Содержание

Must have 2

Задача 9A. Самое дешевое ребро [0.5 sec, 512 mb] 2

Задача 9B. Тесты к задаче про два станка [0.2 sec, 512 mb] 3

Обязательные задачи 5

Задача 9C. Центроиды дерева [3.5 sec, 512 mb] 5

Задача 9D. БДБД [2.5 sec, 512 mb] 6

Дополнительные задачи 7

Задача 9E. Гамильтонов путь конём [1 sec, 512 mb] 7

Задача 9F. Строки в дереве [2.5 sec, 512 mb] 8

Обратите внимание, входные данные лежат в **стандартном потоке ввода** (он же stdin), вывести ответ нужно в **стандартный поток вывода** (он же stdout).

В некоторых задачах большой ввод и вывод. Пользуйтесь быстрым вводом-выводом.

В некоторых задачах нужен STL, который активно использует динамическую память (set-ы, map-ы) переопределение стандартного аллокатора ускорит вашу программу.

Обратите внимание на `GNU C++` компиляторы с суффиксом `inc`, они позволяют пользоваться дополнительной библиотекой. Под ними можно сдать вот это.

Must have

Задача 9A. Самое дешевое ребро [0.5 sec, 512 mb]

Дано подвешенное дерево с корнем в первой вершине. Все ребра имеют веса (стоимости). Вам нужно ответить на M запросов вида "найти у двух вершин минимум среди стоимостей ребер пути между ними".

Формат входных данных

В первой строке файла записано одно числ — n (количество вершин).

В следующих $n-1$ строках записаны два числа — x и y. Число x на строке i означает, что x — предок вершины i, y означает стоимость ребра. $x < i, |y| \leqslant 10^6$.

Далее m запросов вида (x, y) — найти минимум на пути из x в y $(x \neq y)$.

Ограничения: $2 \leqslant n \leqslant 5 \cdot 10^4, 0 \leqslant m \leqslant 5 \cdot 10^4$.

Формат выходных данных

m ответов на запросы.

Пример

stdin	stdout
5 1 2 1 3 2 5 3 2 2 2 3 4 5	2 2

Замечание

Мы изучаем центроиды! Эта задача здесь, чтобы перед чем-то более сложным вы потренировались. Сдавайте только центроиды. Пойманные на баловстве получат минус баллы.

Задача 9B. Тесты к задаче про два станка [0.2 sec, 512 mb]

Имеется множество из n работ и два станка. Время выполнения i-й работы на первом станке равно a_i, время выполнения i-й работы на втором станке равно b_i.

Каждую работу надо выполнить сначала на первом станке, потом на втором. На первом и на втором станках работы нужно выполнять в одинаковом порядке. Каждый станок в каждый момент времени может выполнять только одну работу.

Задача о двух станках – минимизировать время выполнения последней работы на втором станке.

Ваша задача – построить тесты к всевозможным жадностям.

Формат входных данных

В первой строке дано одно целое число n ($4 \leqslant n \leqslant 12$) — количество работ в тесте, который вам нужно построить.

Во второй строке L, R ($1 \leqslant L \leqslant 10$, $L < R \leqslant 10^9$) — ограничения на числа a_i, b_i ($L \leqslant a_i, b_i \leqslant R$).

В третьей строке написан тип жадности, против которой нужно построить тест.

Каждая жадность – запуск `stable_sort` с компаратором `less(i,j)`:

1 : `a[i] < a[j]`
2 : `b[i] < b[j]`
3 : `a[i] + b[i] < a[j] + b[j]`
4 : `a[i] - b[i] < a[j] - b[j]`
5 : `a[i] * b[i] < a[j] * b[j]`
6 : `(double)a[i] / b[i] < (double)a[j] / b[j]`
7 : `min(a[i], b[j]) < min(a[j], b[i])`
8 : `max(a[i], b[j]) < max(a[j], b[i])`
9 : `max(a[i], b[i]) < max(a[j], b[j])`

Далее написано число 0 или 1 — нужно ли развернуть массив после сортировки.

Гарантируется, что такой тест существует.

Формат выходных данных

В первой строке n.

Во второй строке массив a.

В третьей строк массив b.

Примеры

stdin	stdout
5 10 20 7 1	5 12 10 15 14 11 15 14 15 14 15
5 10 20 7 0	5 13 10 11 13 10 11 10 11 15 13

Замечание

В первом примере вам нужно построить тест против следующей жадности:

```
stable_sort(p.begin(), p.end(), [](int i, int j) {
     return max(a[i], b[j]) < max(a[j], b[i]);
});
reverse(p.begin(), p.end());
```

Подсказка по решению

Для $n = 12$ решение за $\mathcal{O}(n!)$ вряд ли зайдёт.

`stable_sort` $\neq$ `sort`, будьте внимательны!

При больших R тест не найдётся, потому что не возникнут крайние случаи $a_i = b_i, a_i = a_j$. Это проявляется, например, на 84-м тесте. Вообще всегда, когда вы стрессите, на больших n и a_i тесты **НЕ** находятся, ищите тесты на маленьких n и a_i.

Обязательные задачи

Задача 9C. Центроиды дерева [3.5 sec, 512 mb]

Дано дерево из n вершин. У каждой вершины есть цвет. Нужно обработать q запросов (v_i, c_i): найти расстояние от v_i до ближайшей к v_i вершины цвета c_i. Расстоянием между вершинами называется минимальное количество рёбер в пути между ними.

Формат входных данных

На первой строке число n ($1 \leqslant n \leqslant 10^5$), следующая строка содержит числа $p_1, p_2, \ldots, p_{n-1}$. $0 \leqslant p_i < i$. p_i – отец вершины i в дереве. Далее строка с числами $a_0, a_1, \ldots, a_{n-1}$. $0 \leqslant a_i < n$. a_i – цвет вершины i. Далее строка с числом q ($1 \leqslant q \leqslant 10^5$). Следующие q строк содержат запросы $v_i q_i$ ($0 \leqslant v_i < n$, $0 \leqslant c_i < n$).

Формат выходных данных

Для каждого запроса выведите одно число – расстояние до ближайшей вершины нужного цвета, или -1, если в дереве нет вершин такого цвета.

Примеры

stdin	stdout
5 0 1 1 3 1 2 3 2 1 9 0 1 0 2 0 3 1 0 2 1 2 2 3 3 3 1 4 2	0 1 2 -1 2 1 2 1 1

Задача 9D. БДБД [2.5 sec, 512 mb]

Большая Древесная База Данных создана для того, чтобы в ней можно было надежно сохранить и раскрасить любое дерево. В новой версии БДБД запланирован новый функционал, для реализации которого потребуется вновь переосмыслить теорию графов.

В БДБД хранится взвешенное дерево. В языке запросов Системы Управления Большой Древесной Базы Данных (СУБДБД) предусмотрены два вида запросов:

1. «1 v d c» — покрасить все вершины, находящиеся на расстоянии не более d от вершины v, в цвет c. Все вершины изначально окрашены в цвет с номером 0.
2. «2 v» — вывести цвет вершины v.

Запрограммируйте работу СУБДБД и ответить на все запросы пользователя.

Формат входных данных

В первой строке число N ($1 \leqslant N \leqslant 10^5$) — количество вершин дерева.

Следующие $N-1$ строк содержат тройки чисел a_i, b_i, w_i ($1 \leqslant a_i, b_i \leqslant N$, $a_i \neq b_i$, $1 \leqslant w_i \leqslant 10^4$), i-я строка обозначает ребро веса w_i между вершинами a_i и b_i.

В следующей строке число запросов Q ($1 \leqslant Q \leqslant 10^5$).

В каждой из Q следующих строк запросы одного из двух видов:

1. Числа 1, v, d, c ($1 \leqslant v \leqslant N$, $0 \leqslant d \leqslant 10^9$, $0 \leqslant c \leqslant 10^9$).
2. Числа 2, v ($1 \leqslant v \leqslant N$).

Все числа во входных данных целые.

Формат выходных данных

Для каждого запроса второго типа необходимо вывести в отдельной строке цвет запрошенной вершины.

Примеры

stdin	stdout
5 1 2 30 1 3 50 3 4 70 3 5 60 8 1 3 72 6 2 5 1 4 60 5 2 3 2 2 1 2 144 7 2 4 2 5	6 6 0 5 7

Дополнительные задачи

Задача 9E. Гамильтонов путь конём [1 sec, 512 mb]

Дана доска размера $n \times n$ ($5 \leqslant n \leqslant 400$).
Известно, что такую доску можно обойти конём, посетив каждую клетку ровно один раз.

Формат входных данных

Число n.

Формат выходных данных

Выведите гамильтонов **путь** – $n \times n$ строк, каждая содержит очередную клетку.
Координаты нумеруются с нуля.
Путь можно строить из любой клетки. Если путей несколько, выведите любой.

Примеры

stdin	stdout
5	4 0 4 4 0 4 0 0 4 2 1 3 2 0 3 3 0 2 3 1 2 4 1 1 2 2	3 2 2 3 1 2 2 1 3 4 0 1 4 1 1 4 1 0 4 3 0 3 3 0

Замечание

Начинайте со случайной точки, и всё получится.

Задача 9F. Строки в дереве [2.5 sec, 512 mb]

Дано дерево. Дерево — это связный граф без циклов. На каждом ребре дерева написана строчная латинская буква. Между каждыми двумя вершинами существует ровно один простой путь, то есть путь по рёбрам дерева, проходящий через каждую вершину не более одного раза. Каждому пути соответствует строка, которая получается, если идти по этому пути и читать буквы на рёбрах в порядке следования. Путь можно проходить, начиная с любого его конца.

Также дана строка S. Соответствует ли она какому-либо простому пути в данном дереве?

Длина строки и размер дерева не превышают 300 000.

Вам необходимо реализовать функцию

`Solve (str, len, tree, n, v, u)`.

- `str` — строка, которую нужно найти
- `len` — длина строки
- `tree` — массив рёбер (`edge`)
- `n` — количество вершин в дереве
- `v`, `u` — переменные, в которые следует записать концы найденного вами пути

Функция должна возвращать 0 или 1 — присутствует ли строка в дереве.

Примеры

stdin	stdout
abc 4 1 2 c 4 3 a 2 4 b	YES 1 3
abc 1	NO
zy 6 1 2 x 1 3 y 1 4 z 3 5 a 4 6 b	YES 3 4

Пояснения к примерам

В первом примере строка `abc` соответствует пути 3–4–2–1.

Во втором примере в дереве нет ни одного ребра.

В третьем примере строка `zy` соответствует пути 3–1–4.