot 10^5$). Далее идут $n-1$ строк, i-я из которых содержит два целых числа p_{i+1} и d_{i+1} — номер предка города $i+1$ и расстояние между городом $i+1$ и этим предком ($1 \leq p_i \leq n$, $1 \leq d_i \leq 2 \cdot 10^5$, p_i образуют дерево).

Формат выходных данных

Выведите n строк. В i-й из этих строк выведите ответ для $k = i$.

Примеры

tdsum.in	tdsum.out
10 4 1 1 1 3 1 3 1 5 1 6 1 6 1 8 1 4 1	0 3 3 4 5 7 10 13 16 19
15 1 3 12 5 5 2 12 1 7 5 5 1 6 1 12 1 11 1 12 4 1 1 5 5 10 4 1 2	0 3 9 13 14 21 22 29 31 37 41 41 47 56 59

Задача E. Грамматика [3 sec, 256 mb]

Формальной грамматикой называется способ описания формального языка, представляющий собой четвёрку $\Gamma = \langle \Sigma, N, S \in N, P \subset N^+ \times (\Sigma \cup N)^* \rangle$, где Σ — *алфавит*, элементы которого называют *терминалами*, N — множество, элементы которого называют *нетерминалами*, S — начальный нетерминал грамматики, P — набор *правил вывода* вида $\alpha \to \beta$.

Здесь N^+ — это все строки из одного или нескольких элементов множества N (непустые строки из нетерминалов), а $(\Sigma \cup N)^*$ — это все строки из нуля, одного или нескольких элементов множества $(\Sigma \cup N)$ (строки из терминалов и нетерминалов, включая пустую).

Грамматика называется *контекстно-свободной*, если в левой части каждого правила вывода всегда стоит только один нетерминал, то есть верно более сильное условие для правил: $P \subset N \times (\Sigma \cup N)^*$.

Для примера рассмотрим такую грамматику (второй тестовый случай примера) над алфавитом $\Sigma = \{\mathtt{a}, \mathtt{b}\}$, множеством нетерминалов $N = \{S, A\}$ и двумя правилами вывода:

1. $S \to \mathtt{b}A$
2. $A \to \mathtt{aa}$

Она, очевидно, является контекстно-свободной.

Чтобы получить язык, описываемый грамматикой, нужно взять строку, составленную из одного начального нетерминала S, и применить к ней один или несколько раз какие-либо правила вывода. Применение правила вывода состоит в нахождении в текущей строке в каком-либо месте строки из левой части этого правила и замены её на правую часть этого же правила вывода. *Языком* грамматики Γ будет называться множество всех строк, состоящих **только** из терминальных символов и выводимых по такому принципу.

Например, в языке описанной выше грамматики есть строка `baa`, выводимая следующим образом: $S \to \mathtt{b}A \to \mathtt{baa}$. Более того, других строк в её языке нет.

Но бывают и грамматики, в языке которых бесконечное количество строк. А бывают и такие, язык которых пуст.

Вам дана контекстно-свободная грамматика, алфавит в которой состоит из двух терминалов «`a`» и «`b`». Ваша задача — проверить, есть ли в её языке строка, в которой символ «`a`» встречается строго большее число раз, чем символ «`b`».

Нетерминалы в этой задаче будут нумероваться числами от 1 до n. Начальный нетерминал всегда имеет номер 1.

Формат входных данных

Во вводе заданы один или несколько тестовых случаев.

В первой строке каждого тестового случая записаны два числа n и m — число нетерминалов и количество правил вывода ($1 \leqslant n \leqslant 100$, $1 \leqslant m \leqslant 50\,000$).

Далее следуют m строк, каждая из которых описывает одно правило вывода в следующем виде. Сначала заданы A_i и k_i — номер нетерминала в левой части ($1 \leqslant A_i \leqslant n$) и число объектов в правой части правила вывода ($0 \leqslant k_i \leqslant 100$), далее следует k_i объектов, каждый из которых — либо нетерминал $B_{i,j}$ ($1 \leqslant B_{i,j} \leqslant n$), либо один из терминалов «`a`» или «`b`». Объекты разделяются одиночными пробелами.

Общая сумма значений n по всем тестовым случаям не превосходит 1000, а общая сумма значений m не превосходит 50 000. Размер ввода не превышает 5 мегабайт.

Ввод завершается строкой из двух нулей.

Формат выходных данных

Для каждого из тестовых случаев необходимо вывести на отдельной строке «YES», если в языке данной грамматики есть строка, в которой количество букв «a» строго больше, чем количество букв «b», и «NO» в противном случае.

Пример

grammar.in	grammar.out
2 2 1 2 a 2 2 1 b 2 2 1 2 b 2 2 2 a a 2 2 1 2 b 2 2 3 a a 1 0 0	NO YES NO