Зміст

Задача 0.0:1. «ads001»

Напишіть програму, яка прочитає кілька рядків, що містять по два числа кожен, і виведе для кожної пари чисел їхню суму.

Вхідні дані.

Стандартний вхідний потік містить кілька (скільки саме — наперед невідомо, але не менше 1 й не більше 1000) рядків, кожен рядок містить по два числа, розділених пробілом. Значення самих чисел не перевищують за модулем мільярд (інакше кажучи, належать проміжку $[-10^9; +10^9]$).

Результати.

У стандартний вихідний потік потрібно вивести суму кожної пари чисел (кожну суму в окремому рядку).

Задача 0.0:2. «ads002»

Орієнтований зважений граф заданий переліком дуг (орієнтованих ребер). Відсортувати ці дуги за зростанням довжин, зберігши (додатковими полями) номери цих дуг у вхідних даних.

Вхідні дані.

Перший рядок містить кількість вершин N ($2 \leqslant N \leqslant 30000$) і кількість дуг (орієнтованих ребер) M ($1 \leqslant M \leqslant 123456$). Кожен із подальших M рядків містить рівно три цілих числа u, v та len — початок, кінець та довжину дуги. $1 \leqslant u, v \leqslant N$, $u \neq v$, $1 \leqslant len \leqslant 10^9$. Гарантовано, що довжини всіх дуг різні.

Результати.

Результат має містити M рядків по чотири цілих числа u, v, len, idx у кожному — початок, кінець, довжину дуги, та її номер у вхідних даних (нумерація з одиниці). При цьому дуги мають бути відсортовані за зростанням довжин.

Примітка.

Задачу рекомендується розв'язати шляхом використання стандартного сортування та задання власного компаратора.

Задача 0.0:3. «ads003»

Інформація про кожного з учнів школи має вигляд: прізвище, ім'я, клас, дата народження.

В залежності від ситуації, їх буває потрібно сортувати у різних порядках. Повний перелік стандартних порядків такий:

- `surname` — за прізвищем, ігноруючи ім'я;
- `fullname` — за прізвищем, а при однаковості прізвищ – за іменем;
- `birthyear` — за роком народження, ігноруючи дату всередині року, від менших дат (старших учнів) до більших дат (молодших учнів);
- `birthdate` — за датою народження, включаючи рік, від менших дат (старших учнів) до більших дат (молодших учнів);
- `birthday` — за днем народження, тобто вважаючи рівними однакові день і місяць різних років, від 01.01 до 31.12 (у календарному смислі, в форматі `дд.мм`);
- `grade` — за класом як номером року навчання, від 1 до 11, ігноруючи букву;
- `class` — спочатку за класом як номером року навчання, а при однаковості – за буквою класу від `A` до `Z`.

Причому, інколи буває потрібно застосовувати декілька порядків: спочатку один, при рівності за першим — другий, і т. д. Наприклад, "`birthyear grade surname`" означає, що спочатку треба вивести всіх школярів, наприклад, 1998 р. н., потім усіх 1999 р. н., потім усіх 2000 р. н., і так далі, причому серед школярів одного й того ж року народження проводити сортування за класами (як роками навчання, ігноруючи букву), а вже тих, у кого однакові і рік народження, і клас без урахування букви, розташувати в алфавітному порядку за прізвищами.

Вхідні дані. У першому рядку записаний необхідний порядок сортування, як одне або декілька зі слів `surname`, `fullname`, `birthyear`, `birthdate`, `birthday`, `grade`, `class`. Хоча б одне слово обов'язково присутнє; якщо слів декілька, то вони розділені одинарними пробілами; одне й те ж слово не може згадуватися більше одного разу.

У другому рядку записана кількість учнів N ($2 \leqslant N \leqslant 12345$).

Далі йдуть N груп по 4 рядки кожна: (1) прізвище — починається з великої латинської літери, далі послідовність латинських букв, дефісів, апострофів; (2) ім'я — починається з великої латинської літери, далі послідовність латинських букв, дефісів, апострофів і пробілів; (3) дата народження (`дд.мм.рр`, тобто і день, і місяць, і рік завжди задані рівно двома цифрами); (4) клас — число від 1 до 11 і одна велика латинська літера, без пробілу між ними.

Всі дати народження правильні (не буває, наприклад, 30.02) і належать діапазону років від 1990 до 2030. Тобто, наприклад, $97 < 02$, бо насправді це 1997 і 2002. Здебільшого, учні старших класів мають раніші дати народження, ніж учні молодших, але можливі виключення.

Більшість (не менше 99,9%) прізвищ та імен мають довжину до 20 символів, але щодо решти 0,1% гарантовано лише те, що їх сумарна довжина не перевищує мільйон символів. Прізвища та імена гарантовано закінчуються або літерою, або апострофом (але не дефісом і не пробілом).

Прізвища з прізвищами та імена з іменами слід порівнювати стандартним для мов програмування регістрозалежним лексикографічним порівнянням рядків (навіть у випадку наявності дефісів, пробілів, тощо).

Результати.

Вивести N рядків, у кожному з яких записані дані про одного з учнів, у форматі: клас, кома, пробіл, прізвище, кома, пробіл, ім'я, кома, пробіл, дата народження.

Клас, прізвище, ім'я і дата народження мають бути байт-у-байт такими ж, як у вхідних даних.

Рядки мають бути впорядкованими у відповідності до описаних в першому рядку вхідних даних порядком сортування.

У всіх випадках, коли застосування вказаних порядків до вказаним даних залишає можливість різних правильних відповідей — виводьте будь-яку одну з правильних відповідей. Наприклад, якщо при вже згаданому порядку “`birthyear grade surname`” в деякому класі (році навчання) є однопрізвищники одного року народження — їх можна вивести в будь-якому порядку, але треба вивести кожного рівно один раз.

Приклади:

Клавіатура (stdin)
birthyear grade surname 8 Gonsales Anna Maria 5A 01.05.03 Ivanov Kyrylo 11A 12.01.97 Ivanov Ivan 3D 20.01.05 Ivanov Andrii 11B 23.12.97 Petrenko Nataliia 11B 23.08.97 Andrijchenko Kateryna 11B 07.02.98 Shevchenko Denys 10A 07.05.98 Smirnov-Kovalenko Yehor 9A 13.11.98
Екран (stdout)
11A, Ivanov, Kyrylo, 12.01.97 11B, Ivanov, Andrii, 23.12.97 11B, Petrenko, Nataliia, 23.08.97 9A, Smirnov-Kovalenko, Yehor, 13.11.98 10A, Shevchenko, Denys, 07.05.98 11B, Andrijchenko, Kateryna, 07.02.98 5A, Gonsales, Anna Maria, 01.05.03 3D, Ivanov, Ivan, 20.01.05

Примітка.

1. Відповідь, у якій `Ivanov Andrii` та `Ivanov Kyrylo` переставлені місцями, також є правильною. Інших, крім цих двох, правильних відповідей для цих вхідних даних не існує.

2. Мова C# дотримується трохи іншого, чим більшість мов програмування, уявлення про стандартне регістрозалежне лексикографічне порівняння рядків, тому нею замість очевидного фрагмента `p1.surName.CompareTo(p2.surName)` слід писати `String.Compare(p1.surName, p2.surName, StringComparison.Ordinal)`, та аналогічно в решті випадків, де це важливо.

Задача 0.0:4. «ads004»

Напишіть програму, яка виконуватиме послідовність запитів виду `ADD` *num*, `PRESENT` *num* та `COUNT` (без параметра). Програму обов'язково слід писати за допомогою бібліотечного типу (колекції) set (її реалізації в конкретних мовах програмування можуть називатися `HashSet`, `TreeSet`, `SortedSet`, ...).

Виконання кожного запиту виду `ADD` *num* має додавати елемент *num* у множину (якщо такий елемент вже є, додавання ще однієї копії не змінює множину), на екран при цьому нічого не виводиться.

При виконанні кожного запиту виду `PRESENT` *num* має видаватися повідомлення `YES` або `NO` (великими літерами, в окремому рядку), відповідно до того, чи є такий елемент у множині; значення множини при цьому не змінюється.

При виконанні кожного запиту виду `COUNT` має видаватися на екран в окремому рядку кількість різних елементів у множині; значення множини при цьому не змінюється.

Вхідні дані.

У першому рядку задано кількість запитів N ($1 \leqslant N \leqslant 10^5$), далі йдуть N рядків, кожен з яких містить по одному запиту згідно з описаним форматом.

Значення чисел є цілими і не перевищують за модулем 100 000 000 (інакше кажучи, належать проміжку $[-10^8; +10^8]$).

Результати.

Виводьте окремими рядками результати запитів `PRESENT` та `COUNT`; на запити `ADD` нічого не виводьте.

Приклади:

Клавіатура (stdin)	Екран (stdout)
7 ADD 5 ADD 7 COUNT PRESENT 3 PRESENT 5 ADD 3 COUNT	2 NO YES 3

Задача 0.0:5. «ads005»

порожній, це так треба

Задача 0.0:6. «ads006»

Попередження. Ця задача може досить легко розв'язуватися одними мовами програмування, і значно важче — іншими. Автору задачі твердо відомо, що її легко розв'язати мовою `C++` з використанням STL-контейнера `set` чи мовою `Java` з використанням колекції `TreeSet`. Наскільки відомо автору задачі, її не можна легко розв'язати ні мовою `Python`, ні мовою `C#`, бо відповідні типи чи колекції (`Python set`, `C# HashSet`, `C# SortedSet`, ...) тупо не містять потрібних для цієї задачі методів, які є в `C++ STL std::set` та `Java TreeSet`. Звісно, ніщо не заважає написати повністю свій аналог цих класів своєю мовою програмування, але це істотно складніше, чим використати готові методи готового бібліотечного типу. Чи є інші, крім перелічених, мови програмування, де цю задачу все-таки можна розв'язати легко, автор задачі не знає.

Кінець попередження.

Напишіть програму, яка виконуватиме послідовність запитів виду `ADD` *num*, `PRESENT` *num* та `COUNT` (без параметра). Програму обов'язково слід писати за допомогою бібліотечного типу (колекції) set (її реалізації в конкретних мовах програмування можуть називатися `HashSet`, `TreeSet`, `SortedSet`, ...).

Виконання кожного запиту виду `ADD` *num* має додавати елемент *num* у множину (якщо такий елемент вже є, додавання ще однієї копії не змінює множину), на екран при цьому нічого не виводиться.

При виконанні кожного запиту виду `PRESENT` *num* має видаватися повідомлення `YES` (великими літерами, в окремому рядку), якщо таке число наявне у множині (як елемент). Якщо ж такого елемента у множині немає, то в окремому рядку має бути виведено спочатку слово `NO` (великими літерами), кома і один пробіл, а потім або слово `EMPTY` (великими літерами), яке позначає, що множина порожня, або нерівність, яка показує найближчі елементи, що знаходяться у множині. Якщо множина містить і менший, і більший елементи, нерівність має бути подвійною, а якщо елемент із поточного запиту менше мінімального або більше максимального — одинарною. Значення множини при виконанні запиту `PRESENT` не змінюється.

При виконанні кожного запиту виду `COUNT` має видаватися на екран в окремому рядку кількість різних елементів у множині; значення множини при цьому не змінюється.

Вхідні дані.

У першому рядку задано кількість запитів N ($1 < N < 500\,000$), далі йдуть N рядків, кожен з яких містить по одному запиту згідно з описаним форматом.

Значення чисел є цілими й належать діапазону знакового 32-бітового цілого типу (в більшості мов програмування, це тип `int`).

Результати.

Виводьте окремими рядками результати запитів `PRESENT` та `COUNT`; на запити `ADD` нічого не виводьте.

Зверніть увагу, що у випадку виведення символів < довкола них не можна ставити пробіли.

Приклади:

Клавіатура (stdin)	Екран (stdout)
12 COUNT PRESENT 3 ADD 2 ADD 5 PRESENT 3 PRESENT 1 PRESENT 17 COUNT ADD 3 ADD 5 PRESENT 3 COUNT	0 NO, EMPTY NO, 2<3<5 NO, 1<2 NO, 5<17 2 YES 3

Попередження. Ця задача може досить легко розв'язуватися одними мовами програмування, і значно важче — іншими. Автору задачі твердо відомо, що її легко розв'язати мовою `C++` з використанням STL-контейнера `set` чи мовою `Java` з використанням колекції `TreeSet`. Наскільки відомо автору задачі, її не можна легко розв'язати ні мовою `Python`, ні мовою `C#`, бо відповідні типи чи колекції (`Python set`, `C# HashSet`, `C# SortedSet`, ...) тупо не містять потрібних для цієї задачі методів, які є в `C++ STL std::set` та `Java TreeSet`. Звісно, ніщо не заважає написати повністю свій аналог цих класів своєю мовою програмування, але це істотно складніше, чим використати готові методи готового бібліотечного типу. Чи є інші, крім перелічених, мови програмування, де цю задачу все-таки можна розв'язати легко, автор задачі не знає.

Кінець попередження.

Задача 0.0:7. «ads007»

Карацуба

Приклади:

Клавіатура (stdin)	Екран (stdout)
2 2 0 12345 123456789101112131415161 7 987654321	4 0 121932631212499102497704297918705 7

Задача 0.0:8. «ads008»

У банка є клієнти. Кожен клієнт має рівно один рахунок.

Напишіть програму, яка виконуватиме послідовність запитів таких двох видів:

1. починається з числа 1, потім через пробіл іде ім'я клієнта (слово з латинських букв), далі через пробіл іде сума грошей, яка додається до рахунку поточного клієнта (ціле число, не перевищує за модулем $10\,000$).
2. починається з числа 2, через пробіл іде ім'я клієнта. На кожен такий запит програма повинна відповісти? яка сума в даний момент є на рахунку заданого клієнта. Якщо таке ім'я клієнта поки що ні разу не згадувалося в запитах виду 1, виводьте замість числа слово `ERROR`.

На початку роботи програми у всіх клієнтів на рахунку 0. Потім суми можуть ставати як додатними, так і від'ємними.

Зверніть увагу, що у ситуації, коли клієнт зняв грошей сумарно рівно стільки ж, скільки поклав, сума на рахунку стає рівною 0; але, раз його ім'я вже зустрічалося, нульове значення **не** є підставою виводити `ERROR`.

Вхідні дані. Перший рядок вхідних даних — кількість запитів N (обмеження на максимальне значення N — приблизно до $10^5 = 100\,000$).

Далі йдуть N рядків, у кожному з яких задано один запит одного з двох вищеописаних видів.

Результати. Формат виведення результатів. На кожний запит 2-го виду потрібно вивести поточне значення на рахунку заданого клієнта (або слово `ERROR`).

Приклади:

Клавіатура (stdin)	Екран (stdout)
7 1 asdf 3 1 zxcv 5 2 asdf 1 asdf -2 2 asdf 2 lalala 2 zxcv	3 1 ERROR 5

Примітка. Для абсолютно повного виконання задачі, її рекомендується зробити двома різними способами.

Один — з використанням бібліотечної словникової структури даних. Відповідна готова колекція чи контейнер може називатися (залежно від мови програмування) `Dictionary`, `dict`, `map`, `HashMap`, `SortedMap`, тощо.

Інший — із використанням власноруч написаних hash-таблиць (не використати біблотечні, а написати спрощений їх аналог вручну, користуючись лише масивами та/або списками).

Задача 0.0:9. «ads009»

Знайдіть усі способи подати вказане натуральне N як суму двох квадратів (тобто, $N = x^2 + y^2$, де x, y — цілі додатні).

Вхідні дані. Єдиний рядок містить єдине число N ($1 \leqslant N \leqslant 10^{13}$).

Результати. Виведіть всі пари цілих додатних x, y, таких, що $x^2 + y^2 = N$. Пари, де x та y лише переставлені місцями, слід вважати різними (при $x \neq y$) й виводити кожну окремо. **Перелік пар обов'язково повинен бути впорядкованим за зростанням** x. Якщо вказане N неможливо подати як $x^2 + y^2$, слід просто нічого не виводити.

Приклади:

Клавіатура (stdin)	Екран (stdout)
16	
10	1 3 3 1
4225	16 63 25 60 33 56 39 52 52 39 56 33 60 25 63 16

Примітка.

1. Просять вивести лише розкладення в суму квадратів цілих додатних, тому в першому тесті нема ні $0^2 + 4^2$, ні $4^2 + 0^2$. З тієї ж причини, у третьому тесті нема ні $0^2 + 65^2$, ні $65^2 + 0^2$.
2. У першому тесті, слід не виводити взагалі нічого, в тому числі ні пробіла, ні символа переведення рядка.

Задача 0.0:10. «ads010»

Напишіть програму, яка знаходитиме кількість натуральних чисел із проміжку $[a; b]$, які задовольняють одночасно двом таким вимогам:

1. число є точним квадратом, тобто корінь з нього цілий (наприклад, точними квадратами є $1 = 1^2$, $9 = 3^2$, $1024 = 32^2$; а 8, 17, 1000 не є точними квадратами);
2. сума цифр цього числа кратна K (Наприклад, сума цифр числа 16 рівна $1 + 6 = 7$).

Програма повинна прочитати три числа в одному рядку $a\,b\,K$ і вивести одне число — кількість чисел, які задовольняють умовам.

Обмеження: $1 \leqslant a \leqslant b \leqslant 2 \cdot 10^9$, $2 \leqslant K \leqslant 42$.

Приклади: Input

Клавіатура (stdin)	Екран (stdout)
7 222 9	4

Примітка. Цими чотирма числами є 9, 36, 81, 144.

Задача 0.0:11. «ads011»

Знайдіть будь-який один спосіб подати вказане натуральне N як суму $a^4 + b^4 + c^4 + d^4 + e^4$ (де a, b, c, d, e — цілі невід'ємні), або з'ясуйте, що це неможливо.

Вхідні дані. Єдиний рядок містить єдине число N ($1 \leqslant N \leqslant 10^9$).

Результати. Якщо подати вказане число потрібним чином можливо, то виведіть в один рядок через пропуски такі п'ять цілих невід'ємних чисел a, b, c, d, e, що $a^4 + b^4 + c^4 + d^4 + e^4 = N$.

Якщо неможливо, виведіть замість відповіді $-1\ -1\ -1\ -1\ -1$ (п'ять разів мінус одиницю).

Приклади:

Клавіатура (stdin)	Екран (stdout)
1	0 0 0 0 1
17	0 0 0 1 2
9	-1 -1 -1 -1 -1

Примітка.

Якщо цілі невід'ємні a, b, c, d, e, такі, що $a^4 + b^4 + c^4 + d^4 + e^4 = N$, існують, то слід вивести будь-який один такий комплект у будь-якому порядку — перевірка відбуватиметься за смислом, а не зіставленням зі зразком.

Ця задача далеко не найскладніша з усіх, але все ж її складність (важкість написання) вища середньої поміж усіх задач комплекту. Так, при написанні цієї задачі варто врахувати досвід задачі $x^2 + y^2 = N$. Але все це не заважає цій задачі бути значно складнішою.

Задача 0.0:12. «ads012»

Напишіть програму, яка знаходитиме перелік прямокутних паралелепіпедів, що мають об'єм V та площу поверхні S. Враховувати лише паралелепіпеди, в яких всі три розміри виражаються натуральними числами.

Вхідні дані. Єдиний рядок містить розділені одинарним пробілом два натуральні числа V S, обидва у межах від 1 до 10^6 (мільйон).

Результати. Кожен рядок повинен містити розділені пробілами цілочисельні розміри $a\ b\ c$ чергового паралелепіпеда. Ці трійки обов'язково виводити у порядку зростання першого розміру a, при рівних a — у порядку зростання другого розміру b (а різних відповідей, в яких рівні і a, і b, не буває).

У випадку, якщо жодного паралелепіпеда з потрібними V, S не існує, виводьте єдиний рядок `0 0 0`.

Примітка. Якщо прямокутний паралелепіпед має розміри a b c, то його об'єм дорівнює $a \cdot b \cdot c$, а площа поверхні $2 \cdot \big(a \cdot b + b \cdot c + c \cdot a\big)$, бо є дві грані (наприклад, передня й задня) площі $a \cdot b$, дві (наприклад, ліва та права) площі $b \cdot c$ та дві (наприклад, верхня й нижня) площі $c \cdot a$.

Гарантовано, що у кожному з тестів, що використовуються для оцінювання цієї задачі, кількість трійок-відповідей строго менша 100.

Приклади:

Клавіатура (stdin)	Екран (stdout)
7 7	0 0 0
6 22	1 2 3 1 3 2 2 1 3 2 3 1 3 1 2 3 2 1
1 6	1 1 1
1600 1120	4 20 20 5 8 40 5 40 8 8 5 40 8 40 5 20 4 20 20 20 4 40 5 8 40 8 5

Задача 0.0:13. «ads013»

Ця задача відрізняється від попередньої лише обмеженнями на V, S.

Напишіть програму, яка знаходитиме перелік прямокутних паралелепіпедів, що мають об'єм V та площу поверхні S. Враховувати лише паралелепіпеди, в яких всі три розміри виражаються натуральними числами.

Вхідні дані. Єдиний рядок містить розділені одинарним пробілом два натуральні числа V S, обидва у межах від 10^6 (мільйон) до 10^{18}.

Результати. Кожен рядок повинен містити розділені пробілами цілочисельні розміри a b c чергового паралелепіпеда. Ці трійки обов'язково виводити у порядку зростання першого розміру a, при рівних a — у порядку зростання другого розміру b (а різних відповідей, в яких рівні і a, і b, не буває).

У випадку, якщо жодного паралелепіпеда з потрібними V, S не існує, виводьте єдиний рядок `0 0 0`.

Примітка. Якщо прямокутний паралелепіпед має розміри a b c, то його об'єм дорівнює $a \cdot b \cdot c$, а площа поверхні $2 \cdot (a \cdot b + b \cdot c + c \cdot a)$, бо є дві грані (наприклад, передня й задня) площі $a \cdot b$, дві (наприклад, ліва та права) площі $b \cdot c$ та дві (наприклад, верхня й нижня) площі $c \cdot a$.

Гарантовано, що у кожному з тестів, що використовуються для оцінювання цієї задачі, кількість трійок-відповідей строго менша 100.

Приклади:

Клавіатура (stdin)	Екран (stdout)
54756000 1240200	120 585 780 120 780 585 130 360 1170 130 1170 360 180 195 1560 180 1560 195 195 180 1560 195 1560 180 360 130 1170 360 1170 130 585 120 780 585 780 120 780 120 585 780 585 120 1170 130 360 1170 360 130 1560 180 195 1560 195 180

Примітка. Легші для сприйняття людиною приклади див. у задачі «Паралелепіпеди (середні обмеження)»

Задача 0.0:14. «ads014»

Є прямокутна таблиця розміром N рядків на M стовпчиків. У кожній клітинці записане невід'ємне ціле число. По ній потрібно пройти згори донизу, починаючи з будь-якої клітинки верхнього рядка, далі переходячи щоразу в одну з «нижньо-сусідніх» і закінчити маршрут у якійнебудь клітинці нижнього рядка. «Нижньо-сусідня» означає, що з клітинки (i, j) можна перейти в $(i+1, j-1)$, або в $(i+1, j)$, або в $(i+1, j+1)$, але не виходячи за межі таблиці (у крайньому лівому стовпчику перший з наведених варіантів стає неможливим, а у крайньому правому — останній).

Напишіть програму, яка знаходитиме максимально можливу *щасливу* суму значень пройдених клітинок серед усіх допустимих шляхів. Як широко відомо у вузьких колах, щасливими є ті й тільки ті ч*и*сла, десятковий запис яких містить лише цифри 4 та/або 7 (можна обидві, можна лише якусь одну; але ніяких інших цифр використовувати не можна). Зверніть увагу, що щасливою повинна бути с*а*ме сума, а обмежень щодо окремих доданків нема.

Вхідні дані. У першому рядку записані N та M — кількість рядків і кількість стовпчиків ($1 \leqslant N, M \leqslant 12$); далі у кожному з наступних N рядків записано рівно по M розділених пробілами невід'ємних цілих чисел, кожне не більш ніж з 12 десяткових цифр — значення клітинок.

Результати. Вивести або єдине ціле число (знайдену максимальну серед щасливих сум за маршрутами зазначеного вигляду), або рядок "`impossible`" (без лапок, маленькими латинськими буквами). Рядок "`impossible`" слід виводити тільки у разі, коли жоден з допустимих маршрутів не має щасливої суми.

Клавіатура (stdin)	Екран (stdout)
3 4 3 0 10 10 5 0 7 4 4 10 5 4	7

Примітка. Взагалі-то максимально можливою сумою є $27 = 10 + 7 + 10$, але число 27 не є щасливим. Тому відповіддю буде максимальна серед щасливих сума $7 = 3 + 0 + 4$, яка досягається уздовж маршруту `a[1][1]`→`a[2][2]`→`a[3][1]`.

Наскільки відомо автору задачі, автором «широко відомого у вузьких колах» такого трактування «щасливого числа» є Василь Білецький, випускник Львівського національного університету імені Івана Франка, котрий був капітаном першої з українських команд, що вибороли золоту медаль на фіналі першості світу ACM ICPC, і тривалий час входив у десятку найсильніших спортивних програмістів світу за рейтингом TopCoder.

Задача 0.0:15. «ads015»

порожній, це так треба

Задача 0.0:16. «ads016»

Нагадаємо, що розміщення з n по k — виборки по k елементів з n можливих, причому порядок елементів у вибірці важливий («1 2» відрізняється від «2 1»).

Наприклад, повний перелік всіх можливих розміщень з 3 по 2: (1,2), (1,3), (2,1), (2,3), (3,1), (3,2).

Напишіть програму, яка за заданими n і k генеруватиме всі можливі розміщення з чисел 1, 2, ..., n по k. Розміщення повинні виводитися в лексикографічному (словниковому, але за числами) порядку.

Вхідні дані. В єдиному рядку через пропуск задано два числа — спочатку n, потім k.

Виконуються умови: $k \leqslant n \leqslant 20$; $1 \leqslant k \leqslant 7$; $1 \leqslant A_n^k < 10^4$ (де A_n^k — кількість розміщень, які слід згенерувати).

Результати. Виведіть знайдені розміщення, кожне в окремому рядку, в лексикографічному (за числами) порядку. Усередині рядка числа повинні відділятися один від одного одиничними пробілами.

Приклади:

Клавіатура (stdin)	Екран (stdout)
3 2	1 2 1 3 2 1 2 3 3 1 3 2

Примітка.

Під «лексикографічним (за числами) порядком» мається на увазі: спочатку треба виводити всі перестановки, де на першому (крайньому зліва) місці 1, потім усі, де на першому місці 2, тощо, насамкінець усі, де на першому місці n. У свою чергу, всі перестановки, де перші числа однакові між собою, мають бути відсортовані за другими числами; всі перестановки, де однакові як перші так і другі числа, мають бути відсортовані за третіми; тощо.

Примітка «за числами» означає, що якби спочатку сформували всі перестановки як рядки, і відсортували рядки, то порядок виявився б таким самим при $n \leqslant 9$, але при $n \geqslant 10$ з'являється відмінність. Наприклад, при $n = 10$ у цій задачі потрібно виводити перестановки, що починаються з 10, наприкінці, а якби поформували рядки й відсортували їх як рядки, то перестановки, що починаються з 10, потрапили б між перестановками, що починаються з 1, і перестановками, що починаються з 2 (що в цій задачі не буде зараховуватися).

Задача 0.0:17. «ads017»

Для вказаних чисел N і K виведіть всі зростаючі послідовності довжини K з чисел $1..N$ у лексикографічному порядку.

Вхідні дані. В одному рядку через пробіл задані 2 числа: спочатку N, потім K. Гарантовано, що $1 \leqslant K \leqslant N \leqslant 100$. Також гарантовано, що подаватимуться лише такі вхідні дані, для яких відповідь не порожня і не перевищує 2 мегабайти.

Результати. Необхідно вивести всі зростаючі послідовності довжини K з чисел $1..N$ у лексикографічному (за числами) порядку. Послідовності виводяться по одній в рядку, числа всередині послідовностей розділяються пробілами.

Приклади:

Клавіатура (stdin)	Екран (stdout)
5 2	1 2 1 3 1 4 1 5 2 3 2 4 2 5 3 4 3 5 4 5

Примітка.

Під «лексикографічним (за числами) порядком» мається на увазі: спочатку треба виводити всі зростаючі послідовності, де на першому (крайньому зліва) місці 1, потім усі, де на першому місці 2, тощо. У свою чергу, всі зростаючі послідовності, де перші числа однакові між собою, мають бути відсортовані за другими числами; всі, де однакові як перші так і другі числа, мають бути відсортовані за третіми; тощо.

Примітка «за числами» означає, що якби спочатку сформували всі послідовності як рядки, і відсортували рядки, то порядок виявився б таким самим при $n \leqslant 9$, але при $n \geqslant 10$ з'являється відмінність. У цій задачі потрібно виводити послідовності, що починаються з 10, після послідовностей, що починаються з 9, а якби поформували рядки й відсортували їх як рядки, то по-

слідовності, що починаються з 10, потрапили б між послідовностями, що починаються з 1, і послідовностями, що починаються з 2 (що в цій задачі не буде зараховуватися).

Спробуємо порівняти цю задачу «Генерація розміщень» із попередньою «Генерація розміщень». Якщо робити їх рекурсивним перебором (як це й рекомендується), то загальна схема алгоритму для цих двох задач приблизно однакова. Те, що потрібно виводити лише зростаючі послідовності, водночас і полегшує реалізацію в одній частині алгоритма, й ускладнює в іншій. Полегшує тим, що не потрібно зберігати відносно складну інформацію про точний перелік чисел, які вже були використані, а які все ще можна використати далі; замість цього досить знати, що всі подальші числа мусять бути більші за останнє досі використане. Ускладнює тим, що потрібно писати деякі додаткові відтинання перебору гарантовано непотрібних гілок рекурсії, які можна вважати спрощеним аналогом відтинань методу гілок та меж (розгалужень та обмежень, branches and bounds). Наприклад, при $N = 10$, $K = 7$, не буває таких послідовностей, щоб і зростали, і на першому місці було число, строго більше 4 (почавши, наприклад, з 5, і щоразу збільшуючи якнайменше, всього на 1, отримаємо `5 6 7 8 9 10…???` — тобто, починаючи з 5, добудувати послідовність, щоб вийшло 7 різних натуральних чисел, менших або рівних 10, неможливо). От прямо зараз все ще цілком поміщається в обмеження (число 5 цілком собі з проміжку від 1 до 10), але (в чому й полягає часткова аналогія з методом гілок та меж), варто полдумати трохи наперед і помітити, що ще шести різних чисел, більших 5, але менших або рівних 10, нема. Якщо не помічати це вчасно, в цій програмі буде перевищення ліміту часу. Дуже істотне. Таке, що якщо програму не обривати насильно, то на деяких із тестів вона не має ніяких шансів скінчити роботу раніше, чим зламається комп'ютер. А правильно написаний перебір, який робить відтинання вчасно, цілком собі вкладається в десяті долі секунди.

Задача 0.0:18. «ads018»

Напишіть програму, яка знаходить всі розв'язки задачі про n ферзів. Прочитати постановку цієї задачі можна, наприклад, у вікіпедії.

Вхідні дані. В єдиному рядку задане єдине число n ($1 \leqslant n \leqslant 12$) — розмір дошки.

Результати. Виведіть спочатку всі знайдені розміщення, наприкінці їх кількість (символу-символ дотримуючись описаного далі формату).

Кожне окреме розміщення, де n ферзів не б'ють один одного, слід виводити окремим рядком такого вигляду: спочатку позначку вертикалі "`a`" (саму букву, без лапок), потім число, що означає, в якій горизонталі розміщено ферзя у цій вертикалі "`a`", спочатку позначку "`b`" та номер горизонталі ферзя у вертикалі "`b`", і так далі; все це в один рядок без пробілів чи будь-яких інших роздільників. Якщо $10 \leqslant n \leqslant 12$, деякі з номерів горизонталей виявляться одноцифрові, деякі двоцифрові; так і виводити; зокрема, для $n = 10$ першим з рядків відповіді повинно бути `a1b3c6d8e10f5g9h2i4j7`. Всі ці рядки, що описують можливі розміщення ферзів, обов'язково повинні бути впорядковані саме так, як виходить при класичному порядку перебору, а саме: в першу чергу за номером горизонталі у вертикалі "`a`", при однаковості цих номерів — за номером горизонталі у вертикалі "`b`", і так далі. (При $n \leqslant 9$ це збігається також зі стандартним словниковим порядком рядків, але при $n \geqslant 10$ це не так; який рядок повинен бути першим при $n = 10$, вже наведено.)

Наприкінці повинен бути ще один рядок, вигляду "`(totally ...)`" (без лапок), де замість трикрапки слід записати кількість знайдених розміщень. Рядок повинен містити лише один пробіл (між словом totally та знайденою кількістю) і завершуватися символом переведення рядка.

Приклади:

Клавіатура (stdin)	Екран (stdout)
1	a1 (totally 1)
2	(totally 0)
4	a2b4c1d3 a3b1c4d2 (totally 2)
6	a2b4c6d1e3f5 a3b6c2d5e1f4 a4b1c5d2e6f3 a5b3c1d6e4f2 (totally 4)

Примітка. У цьому завданні треба строго дотримуватися формату виведення, бо зараховуватися будуть тільки точні збіги байт-у-байт.

Задача 0.0:19. «ads019»

Задача відрізняється від попередньої «Задачі про n ферзів» лише:

- тим, що замість ферзів розглядаються нестандартні фігури «магараджі», які б'ють усі ті самі клітинки, що ферзь, і, додатково, ті клітинки, які б'є кінь;
- трохи збільшеним обмеженням: $1 \leqslant n \leqslant 14$.

Приклади:

Клавіатура (stdin)	Екран (stdout)
1	a1 (totally 1)
4	(totally 0)
10	a3b6c9d1e4f7g10h2i5j8 a4b8c1d5e9f2g6h10i3j7 a7b3c10d6e2f9g5h1i8j4 a8b5c2d10e7f4g1h9i6j3 (totally 4)

Задача 0.0:20. «ads020»

На площині задані координати n ($4 \leqslant n \leqslant 12$) різних вершин.

Знайти довжину найкоротшого замкнутого маршруту, що починається і закінчується в 1-й вершині і відвідує всі інші вершини по одному разу. Дозволяється (якщо так виявляється вигідно) «проїжджати через вершину, не зупиняючись» (див. приклад 1).

Довжина маршруту обчислюється як сума довжин складових його ребер, довжини окремих ребер обчислюються згідно звичайної евклідової метрики, як $\sqrt{(x_A - x_B)^2 + (y_A - y_B)^2}$.

Вхідні дані. Перший рядок містить кількість вершин n ($4 \leqslant n \leqslant 12$). Кожен з наступних n рядків містить по два розділених пропуском числа з плаваючою точкою — x- та y-координати відповідної вершини.

Результати. Результати повинні містити єдине число (з плаваючою крапкою) — знайдену мінімальну довжину замкнутого маршруту.

Приклади:

Клавіатура (stdin)	Екран (stdout)
4 0 0 2 0.2 7 0.7 5 0.5	1.40698258695692E+0001
5 1 0 4 4 3 2 4 0 1 1	1.24721359549995E+0001

Примітка. Задача з такими обмеженнями, по ідеї, повинна вирішуватися хоч методом гілок і меж, хоч динамічним програмуванням по підмножинам, хоч самими лише відтинаннями пошуку з поверненням (backtracking), що не намагається оцінювати можливий діапазон довжини ще не збудованої частини шляху.

Задача 0.0:21. «ads021»

На площині задані координати n ($4 \leqslant n \leqslant 15$) різних вершин.

Знайти найкоротший замкнутий маршрут, що починається і закінчується в 1-й вершині і відвідує всі інші вершини по одному разу. Дозволяється (якщо так виявляється вигідно) «проїжджати через вершину, не зупиняючись» (див. приклад 1).

Довжина маршруту обчислюється як сума довжин складових його ребер, довжини окремих ребер обчислюються згідно звичайної евклідової метрики, як $\sqrt{(x_A - x_B)^2 + (y_A - y_B)^2}$.

Вхідні дані. Перший рядок містить кількість вершин n ($4 \leqslant n \leqslant 15$). Кожен з наступних n рядків містить по два розділених пропуском числа з плаваючою точкою — x- та y-координати відповідної вершини.

Результати. Перший рядок повинен містити єдине число (з плаваючою крапкою) — знайдену мінімальну довжину замкнутого маршруту. Другий рядок повинен містити перестановку чисел 2, 3, ..., n — порядок, в якому треба відвідувати ці вершини. Чи*с*ла всередині другого рядка повинні розділятися одинарними пропусками.

Приклади:

Клавіатура (stdin)	Екран (stdout)
4 0 0 2 0.2 7 0.7 5 0.5	1.40698258695692E+0001 2 4 3
5 1 0 4 4 3 2 4 0 1 1	1.24721359549995E+0001 5 3 2 4

Примітка. Задача з такими обмеженнями, по ідеї, повинна вирішуватися хоч методом гілок і меж, хоч динамічним програмуванням по підмножинам. Але вона, по ідеї, не повинна вирішуватися самими лише відтинаннями пошуку з поверненням (backtracking), що не намагається оцінювати можливий діапазон довжин*и* ще не побудованої частини шляху.

Задача 0.0:22. «ads022»

Як відомо, *простим* називають таке натуральне число, яке має рівно два дільники — одиницю й самого себе. Перші десять простих чисел — 2, 3, 5, 7, 11, 13, 17, 19, 23, 29.

Напишіть програму, яка знайде усі підряд, у порядку зростання, прості ч*и*сла у проміжку від A до B (обидві межі включно).

Вхідні дані. У єдиному рядку через пробіл задані два натуральні числ*а* A та B, які є межами проміжку. Обмеження:

- $1 \leqslant A$;
- $B \leqslant 10^{13}$;
- $A \leqslant B \leqslant A + 100$.

Результати. Виведіть усі прості ч*и*сла проміжку, кожне у окремому рядку. Якщо буде введений проміжок, що не містить жодного простого числ*а*, слід нічого не виводити (навіть символа завершення рядка).

Приклади:

Клавіатура (stdin)	Екран (stdout)
2 5	2 3 5
4 4	

Примітка.

Задача 0.0:23. «ads023»

Є прямокутна таблиця розміром N рядків на M стовпчиків. У кожній клітинці записане ціле число. По ній потрібно пройти згори донизу, починаючи з будь-якої клітинки верхнього рядка, далі переходячи щоразу в *будь-яку* клітинку наступного рядка, і закінчити маршрут у якій-небудь клітинці нижнього рядка.

Напишіть програму, яка знаходитиме максимально можливу суму значень пройдених клітинок серед усіх допустимих шляхів.

Вхідні дані. У першому рядку записані N та M — кількість рядків і кількість стовпчиків ($1 \leqslant N, M \leqslant 200$); далі у кожному з наступних N рядків записано рівно по M розділених пробілами цілих чисел (модуль кожного не перевищує 10^6) — значення клітинок таблиці.

Результати. Вивести єдине ціле число — максимально можливу суму за маршрутами зазначеного вигляду.

Приклад:

Клавіатура (stdin)	Екран (stdout)
4 3 1 15 2 9 7 5 9 2 4 6 9 -1	42
3 3 1 1 100 1 1 10 100 1 1	210

Примітка. У першому тесті, $42 = 15 + 9 + 9 + 9$, маршрут (при нумерації з одиниці) `d[1][2]`→ `d[2][1]`→ `d[3][1]`→ `d[4][2]`. У другому тесті, $210 = 100 + 10 + 100$, маршрут (при нумерації з одиниці) `d[1][3]`→ `d[2][3]`→ `d[3][1]`.

Задача 0.0:24. «ads024»

В заданому додатному числі, яке містить не менше двох і не більше мільйона цифр, потрібно закреслити одну цифру так, щоб число, яке залишиться в результаті, було якнайбільшим.

Вхідні дані. Одне ціле значення n.

Результати. Число n без однієї цифри. Це число має бути максимальним серед усіх можливих варіантів закреслень цифри у числі n.

Приклади:

Клавіатура (stdin)
129
Екран (stdout)
29

Клавіатура (stdin)
998877665544332211234567891248
Екран (stdout)
99887766554433221234567891248

Примітка.

Останній приклад не порушує обмеження «не більше мільйона»: в ньому всього-навсього тридцять цифр, а дозволяється аж до мільйона. Цифр.

Задача 0.0:25. «ads025»

За правилами змагань ICPC, задача може бути лише або зарахованою, або ні (часткових балів не буває). Тому природньо, що при визначенні переможців найважливішим фактором є кількість розв'язаних задач. У випадку, коли різні команди розв'язали однакову кількість задач, вони ранжуються за так званим штрафом. Штраф нараховується так:

- якщо задача не зарахована, вона не приносить штрафа (незалежно від того, чи пробувала команда її здавати);
- якщо зарахована, то штраф дорівнює кількості хвилин (повних чи не повних) від моменту початку туру до моменту, коли зарахована ця задача, плюс по 20 хвилин за кожну невдалу спробу здачі;
- однак, якщо спроба відбувається вже після того, як задача зарахована, вона — байдуже, вдала чи ні — не впливає на штраф.

Команда Dream Team має дві надзвичайні властивості:

1. вона вміє не лише миттєво прочитати всі умови всіх задач, а ще й визначити, скільки хвилин займе процес розв'язування кожної з них;
2. вона ніколи не допускає помилок: якщо почне розв'язувати деяку задачу, то успішно здасть правильний розв'язок рівно через стільки хвилин, як визначила з самого початку.

Разом з тим, команда Dream Team не вміє у розпаралелювання обов'язків між членами команди, тому завжди і всюди може переходити до розв'язування наступної задачі лише після того, як (успішно) здала попередню.

Команда Dream Team в курсі, що на ICPC дозволяється здавати задачі в будь-якому порядку, і хоче вибрати такий порядок здачі, щоб *за рівно 300 хвилин (5 годин)* змагання отримати якнайкращий результат.

Вхідні дані. Перший рядок містить кількість задач N ($1 \leqslant N \leqslant 100$). Другий рядок містить N цілих чисел від 1 до 10^9 — визначені командою Dream Team тривалості розв'язування 1-ї, 2-ї, …, N-ї задач.

Результати. Знайдіть найкращий можливий результат (кількість задач та штраф), який може забезпечити команда Dream Team, якщо вибере найкращий можливий порядок розв'язування задач.

Приклади:

Клавіатура (stdin)	Екран (stdout)
5 100 500 20 180 200	3 440

Примітка. Раніше вже стверджували, що в поточному формулюванні задачі є певний слизький момент. Однак, він цілком вирішується, якщо враховувати, що приклад вхідних даних та результатів теж є частиною умови.

Задача 0.0:26. «ads026»

Щоб зобразити за допомогою паркету Супер-Креативний Візерунок, треба:

- N_1 дощечок розмірами 1×1,
- N_2 дощечок розмірами 2×1,
- N_3 дощечок розмірами 3×1,
- N_4 дощечок розмірами 4×1,
- N_5 дощечок розмірами 5×1.

Купити можна лише дощечки розмірами 5×1. Дощечки можна різати, але не можна склеювати. Наприклад, коли потрібні п'ять дощечок 2×1, їх не можна зробити з двох дощечок 5×1, але можна з трьох. Для цього дві з них розріжемо на три частини 2×1, 2×1 та 1×1 кожну, а третю — на дві частини 2×1 та 3×1. Отримаємо потрібні п'ять дощечок 2×1, а дві дощечки 1×1 та одна 3×1 підуть у відходи.

Напишіть програму, яка, прочитавши кількості дощечок N_1, N_2, N_3, N_4 та N_5, знайде, яку мінімальну кількість дощечок 5×1 необхідно купити.

Вхідні дані. Вхідні дані слід прочитати зі стандартного входу (клавіатури). Це будуть п'ять чисел N_1, N_2, N_3, N_4 та N_5 (саме в такому порядку), розділені пропусками (пробілами).

Результати. Єдине число (скільки дощечок треба купити) виведіть на стандартний вихід (екран).

Приклади:

Клавіатура (stdin)	Екран (stdout)
0 5 0 0 0	3
1 1 1 1 1	3

Задача 0.0:27. «ads027»

У цій задачі трохи інше, чим зазвичай, трактування жадібності. Тут є етап «перепробуємо таку, таку, … відповіді, та виберемо з них максимальну»; однак, ідея, завдяки якій цей максимум не доводиться шукати серед усіх $\Theta(N^2)$ варіантів (пари кожної стінки з кожною іншою), а вдається обійтися значно меншою кількістю ($\Theta(N)$), все ж може бути розцінена як жадібна.

Герой відомого мультсеріала Коливан вирішив побудувати собі басейн. Оскільки він дуже скупий, він намагається використати будівлю, що вже існує.

Будівля являє собою абсолютно рівний коридор одиничної ширини, в якому є N перегородок. Якщо розмістити вісь Ox вздовж коридору, усі перегородки будуть знаходитись точно в її цілочисельних координатах з кроком 1, причому по ширині перегородки займають увесь коридор, а висоти можуть відрізнятися. Басейн, що створюється, повинен мати максимально великий можливий об'єм за умови, що з усіх існуючих перегородок потрібно залишити *тільки* дві та збільшувати їх висоту заборонено.

Вхідні дані. Програма зчитує в першому рядку ціле число N ($2 \leqslant N \leqslant 10^5$) — кількість перегородок. Потім програма зчитує в другому рядку N цілих чисел a_i ($1 \leqslant a_i \leqslant 10^9$) — висоти перегородок.

Результати. Програма виводить єдине число — максимально можливий об'єм створюваного басейна з урахуванням вказаних обмежень.

Приклади:

Клавіатура (stdin)	Екран (stdout)
4 1 2 1 3	4

Задача 0.0:28. «ads028»

Є рюкзак місткістю V мм^3. Є N сипучих речовин; першої з них є у наявності v_1 мм^3, причому вона має питому вартість p_1 коп/мм^3; другої є v_2 мм^3, з питомою вартістю p_2 коп/мм^3; і так далі, до N-ої, якої є v_N мм^3, з питомою вартістю p_N коп/мм^3. Яку найбільшу вартість цих речовин можна набрати, не перевищивши місткість рюкзака? Припускаємо, ніби сипучі речовини можна розділити одну від одної, зовсім не втративши на цьому корисний об'єм рюкзака.

Вхідні дані. Перший рядок містить об'єм рюкзака V ($10 \leqslant V \leqslant 10^9$). Другий рядок містить кількість речовин N ($1 \leqslant N \leqslant 10^5$). Кожен з подальших N рядків містить спочатку кількість відповідної сипучої речовини v_i ($1 \leqslant v_i \leqslant 10^7$), потім (через пробіл) її питому вартість p_i ($1 \leqslant v_i \leqslant 10^7$). V та v_i вимірюються у мм^3, p_i — у коп/мм^3, N у штуках (безрозмірна).

Результати. Ваша програма повинна вивести єдине число — максимальну можливу вартість (у копійках) вибраних речовин.

Приклад:

Клавіатура (stdin)	Екран (stdout)
200 3 10 40000 50 2000 2000 5	500700

Задача 0.0:29. «ads029»

Відвідавши перед Новим роком великий магазин, ви обрали багато подарунків рідним та друзям. Зекономити певну кількість грошей вам можуть допомогти два типи передноворічних знижок, що діють у магазині:

1. При купівлі трьох товарів ви платите за них як за два найдорожчих з них.
2. При купівлі чотирьох товарів ви платите за них як за три найдорожчих з них.

Таким чином, певні товари можна об'єднати у трійки або четвірки і заплатити за них менше. Треба визначити найменшу можливу суму грошей, яка буде витрачена на придбання усіх подарунків. Наприклад, якщо ціни п'яти обраних подарунків складають: 50, 80, 50, 100, 20, то можна окремо придбати чотири перших товари, отримати за них знижку, та потім купити подарунок, що залишився за його номінальну ціну. Загалом вся покупка буде коштувати 250 грошових одиниць, замість 300.

Напишіть програму, що за цінами усіх подарунків знаходить мінімальну суму грошей, якої вистачить на їх купівлю.

Вхідні дані. Перший рядок містить одне ціле число N ($0 \leqslant N \leqslant 10000$). Другий рядок містить N натуральних чисел — ціни подарунків. Сума цін усіх подарунків менша за 10^9. Об'єднувати можна не лише ті товари, що йдуть підряд у вхідних даних.

Результати. Єдиний рядок має містити одне ціле число — знайдену мінімальну суму грошей, за яку можна купити усі подарунки.

Приклади:

Клавіатура (stdin)	Екран (stdout)
5 50 80 50 100 20	250

Задача 0.0:30. «ads030»

Перед святами Шеф отримує дуже багато запрошень на святкові засідання. Щоб краще планувати свій час, Шеф увів правило, щоб у кожному запрошенні був чітко вказаний відрізок часу $[a_i; b_i]$, коли триває засідання. Шеф не любить половинчатих рішень, тому або перебуває на засіданні увесь вказаний час, або не приходить на нього зовсім. Між відвідинами засідань має бути хоча б мінімальна перерва, тобто Шеф може встигнути на j-те (за списком запрошень) після i-го, тоді й тільки тоді, коли $a_j > b_i$. Напишіть програму, що допомагатиме Шефу відвідати якомога більше засідань.

Вхідні дані. Програма читає спочатку кількість заходів N, де $2 \leqslant N \leqslant 500$, потім N пар $[a_i; b_i]$ (кожна пара в окремому рядку). Гарантовано, що $-10^9 \leqslant a_i < b_i < 10^9$.

Результати. Програма має вивести єдине число — максимально можливу кількість засідань, які можливо вибрати для відвідання.

Приклади:

Клавіатура (stdin)	Екран (stdout)
3 2 5 1 3 4 6	2

Задача 0.0:31. «ads031»

Учасників полярної експедиції, які зимували на крижині, спіткало велике нещастя: крижина розкололася, і всі вони опинилися на маленькому її уламку. Потрібно було якнайшвидше переправитися через широку тріщину. У їх розпорядженні є двомісний надувний човен. Для кожного полярника відомий час, за який він може переправитися на цьому човні через тріщину. Якщо ж у човні перебувають 2 полярники, час переправи дорівнює часу менш розторопного з них (тобто того, хто потребує більше часу). За який мінімальний час всі полярники можуть переправитися на велику крижину?

Вхідні дані. Програма зчитує у першому рядку натуральне число N ($3 \leqslant N \leqslant 1000$) — кількість полярників, а у другому рядку через пропуски — N натуральних чисел, не більших 10000, які задають час переправи кожного полярника.

Результати. Програма виводить одне число — шукану мінімальну тривалість переправи.

Приклади:

Клавіатура (stdin)	Екран (stdout)
4 1 6 7 8	23
4 300 500 800 1000	2800

Примітка. Тут треба грамотно скомбінувати *дві* жадібні стратегії. Перший тест з умови розв'язується однією з них, другий — іншою з них, решта тестів — або однією з цих самих двох стратегій, або грамотною комбінацією обох цих жадібних стратегій.

Задача 0.0:32. «ads032»

Однією з багатьох проблем технічної підготовки до виборів є друк бюлетенів. Щоб забезпечити належний ступінь захисту, всі бюлетені друкують на одному поліграфічному комбінаті. Оскільки на бюлетенях вказують номер округу, поліграфкомбінату доводиться мати справу з N різними замовленнями (де N — кількість округів); виконувати ці замовлення можна лише послідовно одне за одним; для бюлетенів кожного i-го округу відомий час друкування $t_{i,1}$.

Бюлетені на різні округи може відвозити різний транспорт. Причому, цього транспорту достатньо, щоб ні при якому порядку друку не виникало затримок, пов'язаних з його (транспорту) очікуванням. Тим не менш, перевезення бюлетенів кожного i-го округу все ж займає значний час $t_{i,2}$.

Момент остаточної готовності до виборів настає тоді, коли бюлетені вже доставлені в усі округи.

Напишіть програму, яка визначатиме такий порядок друку бюлетенів, щоб проміжок часу від початку друкування до моменту остаточної готовності був якомога меншим.

Вхідні дані. у першому рядку вказана кількість округів N ($2 \leqslant N \leqslant 10^5$), наступні N рядків містять по два натуральні числа кожен — час друкування $t_{i,1}$ та час доставки $t_{i,2}$ бюлетенів відповідного округу. Усі значення $t_{i,1}$ та $t_{i,2}$ в межах $2 \leqslant t \leqslant 10^4$.

Результати. Єдине ціле число — знайдений мінімальний час від початку друкування до моменту остаточної готовності.

Приклади:

Клавіатура (stdin)	Екран (stdout)
3 10 5 5 20 5 5	25
4 10 5 5 12 25 8 12 6	57

Задача 0.0:33. «ads033»

Однією з багатьох проблем технічної підготовки до виборів є друк бюлетенів. Щоб забезпечити належний ступінь захисту, всі бюлетені друкують на одному поліграфічному комбінаті. Оскільки на бюлетенях вказують номер округу, поліграфкомбінату доводиться мати справу з N різними замовленнями (де N — кількість округів); виконувати ці замовлення можна лише послідовно одне за одним; для бюлетенів кожного i-го округу відомий час друкування $t_{i,1}$.

Бюлетені на різні округи може відвозити різний транспорт. Причому, цього транспорту достатньо, щоб ні при якому порядку друку не виникало затримок, пов'язаних з його (транспорту) очікуванням. Тим не менш, перевезення бюлетенів кожного i-го округу все ж займає значний час $t_{i,2}$.

Момент остаточної готовності до виборів настає тоді, коли бюлетені вже доставлені в усі округи.

Напишіть програму, яка визначатиме такий порядок друку бюлетенів, щоб проміжок часу від початку друкування до моменту остаточної готовності був якомога меншим.

Вхідні дані. у першому рядку вказана кількість округів N ($2 \leqslant N \leqslant 10^5$), наступні N рядків містять по два натуральні числа кожен — час друкування $t_{i,1}$ та час доставки $t_{i,2}$ бюлетенів відповідного округу. Усі значення $t_{i,1}$ та $t_{i,2}$ в межах $2 \leqslant t \leqslant 10^4$.

Результати. Перший рядок має містити знайдений мінімальний час від початку друкування до моменту остаточної готовності. Наступні N рядків повинні задавати порядок друку замовлень. Тобто, спочатку номер округа, з якого слід почати друк бюлетенів, потім номер округа, бюлетені якого слід друкувати наступним, і так далі. Якщо можливі різні порядки, які забезпечують однаковий мінімальний час, слід вивести будь-який один з них.

Приклади:

Клавіатура (stdin)	Екран (stdout)
3 10 5 5 20 5 5	25 2 1 3
4 10 5 5 12 25 8 12 6	57 3 4 2 1

Задача 0.0:34. «ads034»

Є n дерев, розміщених уздовж дороги в точках з координатами x_1, x_2, …, x_n. Кожне дерево має свою висоту h_i. Дерево можна зрубати і повалити або ліворуч, або праворуч. Тоді воно буде займати або відрізок $[x_i - h_i, x_i]$, або $[x_i; x_i + h_i]$ відповідно. Поки дерево не зрубане, воно займає точку з координою x_i. Дерево можна повалити, якщо на відрізку, який воно має займати після звалювання, немає жодної зайнятої точки.

Яку найбільшу кількість дерев можна повалити?

Вхідні дані. У першому рядку задане ціле число n ($1 \leqslant n \leqslant 10^5$) — кількість дерев. У наступних n рядках задані пари цілих чисел x_i, h_i ($1 \leqslant n \leqslant 10^9$) — координата і висота i-го дерева. Пари задані у порядку зростання x_i. Жодні два дерева не знаходяться в точці з однаковою координатою.

Результати. Потрібно вивести одне число — максимальну кількість дерев, які можна зрубати й повалити згідно зазначених правил.

Приклади:

Клавіатура (stdin)	Екран (stdout)
4 10 4 15 1 19 3 20 1	4
5 1 2 2 1 5 10 10 9 19 1	3

Задача 0.0:35. «ads035»

Шеф завжди приділяє всім відвідувачам рівні проміжки часу (наприклад, кожному по п'ять хвилин); щоб попасти на прийом, слід напередодні записатися у секретарки.

При реєстрації відвідувач вказує єдиний інтервал часу, що задається парою $[A_i; B_i]$ (початковий та кінцевий моменти, коли він згоден *заходити* на прийом). A_i і B_i — невід'ємні цілі числа, що означають кількість інтервалів прийому, що пройшли з початку робочого дня Шефа (отже, момент початку робочого часу Шефа має номер 0). Допоможіть секретарці обробляти зібрані записи і складати графік прийому.

Вхідні дані. Перший рядок містить кількість відвідувачів ($2 \leqslant N \leqslant 10^4$), далі йдуть ще N рядків, у кожному з яких по два числа $_i$ і B_i, $0 \leqslant A_i \leqslant B_i \leqslant 2N$.

Результати. Програма повинна вивести на екран 1 (якщо встановити графік прийому можливо), або 0 (якщо неможливо).

Приклади:

Клавіатура (stdin)	Екран (stdout)
3 1 2 0 1 2 2	1
3 1 2 1 2 1 2	0
3 1 2 1 2 2 4	1

Примітка. У наступній задачі приклад описує для кожного з цих самих прикладів вхідних даних також один з можливих порядків, у якому слід заходити відвідувачам.

Задача 0.0:36. «ads036»

Задача відрізняється від попередньої лише *тим, що потрібно вивести детальнішу відповідь (якщо прийняти всіх відвідувачів згідно з їх побажаннями можна, то потрібен також порядок («розклад»)).*

Шеф завжди приділяє всім відвідувачам рівні проміжки часу (наприклад, кожному по п'ять хвилин); щоб попасти на прийом, слід напередодні записатися у секретарки.

При реєстрації відвідувач вказує єдиний інтервал часу, що задається парою $[A_i; B_i]$ (початковий та кінцевий моменти, коли він згоден *заходити* на прийом). A_i і B_i — невід'ємні цілі числа, що означають кількість інтервалів прийому, що пройшли з початку робочого дня Шефа (отже, момент початку робочого часу Шефа має номер 0). Допоможіть секретарці обробляти зібрані записи і складати графік прийому.

Вхідні дані. Перший рядок містить кількість відвідувачів ($2 \leqslant N \leqslant 10^4$), далі йдуть ще N рядків, у кожному з яких по два числа $_i$ і B_i, $0 \leqslant A_i \leqslant B_i \leqslant 2N$.

Результати. Програма повинна вивести на екран 1 (якщо встановити графік прийому можливо), або 0 (якщо неможливо). Якщо відповідь позитивна (1), то далі в тому ж рядку через пропуски програма має вивести послідовність чисел-номерів відвідувачів у порядку, як вони

потрапляють на прийом. Якщо потрібно, щоб в якийсь момент ніхто не заходив на прийом, слід виводити -1.

Приклади:

Клавіатура (stdin)	Екран (stdout)
3 1 2 0 1 2 2	1 2 1 3
3 1 2 1 2 1 2	0
3 1 2 1 2 2 4	1 -1 1 2 3

Примітка. Зверніть увагу, що B_i є останнім моментом, коли відвідувач усе ще *згоден* заходити на прийом (і в першому, і в третьому прикладах лише завдяки цьому і вдається побудувати порядок прийому).

Зверніть увагу, що справді бувають ситуації, коли розклад можна побудувати лише таким, коли в деякі моменти часу ніхто не заходить на прийом.

Задача 0.0:37. «ads037»

Задача відрізняється від попередньої лише *збільшеним обмеженням на N; відповідно, від позаминулої — як збільшеним обмеженням на N, так і необхідністю вивести детальнішу відповідь (якщо прийняти всіх відвідувачів згідно з їх побажаннями можна, то потрібен також порядок («розклад»)).*

Шеф завжди приділяє всім відвідувачам рівні проміжки часу (наприклад, кожному по п'ять хвилин); щоб попасти на прийом, слід напередодні записатися у секретарки.

При реєстрації відвідувач вказує єдиний інтервал часу, що задається парою $[A_i; B_i]$ (початковий та кінцевий моменти, коли він згоден *заходити* на прийом). A_i і B_i — невід'ємні цілі числа, що означають кількість інтервалів прийому, що пройшли з початку робочого дня Шефа (отже, момент початку робочого часу Шефа має номер 0). Допоможіть секретарці обробляти зібрані записи і складати графік прийому.

Вхідні дані. Перший рядок містить кількість відвідувачів ($2 \leqslant N \leqslant 10^5$), далі йдуть ще N рядків, у кожному з яких по два числа $_i$ і B_i, $0 \leqslant A_i \leqslant B_i \leqslant 2N$.

Результати. Програма повинна вивести на екран 1 (якщо встановити графік прийому можливо), або 0 (якщо неможливо). Якщо відповідь позитивна (1), то далі в тому ж рядку через пропуски програма має вивести послідовність чисел-номерів відвідувачів у порядку, як вони потрапляють на прийом. Якщо потрібно, щоб в якийсь момент ніхто не заходив на прийом, слід виводити -1.

Приклади:

Клавіатура (stdin)	Екран (stdout)
3 1 2 0 1 2 2	1 2 1 3
3 1 2 1 2 1 2	0
3 1 2 1 2 2 4	1 -1 1 2 3

Примітка. Зверніть увагу, що B_i є останнім моментом, коли відвідувач усе ще *згоден* заходити на прийом (і в першому, і в третьому прикладах лише завдяки цьому і вдається побудувати порядок прийому).

Зверніть увагу, що справді бувають ситуації, коли розклад можна побудувати лише таким, коли в деякі моменти часу ніхто не заходить на прийом.

Задача 0.0:38. «ads038»

порожній, це так треба

Задача 0.0:39. «ads039»

Найголовніша відмінність цієї задачі від початкової задачі серії «Комп'ютерна гра (платформи)» полягає в додатковому обмеженні на кількість використання суперприйомів (тут — від $k_{\min}$ до $k_{\max}$, там — будь-яка).

Ця задача відрізняється від наступної задачі «Платформи з обмеженням на кількість суперприйомів (2)» лише обмеженнями на кількість платформ та на дозволений обсяг пам'яті.

У багатьох старих іграх з двовимірною графікою можна зіткнутися з такою ситуацією. Якийнебудь герой стрибає по платформах (або острівцям), які висять у повітрі. Він повинен перебратися від одного краю екрану до іншого. При цьому при стрибку з платформи на сусідню, герой витрачає $|y_2 - y_1|$ одиниць енергії, де y_1 і y_2 — висоти, на яких розташовані ці платформи. Крім того, у героя є суперприйом, який дозволяє перескочити через платформу, але на це витрачається $3 \cdot |y_3 - y_1|$ одиниць енергії. (Суперприйом *можна* застосовувати багатократно.) **Кількість використань суперприйому обмежена й повинна перебувати в межах від $k_{\min}$ до $k_{\max}$ разів (обидві межі включно).** Звісно, енергію слід витрачати максимально економно.

Нехай вам відомі координати всіх платформ у порядку від лівого краю до правого. Чи зможете ви знайти, яка мінімальна кількість енергії необхідна герою, щоб дістатися з першої платформи до останньої?

Вхідні дані. У першому рядку записано кількість платформ n ($1 \leqslant n \leqslant 100$). Другий рядок містить n натуральних чисел, що не перевищують 30000 — висоти, на яких розташовані платформи. Третій рядок містить два цілі невід'ємні числа k_{min} та k_{max} ($0 \leqslant k_{\text{min}} \leqslant k_{\text{max}} \leqslant \frac{n-1}{2}$).

Результати. Виведіть єдине число — мінімальну кількість енергії, яку має витратити гравець.

Приклади:

Клавіатура (stdin)	Екран (stdout)
3 1 5 10 0 1	9
3 1 5 10 1 1	27
3 1 5 2 0 1	3
3 1 5 2 0 0	7
5 1 2 3 30 31 0 1	30
5 1 2 3 30 31 1 2	34

Примітка.

Тест 1 Вигідно стрибати, не користуючись суперприйомом (використавши його 0 разів).

Тест 2 Герой зобов'язаний використати суперприйом рівно один раз, і не має іншого вибору, крім як стрибати з першої платформи на останню.

Тест 3 Вигідно використати один суперприйом, щоб стрибнути з першої платформи на останню.

Тест 4 Суперприйомів фактично нема (кількість=0), тож нема іншого вибору, крім як стрибати послідовно через усі платформи одна за одною.

Тест 5 Вигідно стрибати, не користуючись суперприйомом (використавши його 0 разів)

Тест 6 Герой зобов'язаний використати суперприйом або один раз, або двічі; вибираючи, чи краще використати його двічі (з 1 на 3, потім з 3 на 31), чи один раз з 1 на 3, чи один раз з 2 на 30, чи один раз з 3 на 31, бачимо, що найвигідніше використати один раз з 1 на 3.

Задача 0.0:40. «ads040»

Ця задача відрізняється від попередньої задачі «Платформи з обмеженням на кількість суперприйомів» лише обмеженнями на кількість платформ та на дозволений обсяг пам'яті.

У багатьох старих іграх з двовимірною графікою можна зіткнутися з такою ситуацією. Який-небудь герой стрибає по платформах (або острівцям), які висять у повітрі. Він повинен перебратися від одного краю екрану до іншого. При цьому при стрибку з платформи на сусідню,

герой витрачає $|y_2 - y_1|$ одиниць енергії, де y_1 і y_2 — висоти, на яких розташовані ці платформи. Крім того, у героя є суперприйом, який дозволяє перескочити через платформу, але на це витрачається $3 \cdot |y_3 - y_1|$ одиниць енергії. (Суперприйом *можна* застосовувати багатократно.) **Кількість використань суперприйому обмежена й повинна перебувати в межах від $k_{\min}$ до $k_{\max}$ разів (обидві межі включно).** Звісно, енергію слід витрачати максимально економно.

Нехай вам відомі координати всіх платформ у порядку від лівого краю до правого. Чи зможете ви знайти, яка мінімальна кількість енергії необхідна герою, щоб дістатися з першої платформи до останньої?

Вхідні дані. У першому рядку записано кількість платформ n ($1 \leqslant n \leqslant 10000$). Другий рядок містить n натуральних чисел, що не перевищують 30000 — висоти, на яких розташовані платформи. Третій рядок містить два цілі невід'ємні числа $k_{\min}$ та $k_{\max}$ ($0 \leqslant k_{\min} \leqslant k_{\max} \leqslant \frac{n-1}{2}$).

Результати. Виведіть єдине число — мінімальну кількість енергії, яку має витратити гравець.

Приклади:

Клавіатура (stdin)	Екран (stdout)
3 1 5 10 0 1	9
3 1 5 10 1 1	27
3 1 5 2 0 1	3
3 1 5 2 0 0	7
5 1 2 3 30 31 0 1	30
5 1 2 3 30 31 1 2	34

Примітка.

Тест 1 Вигідно стрибати, не користуючись суперприйомом (використавши його 0 разів).

Тест 2 Герой зобов'язаний використати суперприйом рівно один раз, і не має іншого вибору, крім як стрибати з першої платформи на останню.

Тест 3 Вигідно використати один суперприйом, щоб стрибнути з першої платформи на останню.

Тест 4 Суперприйомів фактично нема (кількість=0), тож нема іншого вибору, крім як стрибати послідовно через усі платформи одна за одною.

Тест 5 Вигідно стрибати, не користуючись суперприйомом (використавши його 0 разів)

Тест 6 Герой зобов'язаний використати суперприйом або один раз, або двічі; вибираючи, чи краще використати його двічі (з 1 на 3, потім з 3 на 31), чи один раз з 1 на 3, чи один раз з 2 на 30, чи один раз з 3 на 31, бачимо, що найвигідніше використати один раз з 1 на 3.

Задача 0.0:41. «ads041»

Ця задача відрізняється від наступної лише тим, що тут потрібно знаходити й виводити лише мінімальну сумарну висоту, а в наступній ще й розподіл блоків по рядкам.

Абзац містить блоки різної висоти (наприклад, звичайні слова й математичні системи). Цей абзац навряд чи поміщається весь в один рядок, тому його, найімовірніше, потрібно розбити на рядки. Висота кожного рядка визначається за найвищим з блоків цього рядка. Висота абзацу дорівнює сумі висот всіх рядків (ніби друкуємо не на окремих сторінках, а на довгому свитку). Довжина кожного рядка визначається як сумарна ширина блоків, включених до цього рядка (враховувати пробіли між блоками не потрібно). Можливість розбиття блоку для перенесення з рядка на рядок не розглядається. Змінювати порядок блоків не можна (і це природньо: ну куди це годиться, щоб у друкарні взяли й переставляли слова в тексті, який вони мали лише надрукувати, а не відредагувати).

Потрібно знайти таке розбиття абзацу на рядки, щоб висота абзацу була мінімальною. Ширина і висота кожного блоку (w_i, h_i) та максимально допустима довжина рядка TW (скорочення від TextWidth) задаються у вхідних даних.

Вхідні дані. У першому рядку записано два числа: TW (максимально допустима довжина рядка) і N (кількість блоків в абзаці, де $5 \leqslant N \leqslant 5000$). У наступних N рядках записано по два числа w_i та h_i — ширина і висота чергового блоку.

Всі розміри (окремих блоків та TW) — натуральні числа, не більші 10^6. Гарантовано, що для кожного окремо взятого блоку $w_i \leqslant TW$.

Результати. Виведіть єдине число — мінімальну висоту абзацу.

Приклади:

Клавіатура (stdin)	Екран (stdout)
7 6 3 1 2 1 2 3 1 1 3 3 3 1	5

Примітка. Як отримати відповідь 5 при цих вхідних даних, див. у наступній задачі.

Задача 0.0:42. «ads042»

Ця задача відрізняється від попередньої лише тим, що там потрібно було знаходити й виводити лише мінімальну сумарну висоту, а в цій ще й розподіл блоків по рядкам.

Абзац містить блоки різної висоти (наприклад, звичайні слова й математичні системи). Цей абзац навряд чи поміщається весь в один рядок, тому його, найімовірніше, потрібно розбити на рядки. Висота кожного рядка визначається за найвищим з блоків цього рядка. Висота абзацу дорівнює сумі висот всіх рядків (ніби друкуємо не на окремих сторінках, а на довгому свитку). Довжина кожного рядка визначається як сумарна ширина блоків, включених до цього рядка (враховувати пробіли між блоками не потрібно). Можливість розбиття блоку для перенесення з рядка на рядок не розглядається. Змінювати порядок блоків не можна (і це природньо: ну куди це годиться, щоб у друкарні взяли й переставляли слова в тексті, який вони мали лише надрукувати, а не відредагувати).

Потрібно знайти таке розбиття абзацу на рядки, щоб висота абзацу була мінімальною. Ширина і висота кожного блоку (w_i, h_i) та максимально допустима довжина рядка TW (скорочення від TextWidth) задаються у вхідних даних.

Вхідні дані. У першому рядку записано два числа: TW (максимально допустима довжина рядка) і N (кількість блоків в абзаці, де $5 \leqslant N \leqslant 5000$). У наступних N рядках записано по два числа w_i та h_i — ширина і висота чергового блоку.

Всі розміри (окремих блоків та TW) — натуральні числа, не більші 10^6. Гарантовано, що для кожного окремо взятого блоку $w_i \leqslant TW$.

Результати. У першому рядку виведіть мінімальну висоту абзацу. У другому — кількість рядків M, на які потрібно розбити абзац, а в кожному з наступних M рядків виведіть кількість блоків у відповідному рядку абзаца.

Приклади:

Клавіатура (stdin)	Екран (stdout)
7 6 3 1 2 1 2 3 1 1 3 3 3 1	5 3 2 3 1

Примітка.

1. Зрозуміло, в цій задачі для деяких вхідних даних можливі різні детальні правильні відповіді (розподіл блоків по рядкам), які дають однакову мінімальну сумарну висоту. Ваша програма повинна виводити (єдино можливу) правильну мінімальну сумарну висоту та будь-який один із правильних розподілів блоків по рядкам.
2. Зверніть увагу, що у прикладі з умови мінімальна сумарна висота досягається при немінімальній кількості рядків: у першому рядку абзаца можна розмістити не два (як у наведеній відповіді), а три блоки, і тоді всі шість блоків поміщаються у два рядки; але це невигідно, бо сумарна висота тоді вийшла б $\max(1,1,3)+\max(1,3,1) = 3+3 = 6$, а при наведеному розбитті на три рядки сумарна висота виходить $\max(1,1) + \max(3,1,3) + \max(1) = 1+3+1 = 5$.

Задача 0.0:43. «ads043»

У деякій державі в обігу перебувають банкноти певних номіналів. Національний банк хоче, щоб банкомат видавав будь-яку запитану суму за допомогою мінімального числа банкнот, вважаючи, що запас банкнот кожного номіналу необмежений. Допоможіть Національному банку вирішити цю задачу.

Вхідні дані. Перший рядок містить натуральне число N, що не перевищує 50 — кількість номіналів банкнот у обігу. Другий рядок вхідних даних містить N різних натуральних чисел x_1, x_2,…, x_N, що не перевищують 10^5 — номінали банкнот. Третій рядок містить натуральне число S, що не перевищує 10^5 — суму, яку необхідно видати.

Результати. Програма повинна вивести єдине число — знайдену мінімальну кількість банкнот. Якщо видати вказану суму вказаними банкнотами неможливо, програма повинна вивести рядок “`No solution`” (без лапок, перша літера велика, решта маленькі).

Приклади:

Клавіатура (stdin)	Екран (stdout)
7 1 2 5 10 20 50 100 72	3
2 20 50 60	3

Примітка.

У першому тесті, 72 можна видати трьома банкнотами 50, 20 і 2. У другому, 60 можна видати трьома банкнотами 20, 20 і 20.

Задача 0.0:44. «ads044»

Зверніть увагу, що тут Вас просять відновити детальну відповідь (сукупність банкнот), ще й **в умовах відносно невеликого обсягу дозволеної пам'яті**.

У деякій державі в обігу перебувають банкноти певних номіналів. Національний банк хоче, щоб банкомат видавав будь-яку запитану суму за допомогою мінімального числа банкнот, вважаючи, що запас банкнот кожного номіналу необмежений. Допоможіть Національному банку вирішити цю задачу.

Вхідні дані. Перший рядок містить натуральне число N, що не перевищує 100 — кількість номіналів банкнот у обігу. Другий рядок вхідних даних містить N різних натуральних чисел x_1, x_2,..., x_N, що не перевищують 10^6 — номінали банкнот. Третій рядок містить натуральне число S, що не перевищує 10^6 — суму, яку необхідно видати.

Результати. Програма повинна знайти подання числа S у вигляді суми доданків з множини x_i, що містить мінімальне число доданків, і вивести це подання у вигляді послідовності чисел, розділених пробілами.

Якщо таких подань існує декілька, то програма повинна вивести будь-яке (одне) з них. Якщо такого подання не існує, то програма повинна вивести рядок "`No solution`" (без лапок, перша літера велика, решта маленькі).

Приклади:

Клавіатура (stdin)	Екран (stdout)
7 1 2 5 10 20 50 100 72	50 20 2
2 20 50 60	20 20 20

Задача 0.0:45. «ads045»

Правда, вам набридли абсолютно неприродні олімпіадні задачі? Ну навіщо казати: «Якби банкомат заправили банкнотами по 10, 50 і 60, то суму 120 варто було б видавати не як $100 + 10 + 10$, а як $60 + 60$»… Ніхто ж не стане вводити в обіг банкноти номіналом 60… Тому зараз пропонуємо розв'язати абсолютно практичну задачу.

В обігу перебувають банкноти номіналами 1, 2, 5, 10, 20, 50, 100, 200 та 500 гривень. Причому, банкноти номіналами 1 грн та 2 грн в банкомати ніколи не кладуть. Так що в банкоматі є N_5 штук банкнот по 5 грн, N_{10} штук банкнот по 10 грн, N_{20} штук банкнот по 20 грн, N_{50} штук банкнот по 50 грн, N_{100} штук банкнот по 100 грн, N_{200} штук банкнот по 200 грн та N_{500} штук банкнот по 500 грн.

Для банкомата діють адміністративне обмеження «видавати не більш як 2000 грн за один раз» та технічне обмеження «видавати не більш як 40 банкнот за один раз». В останньому обмеженні мова йде про сумарну кількість банкнот (можливо, різних номіналів).

Напишіть програму, яка визначатиме, як видати потрібну суму мінімально можливою кількістю банкнот (з урахуванням указаних обмежень).

Вхідні дані. Програма читає одним рядком сім чисел N_5, N_{10}, N_{20}, N_{50}, N_{100}, N_{200} та N_{500} — кількості банкнот відповідних номіналів; потім, наступним рядком, суму S, яку треба видати.

Усі чи*с*ла вхідних даних є цілими, перебувають у межах від 0 включно до 5000 включно.

Результати. Програма повинна вивести сім чисел — скільки треба видати банкнот по 5 грн, по 10 грн, по 20 грн, по 50 грн, по 100 грн, по 200 грн та по 500 грн. Ці сім чисел треба вивести в один рядок, розділяючи пропусками. Сума цих чисел (загальна кількість банкнот до видачі) повинна бути мінімально можливою.

Якщо видати суму, дотримуючись обмежень, неможливо, програма повинна замість відповіді вивести (єдине) число -1.

Приклади:

Клавіатура (stdin)	Екран (stdout)
0 100 1 100 0 0 0 190	0 2 1 3 0 0 0
5000 2000 5000 2000 5000 2000 500 17	-1

Задача 0.0:46. «ads046»

На площині задана сукупність точок. Гарантовано, що різні точки сукупності дійсно різні (ніяка пара точок не збігається). Напишіть програму, яка знаходить мінімальну серед усіх відстаней між цими точками. (Інакше кажучи — відстань між парою найближчих точок). Відстані рахувати у звичайній евклідовій метриці, як $\sqrt{(x_A - x_B)^2 + (y_A - y_B)^2}$.

Вхідні дані. Перший рядок містить цілу кількість точок N ($2 \leqslant N \leqslant 123456$), потім слідують N рядків, кожен з яких містить розділені пропуском x- і y- координати точки (цілі числа, що не перевищують за модулем 10^8 (сто мільйонів)).

Результати. Програма повинна вивести єдине число — знайдену мінімальну відстань. Відповідь буде зараховуватися, якщо відносна похибка не перевищить 10^{-9}.

Приклади:

Клавіатура (stdin)	Екран (stdout)
3 1 4 -1 1 3 2	2.8284271247

Примітка.

Це **складна** задача. Вкладені цикли `for for` **повинні не проходити** за часом, це так задумано. Рекомендований чесний спосіб вирішення цієї задачі оснований на «Поділяй і володарюй». В принципі можливо вкластися в обмеження часу й не зовсім чесним способом, придумавши одну неочевидну оптимізацію вкладених циклів `for for`, котра працює правильно і швидко не завжди, але майже завжди, й кінець кінцем пройти тести (можливо, не з першого разу). Я навіть згоден поставити бали (у дисципліну «Алгоритми і структури даних», не тут на сайті) і за одне, і за інше (за «Поділяй і володарюй» більші, за пропихування не зовсім чесної оптимізації, якщо це вдасться, менші).

Задача 0.0:47. «ads047»

Завдання в цілому аналогічне завданню «Структура (колекція) set» (і наполегливо рекомендується зробити спочатку його), але елементами повинні бути не числа, а дещо складніші об'єкти: прямокутники, кожен з яких описується довжинами двох своїх сторін *a*, *b* (два цілі числа) та кольором (`string`) *col*. Повертати прямокутники дозволено (*b*×*a* — те само, що *a*×*b*), а колір враховується. Інакше кажучи, елемент-прямокутник належить множині тоді й тільки тоді, коли множина містить елемент-прямокутник, в якого такі самі розміри, але порядок цих розмірів може бути хоч таким самим, хоч переставленим; при цьому колір повинен бути таким самим (у смислі звичайної case-sensitive рівності рядків).

Кожен із запитів `ADD` та/або `PRESENT` має по три параметри *a b col*, саме в такому порядку. Вони задаються в одному рядку через пробіли.

Напишіть програму, яка виконуватиме послідовність запитів виду `ADD` *a b col*, `PRESENT` *a b col* та `COUNT` (без параметрів). Програму обов'язково слід писати за допомогою бібліотечного типу (колекції) set (її реалізації в конкретних мовах програмування можуть називатися `HashSet`, `TreeSet`, `SortedSet`, …).

Виконання кожного запиту виду `ADD` *a b col* повинно додавати прямокутник до множини (якщо такий само прямокутник вже є, додавання ще однієї копії не змінює множину). На екран при цьому нічого не виводиться.

При виконанні кожного запиту виду `PRESENT` *a b col* має видаватися повідомлення `YES` або `NO` (великими літерами, в окремому рядку), відповідно до того, чи є такий елемент-прямокутник у множині; значення множини при цьому не змінюється.

При виконанні кожного запиту виду `COUNT` має видаватися на екран в окремому рядку кількість різних елементів-прямокутників у множині; значення множини при цьому не змінюється.

Вхідні дані.

У першому рядку задано кількість запитів N ($5 \leqslant N \leqslant 2 \cdot 10^5$), далі йдуть N рядків, кожен з яких містить по одному запиту згідно з описаним форматом. Всі параметри *a* та *b* є цілими числами від 1 до 1234567890. Всі параметри *col* є рядками, що містять від 1 до 15 латинських (англійських) літер, без будь-яких інших символів.

Результати.

Виводьте окремими рядками результати запитів `PRESENT` (рівно одне «YES» чи «NO» великими буквами) та `COUNT` (рівно одне число); на запити `ADD` нічого не виводьте.

Приклади:

Клавіатура (stdin)	Екран (stdout)
11 ADD 5 2 red COUNT PRESENT 5 2 red PRESENT 5 2 ReD PRESENT 2 5 red ADD 2 5 green COUNT ADD 2 5 red COUNT ADD 5 2 red COUNT	1 YES NO YES 2 2 2

Примітка.

Перша відповідь `1` правильна, бо на той момент множина містить рівно один елемент-прямокутник `5 2 red`.

Друга відповідь `YES` правильна, бо коли питають про наявність прямокутника, з такими самими розмірами `5 2` в такому ж порядку, причому такого самого кольору `red`, то він вже наявний.

Третя відповідь `NO` правильна, бо хоч розміри `5 2` такі самі в такому ж порядку, але колір інший (case-sensitive передбачає, що `ReD` не дорівнює `red`).

Четверта відповідь `YES` правильна, бо розміри `2 5` і є тими самим числами (хоч і в іншому порядку), що й `5 2`, і колір правильний.

П'ята відповідь `2` правильна, бо перед тим був запит `ADD` з елементом-прямокутником, що відрізняється від вже наявного кольором, тому тоді його додали, від чого кількість елементів множини змінилася.

Шоста відповідь `2` правильна, бо перед тим був запит `ADD` з елементом-прямокутником `2 5 red`, що (незважаючи на переставляння місцями чисел) вважається рівним одному з уже наявних у множині прямокутників `5 2 red`, тому множина не змінилася й продовжує містити рівно 2 елементи-прямокутники.

Сьома відповідь `2` правильна, бо перед тим був запит `ADD` з елементом-прямокутником, за всіма ознаками рівним вже наявному (найпершому) елементу-прямокутнику, тому множина не змінилася й продовжує містити рівно 2 елементи-прямокутники.

Абсолютно повне виконання завдання передбачає, що воно виконане і через бібліотечну колекцію `HashSet` (з написанням власних `GetHashCode` та `Equals`), і через бібліотечну колекцію `SortedSet` (з написанням власного компаратора). В разі виконання іншою мовою програмування, з'ясуйте відповідні деталі самостійно, але це повинен бути бібліотечний спосіб через хеш-таблиці та бібліотечний спосіб через дерева. Само собою, можна виконати частину, на відповідну частину балів.

Задача 0.0:48. «ads048»

Напишіть програму, яка згенерує багато (беззмістовних з точки зору природньої мови) string-ів однакової довжини, котрі мають однаковий hash, якщо рахувати його за правилами типу String мови Java, тобто `s[0]*31`$^{n-1}$`+s[1]*31`$^{n-2}$`+...+s[n-2]*31+s[n-1]` (де n — довжина рядка; інакше кажучи, обчислення hash-коду починається з числа 0, далі на кожному кроці спочатку старе значення домножається на 31, потім до нього додається ASCII-код чергової літери слова); обчислення hash-коду відбувається у знаковому 32-бітовому типі, з переповненнями.

Вхідні дані. В одному рядку через пропуски (пробіли) записані: спочатку потрібна однакова довжина рядків L ($8 \leqslant L \leqslant 32$); потім кількість блоків K ($1 \leqslant K \leqslant 2017$), таких, що всередині кожного блоку всі string-и повинні мати однаковий hash-код, а для різних блоків різні; потім кількість різних рядків всередині кожного з цих блоків N ($2 \leqslant N \leqslant 2017$).

K та N *не будуть великими одночасно*; гарантовано, що сумарний розмір всіх рядків результату не перевищуватиме одного мегабайту.

Результати. Слід вивести $K \cdot N$ рядків, кожен з яких містить унікальний (у межах поточного запуску програми) string довжиною рівно L, причому всі рядки з 1-го по N-й повинні мати однаковий hash-код; якщо $K \geqslant 2$, то всі рядки з $(N+1)$-го по $(2N)$-й повинні мати однаковий між собою, але відмінний від рядків з 1-го по N-й, hash-код, і так далі. (В цьому поясненні вважаємо, що нумерація починається з одиниці). Таким чином, рядки з 1-го по N-ий утворюватимуть один блок, рядки з $(N+1)$-го по $(2N)$-й (якщо $K \geqslant 2$, тобто якщо їх взагалі треба виводити) утворюватимуть ще один блок, і так далі. Розділяти ці блоки якимись додатковими роздільниками не треба.

Символи рядків, які виводить Ваша програма, не зобов'язані бути літерами, але зобов'язані мати ASCII-коди від 33 до 126 (обидві межі включно).

Клавіатура (stdin)	Екран (stdout)
8 3 3	BBBBBBBB AaBBBBBB BAaBBBBB FFFFFFFF EeFFFFFF FEeFFFFF SSSSSSSS RrSSSSSS SRrSSSSS

Примітка.

У цій задачі можуть бути різні правильні відповіді. Ваша програма повинна виводити будь-яку правильну.

Поки що якість автоматичної перевірки цієї задачі вивірена не досить ретельно, в разі обґрунтованих сумнівів щодо правильності перевірки звертайтеся.

Головна ідея і наведена в прикладі введення/виведення, і була проговорена на лекції: якщо у двох сусідніх символах лівий збільшити на 1, а правий зменшити на 31 (у смислі відповідного збільшення/зменшення ASCII-кодів цих символів), то hash-код (обчислений саме за цією hash-функцією! це важливо!) лишиться незмінним. Однак, для проходження всіх тестів цієї задачі потрібно трошки посилити цю ідею:

- помітити, що такі збільшення та зменшення можна застосувати також і одночасно в різних парах сусідніх символів;
- помітити, що абсолютно те саме буде також і при збільшенні лівого на 2 та зменшенні правого на 62, або зменшенні лівого на 1 та збільшенні правого на 31, або зменшенні лівого на 2 та збільшенні правого на 62. (Забезпечити, щоб усі перелічені варіанти давали символи в

проміжку ASCII-кодів від 33 до 126, неможливо; але це й не треба: досить забезпечити в тій самій парі сусідніх символів 2–3 зсуви (3–4 значення).)

Наприклад, усі «слова» з переліку

- 3&&
- 2E&
- 1d&
- 3%E
- 2DE
- 1cE
- 3$d
- 2Cd
- 1bd

мають однаковий Java-hash-код 50227. Їх вдалося зробити аж 9, при довжині всього 3 та дотримавшись обмеження «ASCII-коди всіх символів від 33 до 126».

Задача 0.0:49. «ads049»

Завдання в цілому аналогічне завданням «Структура (колекція) set» та «Структура (колекція) set від власного типу — 1» (і наполегливо рекомендується зробити спочатку їх, саме в такому порядку), але елементами повинні бути не числа, а дещо складніші об'єкти: прямокутники, кожен з яких описується довжинами двох своїх сторін *a*, *b* (два цілі числа) та кольором (`string`) *col*.

Слід постійно підтримувати чотири множини, виходячи з таких чотирьох можливих трактувань задачі:

1. повертати прямокутники заборонено (довжина повинна бути довжиною, ширина повинна бути шириною), колір враховується; інакше кажучи, елемент-прямокутник належить множині тоді й тільки тоді, коли множина містить елемент-прямокутник, в якого такі самі довжина і ширина (в такому самому порядку), причому такого самого кольору (в смислі звичайної case-sensitive рівності рядків);
2. повертати прямокутники заборонено (довжина повинна бути довжиною, ширина повинна бути шириною), колір ігнорується; інакше кажучи, елемент-прямокутник належить множині тоді й тільки тоді, коли множина містить елемент-прямокутник, в якого такі самі довжина і ширина (в такому самому порядку), але колір не перевіряється, може бути хоч однаковим, хоч різним;
3. повертати прямокутники дозволено (можна хоч залишити довжину довжиною та ширину шириною, хоч повернути, зробивши довжину шириною та ширину довжиною), колір враховується; інакше кажучи, елемент-прямокутник належить множині тоді й тільки тоді, коли множина містить елемент-прямокутник, в якого такі самі розміри, але порядок цих розмірів може бути хоч таким самим, хоч переставленим; при цьому колір повинен бути таким самим (у смислі звичайної case-sensitive рівності рядків);
4. повертати прямокутники дозволено (можна хоч залишити довжину довжиною та ширину шириною, хоч повернути, зробивши довжину шириною та ширину довжиною), колір ігнорується; інакше кажучи, елемент-прямокутник належить множині тоді й тільки тоді, коли множина містить елемент-прямокутник, в якого такі самі розміри, але порядок цих розмірів може бути хоч

таким самим, хоч переставленим; при цьому колір не перевіряється, може бути хоч однаковим, хоч різним.

(див. також пояснення до прикладів).

Кожен із запитів `ADD` та/або `PRESENT` має по три параметри *a b col*, саме в такому порядку. Вони задаються в одному рядку через пробіли.

Напишіть програму, яка виконуватиме послідовність запитів виду `ADD` *a b col*, `PRESENT` *a b col* та `COUNT` (без параметрів). Програму обов'язково слід писати за допомогою бібліотечного типу (колекції) set (її реалізації в конкретних мовах програмування можуть називатися `HashSet`, `TreeSet`, `SortedSet`, ...).

Виконання кожного запиту виду `ADD` *a b col* повинно додавати прямокутник до всіх чотирьох множин (якщо такий само прямокутник вже є, додавання ще однієї копії не змінює множину; слід розуміти, що можливі ситуації, коли з точки зору деяких із трактувань 1, 2, 3, 4 такий прямокутник вже є й додавання ще однієї копії не змінить множину, а з точки зору інших трактувань такого прямокутника нема і його слід обов'язково додати). На екран при цьому нічого не виводиться.

При виконанні кожного запиту виду `PRESENT` *a b col* повинно видаватися рівно чотири повідомлення «`YES`» чи «`NO`» (великими літерами, в одному рядку через пробіли, без лапок), на позначення того, чи наявний такий прямокутник у кожній з цих множин, згідно описаних вище трактувань 1, 2, 3, 4, саме в такому порядку. Значення множин при цьому не змінюються.

При виконанні кожного запиту виду `COUNT` повинно видаватися (цифрами, в одному рядку через пробіли) рівно чотири кількості елементів, згідно описаних вище трактувань 1, 2, 3, 4, саме в такому порядку. Значення множин при цьому не змінюються.

Вхідні дані.

У першому рядку задано кількість запитів N ($5 \leqslant N \leqslant 2 \cdot 10^5$), далі йдуть N рядків, кожен з яких містить по одному запиту згідно з описаним форматом. Всі параметри *a* та *b* є цілими числами від 1 до 1234567890. Всі параметри *col* є рядками, що містять від 1 до 15 латинських (англійських) літер, без будь-яких інших символів.

Результати.

Виводьте на стандартний вихід (екран) в окремих рядках результати запитів `PRESENT` (рівно чотири «`YES`» чи «`NO`» великими літерами, в одному рядку через пробіли, без лапок) та COUNT (рівно чотири числа, в одному рядку через пробіли); на запити ADD нічого виводити не слід.

Приклади:

Клавіатура (stdin)	Екран (stdout)
12 ADD 5 2 red COUNT PRESENT 5 2 red PRESENT 5 2 ReD PRESENT 2 5 red ADD 2 5 green COUNT ADD 2 5 red COUNT PRESENT 7 17 yellow ADD 3 3 brown COUNT	1 1 1 1 YES YES YES YES NO YES NO YES NO NO YES YES 2 2 2 1 3 2 2 1 NO NO NO NO 4 3 3 2

Примітка.

Перша відповідь `1 1 1 1` правильна, бо для будь-якого трактування правильно, що в той' момент множина містить рівно один елемент-прямокутник `5 2 red`.

Друга відповідь `YES YES YES YES` правильна, бо коли питають про наявність прямокутника, з такими самими розмірами `5 2` в такому ж порядку, причому такого самого кольору `red`, то за будь-яким з трактувань він вже наявний.

Третя відповідь `NO YES NO YES` правильна, бо коли розміри `5 2` такі самі в такому ж порядку, але колір інший (case-sensitive передбачає, що `ReD` не дорівнює `red`), то лише за трактуваннями 2 та 4, які ігнорують колір, можна вважати, що такий прямокутник вже є.

Четверта відповідь `NO NO YES YES` правильна, бо коли розміри `2 5` є тими самим числами, але в іншому порядку, то лише за трактуваннями, які дозволяють повертати прямокутник, тобто трактуванням 3 (враховуючи однаковість кольору `red`) та 4, можна вважати, що такий прямокутник вже є.

П'ята відповідь 2 2 2 1 правильна, бо за трактуванням 4 розміри 2 5 рівноцінні розмірам 5 2, а колір не перевіряється, тому додавання ще однієї копії не змінює множину у трактуванні 4; за будь-яким іншим трактуванням або одна, або обидві з причин «в уже наявного прямокутника інший колір, тому цей новий» та/або «в уже наявного прямокутника розміри в іншому порядку, тому цей новий» роблять необхідним додавання нового елемента-прямокуника.

Шоста відповідь 3 2 2 1 правильна, бо за трактуванням 1 прямокутник 2 5 red новий і його треба додати, тоді як за всіма іншими трактуваннями такий прямокутник вже був (як 2 5 green або 5 2 red).

Сьома відповідь NO NO NO NO правильна, бо нічого досить схожого на прямокутник 7 17 yellow раніше не згадувалося, навіть якщо дозволяти переставляння сторін місцями та/або ігнорування кольору.

Восьма відповідь 4 3 3 2 правильна, бо прямокутник 3 3 brown (який є також квадратом, але це неважливо) є новим за всіма трактуваннями і його треба додати.

Абсолютно повне виконання завдання передбачає, що воно виконане і через бібліотечну колекцію HashSet (з написанням власних GetHashCode та Equals), і через бібліотечну колекцію SortedSet (з написанням власного компаратора). Само собою, можна виконати частину, на відповідну частину балів.

Рекомендується описати чотири різні власні класи, які відповідають прямокутникам згідно кожного з чотирьох трактувань. Чи організовувати їх так, щоб якісь з них були нащадками інших, чи робити, щоб вони всі були нащадками одного абстрактного класу, чи ще якось; чи користуватися поліморфізмом — все це повністю на вибір студента. ООП взагалі не є предметом розгляду курсу «Алгоритми та структури даних». Якщо вдасться зробити завдання, використавши лише лямбда-функції без написання чотирьох різних власних класів — теж можна. Але навряд чи вийде зробити це завдання, взагалі не використавши ні власні класи, ні лямбда-функції.)

Перша відповідь `1` правильна, бо на той момент множина містить рівно один елемент-прямокутник `5 2 red`.

Друга відповідь `YES` правильна, бо коли питають про наявність прямокутника, з такими самими розмірами `5 2` в такому ж порядку, причому такого самого кольору `red`, то він вже наявний.

Третя відповідь `NO` правильна, бо хоч розміри `5 2` такі самі в такому ж порядку, але колір інший (case-sensitive передбачає, що `ReD` не дорівнює `red`).

Четверта відповідь `YES` правильна, бо розміри `2 5` і є тими самим числами (хоч і в іншому порядку), що й `5 2`, і колір правильний.

П'ята відповідь `2` правильна, бо перед тим був запит `ADD` з елементом-прямокутником, що відрізняється від вже наявного кольором, тому тоді його додали, від чого кількість елементів множини змінилася.

Шоста відповідь `2` правильна, бо перед тим був запит `ADD` з елементом-прямокутником `2 5 red`, що (незважаючи на переставляння місцями чисел) вважається рівним одному з уже

наявних у множині прямокутників `5 2 red`, тому множина не змінилася й продовжує містити рівно 2 елементи-прямокутники.

Сьома відповідь `2` правильна, бо перед тим був запит `ADD` з елементом-прямокутником, за всіма ознаками рівним вже наявному (найпершому) елементу-прямокутнику, тому множина не змінилася й продовжує містити рівно 2 елементи-прямокутники.

Абсолютно повне виконання завдання передбачає, що воно виконане і через бібліотечну колекцію `HashSet` (з написанням власних `GetHashCode` та `Equals`), і через бібліотечну колекцію `SortedSet` (з написанням власного компаратора). В разі виконання іншою мовою програмування, з'ясуйте відповідні деталі самостійно, але це повинен бути бібліотечний спосіб через хеш-таблиці та бібліотечний спосіб через дерева. Само собою, можна виконати частину, на відповідну частину балів.

Задача 0.0:50. «ads050»

Напишіть програму, яка реалізовуватиме у бінарному дереві пошуку дії «вставити» та «знайти» (за значенням). Програма повинна виконувати послідовність запитів вигляду "`ADD` *n*", "`SEARCH` *n*", та "`PRINTTREE`", де *n* — натуральне число.

Для кожного запиту "`ADD` *n*" слід виконати такі дії: якщо вказаного числа ще нема в дереві, вставити у дерево і вивести на екран слово "`DONE`", якщо вже є — залишати дерево як було (не вставляти додаткову копію) і виводити на екран слово "`ALREADY`".

Для кожного запиту "`SEARCH` *n*" слід виводити на екран слово "`YES`" (якщо значення знайдене у дереві) або слово "`NO`" (якщо не знайдене); при виконанні запитів `SEARCH` дерево не змінюється.

Для кожного запиту "`PRINTTREE`" слід виводити на екран усе дерево, обов'язково дотримуючись того ж формату, що й наведений далі алгоритм.

Вхідні дані. В кожному рядку вхідних даних записаний один із запитів "`ADD` *n*", або "`SEARCH` *n*", або "`PRINTTREE`" (без лапок; слова записані великими латинськими буквами; для запитів `ADD` та `SEARCH` число відділене від слова одинарним пробілом і перебуває в межах від 1 до 1234567890). Гарантується, що запити `PRINTTREE` будуть лише у моменти, коли дерево не порожнє. Загальна кількість запитів не перевищує 1000, з них не більше 20 запитів `PRINTTREE`.

Результати. Для кожного запиту виводити відповідь на нього. Для запитів `ADD` та `SEARCH` — відповідне слово в окремому рядку (великими латинськими буквами, без лапок). На запит `PRINTTREE` треба виводити дерево, обов'язково відповідно до такого алгоритму:

```
public void PrintTree(Node root, int level)
{
    if (root == null)
        return;
    PrintTree(root.left, level + 1);
    Console.WriteLine(new String('.', level) + root.Data);
    PrintTree(root.right, level + 1);
}
```

(Початковий виклик цього методу — `PrintTree(root, 0)`. Якщо описати цей алгоритм словами, вийде приблизно так: слід виводити спочатку ліве піддерево, потім корінь, потім праве піддерево, і результат виходить впорядкованим згори донизу так, як при класичному зображенні мав би бути впорядкований зліва направо; перед кожним елементом треба виводити стільки крапочок, який це ярус, вважаючи, що корінь є єдиним елементом ярусу 0 (слід взагалі не ставити крапочок), сини кореня є елементами ярусу 1 (слід поставити рівно одну крапочку), «онуки» (сини синів) кореня є елементами ярусу 2 (слід поставити рівно дві крапочки), і так далі.)

Приклади:

Клавіатура (stdin)	Екран (stdout)
ADD 2 ADD 3 ADD 2 SEARCH 2 ADD 5 PRINTTREE SEARCH 7	DONE DONE ALREADY YES DONE 2 .3 ..5 NO

Примітка. Детальніше пояснимо приклад:
`DONE` — у відповідь на ADD 2
`DONE` — у відповідь на ADD 3
`ALREADY` — у відповідь на ADD 2
`YES` — у відповідь на SEARCH 2
`DONE` — у відповідь на ADD 5
`2` — цим рядком починається відповідь на `PRINTTREE`
`.3`
`..5` — а цим рядком та сама відповідь на той самий `PRINTTREE` закінчується
`NO` — у відповідь на SEARCH 7

Задача 0.0:51. «ads051»

Ця задача повністю включає в себе попередню задачу «Дерева (вставка, пошук)». Тому, в разі виникнення проблем із розв'язуванням цієї задачі, рекомендується спочатку зробити попередню, а також прочитати примітку наприкінці умови цієї задачі.

Напишіть програму, яка реалізовуватиме у бінарному дереві пошуку дії «вставити» та «знайти» (за значенням). Програма повинна виконувати послідовність запитів вигляду "`ADD` *n*", "`SEARCH` *n*", "`DELETE` *n*" та "`PRINTTREE`", де *n* — натуральне число.

Для кожного запиту "`ADD` *n*" слід виконати такі дії: якщо вказаного числа ще нема в дереві, вставити у дерево і вивести на екран слово "`DONE`", якщо вже є — залишати дерево як було (не вставляти додаткову копію) і виводити на екран слово "`ALREADY`".

Для кожного запиту "`SEARCH` *n*" слід вивести на екран слово "`YES`" (якщо значення знайдене у дереві) або слово "`NO`" (якщо не знайдене); при виконанні запитів `SEARCH` дерево не змінюється.

Для кожного запиту "`DELETE` *n*" слід виконати такі дії: якщо вказане число є в дереві, вилучити його з дерева і вивести на екран слово "`DONE`", якщо нема — залишати дерево як було

і виводити на екран слово “CANNOT”. При видаленні елемента, що має обох синів, обов’язково обмінювати значення з максимальним елементом лівого піддерева.

Для кожного запиту “PRINTTREE” слід вивести на екран усе дерево, обов’язково дотримуючись того ж формату, що й наведений далі алгоритм.

Вхідні дані. В кожному рядку вхідних даних записаний один із запитів “ADD *n*”, або “SEARCH *n*”, або “DELETE *n*”, або “PRINTTREE” (без лапок; слова записані великими латинськими буквами; для запитів ADD, SEARCH та DELETE число відділене від слова одинарним пробілом і перебуває в межах від 1 до 1234567890). Гарантується, що запити PRINTTREE будуть лише у моменти, коли дерево не порожнє. Загальна кількість запитів не перевищує 1000, з них не більше 20 запитів PRINTTREE.

Результати. Для кожного запиту виводити відповідь на нього. Для запитів ADD та SEARCH — відповідне слово в окремому рядку (великими латинськими буквами, без лапок). На запит PRINTTREE треба виводити дерево, обов’язково відповідно до такого алгоритму:

```
public void PrintTree(Node root, int level)
{
    if (root == null)
        return;
    PrintTree(root.left, level + 1);
    Console.WriteLine(new String('.', level) + root.Data);
    PrintTree(root.right, level + 1);
}
```

(Початковий виклик цього методу — PrintTree(root, 0). Якщо описати цей алгоритм словами, вийде приблизно так: слід виводити спочатку ліве піддерево, потім корінь, потім праве піддерево, і результат виходить впорядкованим згори донизу так, як при класичному зображенні мав би бути впорядкований зліва направо; перед кожним елементом треба виводити стільки крапочок, який це ярус, вважаючи, що корінь є єдиним елементом ярусу 0 (слід взагалі не ставити крапочок), сини кореня є елементами ярусу 1 (слід поставити рівно одну крапочку), «онуки» (сини синів) кореня є елементами ярусу 2 (слід поставити рівно дві крапочки), і так далі.)

Приклади:

Клавіатура (stdin)	Екран (stdout)
ADD 2 ADD 7 ADD 5 PRINTTREE ADD 5 DELETE 3 ADD 1 PRINTTREE DELETE 7 PRINTTREE	DONE DONE DONE 2 ..5 .7 ALREADY CANNOT DONE .1 2 ..5 .7 DONE .1 2 .5

Примітка. 1) Враховувати вищезгадану фразу «При видаленні елемента, що має обох синів, обов’язково обмінювати значення з максимальним елементом лівого піддерева.», на жаль,

абсолютно необхідно для зарахування розв'язку. Попри те, що це взагалі-то не є об'єктивно необхідним для непротирічивості дерева (цей спосіб правильний, але є й інші правильні способи). Однак, у цій задачі все-таки необхідно застосувати саме конкретно цей спосіб, бо конкретно в ній визначення правильності Вашої програми робиться просто порівнянням її відповіді зі зразком (без глибшого аналізу смислу), і це переписуватися не буде. Кому це не подобається — може просто не робити цю задачу.

2) Одна з відносно частих помилок при виконанні цієї задачі — написаний студентом код помилково виводить два повідомлення `YES` замість одного при успішному видаленні елемента, що має обох синів. Переконайтеся, що у Вашого коду нема конкретно цієї проблеми.

Задача 0.0:52. «ads052»

Нема жорсткої вимоги робити цю задачу саме з використанням бінарного дерева пошуку, що підтримує операцію «знайти за номером», але це — один зі способів. Гарантовано, що вхідні дані згенеровані суто випадково, тому небалансоване дерево не має істотних переваг над балансованим (процес балансування можна й не писати). Можна використати й деяку іншу структуру даних (взагалі не дерево, а зовсім іншу). Однак, просто масив, начебто, повинен виявлятися не досить ефективним.

Напишіть програму, яка реалізовуватиме деяку колекцію, яка підтримує дії «вставити» та «знайти» (за значенням). Програма повинна виконувати послідовність запитів вигляду "`ADD` *v*", "`SEARCH` *v*", "`INDEX` *i*", "`VAL2INDEX` *v*" та "`COUNT`", де *v*, *i* — цілі числа.

Для кожного запиту "`ADD` *v*" слід виконати такі дії: якщо вказаного числа *v* ще нема в колекції, то вставити (додати) його, якщо вже є — залишити як було (не вставляти додаткову копію). При виконанні запитів `ADD` ніякого результату не виводиться, лише змінюється внутрішній стан колекції.

Для кожного запиту "`SEARCH` *v*" слід вивести на екран слово "`YES`" (якщо значення *v* знайдене) або слово "`NO`" (якщо не знайдене); при виконанні запитів `SEARCH` колекція не змінюється.

Для кожного запиту "`INDEX` *i*", слід вивести на екран значення, яке мало би вказаний порядковий номер *i*, якби колекція була відсортованим масивом з нумерацією з нуля. Інакше кажучи, таке значення з колекції, що рівно *i* інших значень колекції строго менші за знайдене. У випадку, якщо такого елемента не існує (значення *i* занадто велике або занадто мале), слід вивести слово "`NO`". В будь-якому разі, при виконанні запитів `INDEX` колекція не змінюється.

Для кожного запиту "`VAL2INDEX` *n*", якщо значення *n* є у колекції, то слід вивести, під яким індексом воно було б у колекції, якби колекція була відсортованим масивом з нумерацією з нуля. Інакше кажучи, якщо значення *n* є у колекції, то слід вивести, скільки інших елементів колекції строго менші вказаного *n*. У випадку, якщо колекція не містить такого значення, слід вивести слово "`NO`". В будь-якому разі, при виконанні запитів `VAL2INDEX` колекція не змінюється.

Для кожного запиту "`COUNT`", слід вивести поточну кількість елементів у колекції; колекція при цьому не змінюється.

Вхідні дані. В кожному рядку вхідних даних записаний один із запитів "`ADD` *v*", або "`SEARCH` *v*", або "`INDEX` *i*", або "`VAL2INDEX` *v*", або "`COUNT`" (без лапок; слова записані великими латинськими буквами; для всіх запитів, що передають число (всіх крім `COUNT`), число відділене від слова одинарним пробілом, і для запитів "`COUNT`" перебуває в межах від 0 до 10^6, а для решти запитів у межах від -10^9 до $+10^9$). Загальна кількість запитів не перевищує $5 \cdot 10^5$ (пів

мільйона). *Гарантовано, що вхідні дані згенеровані суто випадково, тому небалансоване дерево не має істотних переваг над балансованим (процес балансування можна й не писати).*

Результати. Для кожного запиту слід виводити відповідь на нього, в окремому рядку; якщо відповідь є словом, то великими латинськими буквами; в будь-якому разі, без лапок.

Приклади:

Клавіатура (stdin)	Екран (stdout)
20 ADD 50 ADD -6 COUNT ADD -2 ADD 57 INDEX 1 ADD -1 VAL2INDEX 50 ADD -2 ADD 29 ADD 19 INDEX 6 ADD 27 ADD 22 PRESENT -3 ADD -7 VAL2INDEX -7 VAL2INDEX 73 VAL2INDEX 22 PRESENT -7	2 -2 3 57 NO 0 NO 5 YES

Примітка. Детальніше пояснимо приклад:

`2` — у відповідь на COUNT, коли колекція містить –6, 50

`-2` — у відповідь на INDEX 1, коли колекція містить –6, –2, 50, 57.

`3` — у відповідь на VAL2INDEX 50, коли колекція містить –6, –2, –1, 50, 57.

`57` — у відповідь на INDEX 6, коли колекція містить –6, –2, –1, 19, 29, 50, 57 (причому, повторне включення –2 не змінило вміст колекції).

`NO` — у відповідь на PRESENT –3, коли колекція містить –6, –2, –1, 19, 22, 27, 29, 50, 57.

`0` — у відповідь на VAL2INDEX -7, коли колекція містить –7, –6, –2, –1, 19, 22, 27, 29, 50, 57.

`0` — у відповідь на VAL2INDEX 73, коли колекція містить –7, –6, –2, –1, 19, 22, 27, 29, 50, 57.

`5` — у відповідь на VAL2INDEX 22, коли колекція містить –7, –6, –2, –1, 19, 22, 27, 29, 50, 57.

`YES` — у відповідь на PRESENT -7, коли колекція містить –7, –6, –2, –1, 19, 22, 27, 29, 50, 57.

Задача 0.0:53. «ads053»

Напишіть програму, яка реалізовуватиме пошук у ширину в орієнтованому незваженому не мульти графі без петель. Вершини графа є цілими невід'ємними числами в діапазоні від 0 до N–1 (де N — кількість вершин у графі). Граф обов'язково подавати списками суміжності. (Наприклад, як `List<int>[]`, але чіткої вимоги щодо конкретного типу не ставиться. А вимога

просто використати списки суміжності ставиться; власне, інакше взагалі неясно як отримати правильний результат із прийнятними витратами часу й пам'яті.)

Вхідні дані. В першому рядку задано число NUM — кількість різних пошуків у ширину, які треба виконати (на різних графах). Далі йдуть NUM блоків, кожен з яких має таку структуру.

Перший рядок блоку містить два числа N та M, розділені пробілом — кількість вершин та кількість ребер графа. Далі йдуть M рядків, кожен з яких містить по два числа (розділені пробілом, від 0 до N–1 кожне) — початок та кінець відповідної дуги. Далі, в останньому рядку блоку, записане єдине число від 0 до N–1 — вершина, починаючи з якої треба запустити пошук.

Точні обмеження на N та M свідомо не повідомляються; приблизні — N не перевищує кількадесят тисяч, M не перевищує кількасот тисяч.

Результати. Виведіть NUM рядків, у кожному з яких по N_i чисел, розділених пробілами — відстані від указаної стартової вершини орграфа до його 0-ої, 1-ої, 2-ої і т. д. вершин. Якщо деяка вершина недосяжна з указаної початкової, замість відстані виводьте число 987654321.

У попередньому абзаці сказано про N_i (а не N), щоб підкреслити: кількості вершин різних графів можуть бути різними, тому різні рядки результатів можуть бути різної довжини.

Приклади:

Клавіатура (stdin)	Екран (stdout)
2 3 2 0 1 1 2 1 4 4 0 1 0 3 1 2 2 3 0	987654321 0 1 0 1 2 1

Примітка.

1. Переконайтеся, що Ваша програма правильно враховує, що за один запуск слід обробити кілька різних графів (зокрема, працює, коли «менший» граф йде після «більшого»).
2. У наведеному прикладі треба зробити 2 пошуки вшир: один — на орграфі з 3 вершин та 2 ребер $0 \to 1$ і $1 \to 2$, починаючи з вершини 1; інший — на орграфі з 4 вершин та 4 ребер $0 \to 1$, $0 \to 3$, $1 \to 2$ і $2 \to 3$, починаючи з вершини 0.

Задача 0.0:54. «ads054»

Нехай потрібно мати справу з незваженим неорієнтованим мультиграфом, причому в ньому дуже багато вершин і не дуже багато ребер. Причому, для повного щастя цей мультиграф *динамічний*, тобто змінний у часі: іноді приходять запити, що потрібно додати ребро між вершинами $v[j]$ та $v[k]$ (якщо ребро вже є — додати ще одне, на те і мультиграф) або вилучити ребро між вершинами $v[j]$ та $v[k]$ (якщо ребер кілька — має стати на одне менше; якщо одне — має не стати; якщо і так не було — так і лишається, що нема, бо навіть у мультиграфі кількість ребер не може бути від'ємною).

Раз «дуже багато вершин і не дуже багато ребер», однозначно не варто користуватися матрицею суміжності, а слід користуватися деяким аналогом списків суміжності. Але якщо можливо, що з деяких вершин може виходити по багато ребер, то при традиційній реалізації списків суміжності операція вилучення ребер буде надто повільною. Тому в такій ситуації може бути доцільним поєднання ідеї списків суміжності з іншими, чим масив або List, структурами даних. Як це найкраще робити мовою C# — питання відкрите, але, наприклад, можна розглянути спосіб `Dictionary<int,int>[]`, де кожен окремо взятий `Dictionary<int,int>` описує, куди йдуть ребра з поточної вершини графа, причому ключами є номери тих вершин, куди йдуть ребра, а значеннями — кількості таких ребер.)

Напишіть програму, яка оброблятиме послідовність запитів таких видів:

- `RESET_GRAPH` *num* — створити новий граф, що містить *num* вершин (з 0-ої по (*num*–1)-у) і не містить жодного ребра. Якщо перед цим вже зберігався якийсь граф, він знищується. Дія відбувається лише у пам'яті, нічого не виводиться.
- `ADD` *j k* — додати до неорієнтованого мультиграфа ребро $\{v[j], v[k]\}$. Дія відбуваються лише у пам'яті, нічого не виводиться.
- `DEL` *j* k — вилучити з неорієнтованого мультиграфа ребро $\{v[j], v[k]\}$. Якщо це можливо, тобто між цими вершинами є хоча б одне ребро, то дія відбувається лише у пам'яті, на екран нічого не виводиться. Якщо це неможливо, тобто між цими вершинами нема жодного ребра, на екран виводиться повідомлення "`CNNT`" (великими літерами, скорочено від "CANNOT"), а дані в пам'яті не змінюються.
- `CHK` *j k* — перевірити, чи існує ребро, яке з'єднує вершини $v[j]$ та $v[k]$. Треба вивести велику латинську літеру "`Y`" або "`N`" (скорочено від "YES"/"NO"); дані в пам'яті при цьому не змінюються.
- `CHECK_PATH` *j k* — перевірити, чи існує шлях довільної довжини, який з'єднує вершини $v[j]$ та $v[k]$. Треба вивести слово "`YES`" або "`NO`" (великими літерами); дані в пам'яті при цьому не змінюються.

Вхідні дані. У вхідних даних записано послідовність запитів RESET_GRAPH, ADD, DEL, CHK та CHECK_PATH — кожен у окремому рядку, згідно вищеописаного формату. Першим гарантовано йде RESET_GRAPH, далі порядок запитів довільний. При кожному запиті ADD, DEL, CHK або CHECK_PATH вказуються два різні номери вершин від 0 до *num*–1, де *num* — кількість вершин поточного графа (згідно останнього виконаного запиту RESET_GRAPH). Кількість запитів ніде не вказана. Відомо, що при перевірці на великих вхідних даних запитів ADD, DEL та CHK буде дуже багато, а запитів RESET_GRAPH та CHECK_PATH — лише по кілька штук.

Результати. Виведіть повідомлення згідно описаних у форматах запитів правил. Кожне повідомлення слід виводити в окремому рядку.

Приклади:

Клавіатура (stdin)	Екран (stdout)
RESET_GRAPH 3 ADD 0 1 ADD 0 1 ADD 1 2 CHK 0 2 CHECK_PATH 0 2 DEL 1 2 DEL 1 2 ADD 0 2 RESET_GRAPH 7 CHECK_PATH 0 2	N YES CNNT NO

Примітка.

В першому тесті відповідь “N” (однолітерна) видається на запит “CHK 0 2”, “YES” — на запит “CHECK_PATH 0 2”, “CNNT” — на другий із запитів “DEL 1 2”, “NO” (слово) — на останній із запитів “CHECK_PATH 0 2”.

Задача 0.0:55. «ads055»

Напишіть програму, яка реалізовуватиме пошук у ширину в простому графі, вершини якого не нумеровані й ідентифікуються словесними назвами.

Вхідні дані. В першому рядку вхідних даних задано число *NUM* — кількість різних пошуків у ширину, які треба виконати (на різних графах). Далі йдуть *NUM* блоків, кожен з яких має таку структуру.

Перший рядок блоку містить єдине ціле число *M* — кількість ребер графа. Далі йдуть *M* рядків, кожен з яких містить по дві назви (назви гарантовано не містять пробілів і відділені одна від одної одним пробілом) — кінці відповідного ребр*а*. Далі, в останньому рядку блоку, записана єдина назва — вершина, починаючи з якої треба запустити пошук (ця назва гарантовано хоча б раз згадувалася як кінець одного з ребер).

Результати. Виведіть на стандартний вихід (екран) *NUM* блоків, у кожному з яких записані відстані від вказаної початкової вершини до всіх досяжних (якщо є недосяжні вершини, вони взагалі не згадуються). Перелік має бути відсортований по назвам вершин, кожна пара (назва, відстань) має виводитися в окремому рядку, блоки мають бути відділені один від одного рядком “===” (три знаки “дорівнює”).

Приклади:

Клавіатура (stdin)	Екран (stdout)
2 2 Cherk Zol Cherk Sm Zol 4 A Bb Bb Ccc Ccc A Dddd Eeeee Bb	Cherk 1 Sm 2 Zol 0 === A 1 Bb 0 Ccc 1

Примітка.

Задачу можна розв’язати, наприклад, будь-яким з таких двох способів (можливі й інші, це лише приклади правильних):

- Граф подавати так само, як у «Пошуку в ширину–1», а для перетворень назв у номери та номерів у назви користуватися `SortedDictionary<string, int>` та `List<string>` відповідно. Для виведення у відсортованому порядку використати той самий `SortedDictionary<string, int>`, що перетворює назви у номери.
- Увесь час працювати безпосередньо з рядковими назвами, подаючи граф, наприклад, як `Dictionary<string, List<string> >`. Відповідно замінюються й решта структур да-

них. Зокрема, масив відстаней перетворюється у, наприклад, `SortedDictionary<string, int>`, який міститиме по суті готову відповідь конкретного пошуку.

Задача 0.0:56. «ads056»

Напишіть програму, яка знаходитиме відстані в неорієнтованому зваженому графі з невід'ємними довжинами ребер, від зазначеної вершини до всіх інших. Програма повинна працювати швидко для великих розріджених графів.

Вхідні дані. У першому рядку вхідних даних задано число *NUM* — кількість різних запусків алгоритму Дейкстри (на різних графах). Далі слідують *NUM* блоків, кожен з яких має таку структуру.

Перший рядок блоку містить два числа *N* і *M*, розділені пропуском — кількість вершин і кількість ребер графа. Далі слідують *M* рядків, кожен з яких містить по три цілих числа, розділені пробілами. Перші два з них в межах від 0 до *N*–1 кожне і позначають кінці відповідного ребр*а*, третє — в межах від 0 до 20000 і позначає довжину цього ребр*а*. Далі, в останньому рядку блоку, записане єдине число від 0 до *N*–1 — вершина, відстані від якої треба шукати.

Кількість різних графів в одному тесті *NUM* не перевищує 5. Кількість вершин не перевищує 60000, ребер — 200000.

Результати. Виведіть *NUM* рядків, у кожному з яких по N_i чисел, розділених пропусками — відстані від зазначеної початкової вершини зваженого неорієнтованого графа до його 0-ї, 1-ї, 2-ї і т. д. вершин (допускається зайвий пробіл після останнього числа). Якщо деяка вершина недосяжна від зазначеної початкової, замість відстані виводите число 2009000999 (гарантовано, що всі реальні відстані менші).

Приклади:

Клавіатура (stdin)	Екран (stdout)
1 5 7 1 2 5 1 3 2 2 3 4 2 4 3 3 4 6 0 3 20 0 4 10 1	18 0 5 2 8

Примітка.

Переконайтеся, що програма правильно враховує, що за один запуск слід обробити кілька різних графів.

Щоб забезпечити асимптотичну оцінку часу виконання $O(M \log N)$, слід:

1. Подавати граф списками суміжності і робити перебір сусідів аналогічно завданню «Пошук в ширину — 1». (Однак, списками суміжності слід подати *зважений* граф, тому елементами списків повинні бути вже не числ*а*, а, наприклад, структури з двох полів (вершина, куди йде ребро, і довжина ребра).)

2. При виборі, яка саме вершина статусу 1 має найнижчу оцінку відстані, користуватися деякою структурою даних, що вміє робити це ефективно. Зокрема (але не тільки), це може бути будь-який з таких способів:
   (а) деяка реалізація priority queue, наприклад, піраміда;
   (б) SortedSet.

   В будь-якому з цих випадків, зберігати в піраміді чи в SortedSet'і чи в ще якомусь аналогу треба не самі лише оцінки відстаней, а такі структури чи класи, де кожен примірник містить і оцінку відстані, і номер вершини, а компаратор заданий так, щоб зручно й ефективно знаходити мінімальну оцінку відстані.

Додаткову проблему створює те, що при виконанні алгоритму Дейкстри можлива зміна оцінки відстані до вершини статусу 1 (новий маршрут виявляється коротшим, чим знайдений раніше), після чого піраміда (чи SortedSet, чи ще якийсь аналог) має містити новий примірник структури/класу з тією ж вершиною, але меншою оцінкою відстані. Якщо лише вставляти новий примірник додатково до старого, це призводитиме до багатократного виймання з піраміди (чи SortedSet'а, чи ще якогось аналога) однієї з тієї ж вершини. Якщо таких вершин чимало, і з них виходить чимало ребер — це може призводити до істотного сповільнення роботи всього алгоритму. Наприклад, неважко побудувати граф, де оцінка відстані деякої вершини зменшується $\approx 0,4 \cdot N$ разів, і саме з цієї вершини виходить $\approx 0,5 \cdot N$ ребер. Погана реалізація алгоритму Дейкстри може $\approx (0,4 \cdot N) - 1$ раз переглядати оцінки всих цих $\approx 0,5 \cdot N$ вершин (без жодної на те потреби, бо саме в цьому випадку пізніше побудовані оцінки будуть гіршими (більшими) за раніше побудовані). Тобто, погана реалізація алгоритму Дейкстри запросто може виконати $O(N^2)$ відверто зайвої роботи, що у випадку розріджених графів прямо протирічить меті «реалізувати алгоритм Дейкстри складністю $O(M \log N)$». І в тестах цієї задачі такі (та ще деякі схожі) випадки є.

Щоб уникати таких непродуктивних ситуацій, варто забезпечити одну з двох властивостей:

- коли для деякої вершини зменшується оцінка відстані, відповідний примірник структури/класу слід вийняти з тієї структури даних, що використовується для пошуку мінімальної оцінки — так, щоб там взагалі ніколи не було кількох різних примірників, відповідних одній і тій самій вершині;
- коли деякий примірник структури/класу виймають з тієї структури даних, що використовується для пошуку мінімальної оцінки, треба додатково перевірити, чи така сама вершина ще не була вийнята раніше (іншими словами, чи вершина все ще статусу 1, а не статусу 2).

(Яку з цих властивостей краще підтримувати? Якщо використовувати піраміду, то асимптотично ефективно забезпечити першу властивість важко (піраміда не підтримує ефективних вилучень не кореневих елементів), а другу неважко, тому варто забезпечувати другу. Якщо використовувати SortedSet, то забезпечувати першу стає легше (підтримання другої при цьому не ускладнюється), й тому більш-менш байдуже. Чи можна забезпечувати відразу обидві ці властивості? Можна, але не варто, бо будь-яка одна вже позбавить від вищезгаданої відверто зайвої роботи.)

Задача 0.0:57. «ads057»

Напишіть програму, яка буде обробляти послідовність запитів таких видів:

`CLEAR` — зробити піраміду порожньою (якщо в піраміді вже були якісь елементи, видалити все). Дія відбувається тільки з даними в пам'яті, нічого не виводиться.

`ADD` *n* — додати в піраміду число *n*. Дія відбувається тільки з даними в пам'яті, нічого не виводиться.

`EXTRACT` — вийняти з піраміди максимальне значення. Слід і змінити дані в пам'яті, і вивести на екран або знайдене максимальне значення, або, якщо піраміда була порожньою, слово "`CANNOT`" (великими літерами, без лапок).

Вхідні дані. У вхідних даних записано довільну послідовність запитів CLEAR, ADD і EXTRACT — кожен в окремому рядку, відповідно до вищеописаного формату. Сумарна кількість всіх запитів не перевищує 200000.

Результати. Для кожного запиту EXTRACT виведіть в окремому рядку його результат.

Приклади:

Клавіатура (stdin)	Екран (stdout)
ADD 192168812 ADD 125 ADD 321 EXTRACT EXTRACT CLEAR ADD 7 ADD 555 EXTRACT EXTRACT EXTRACT	192168812 321 555 7 CANNOT

Примітка. В кого проходять тести 1–8, але тест 9 дає вердикт «Неправильна відповідь» — у першу чергу слід перевірити, чи підтримує Ваша реалізація одночасне перебування у структурі кількох елементів з однаковим значенням (повинна підтримувати, в тому смислі, що скільки штук поклали, стільки й повинно зберігатися, а якщо до них дійде черга, то стільки штук і вийматися).

Задача 0.0:58. «ads058»

Напишіть програму, яка міститиме реалізацію структури даних для сукупності неперетинних підмножин (disjoint sets) і оброблятиме послідовність запитів таких виглядів:

`RESET` *n* — створити нову серію підмножин: множина з самого лише елемента 0, з самого лише елемента 1, і так до множини з самого лише елемента *n*–1 включно. Якщо структура вже містила якусь іншу сукупність неперетинних підмножин, уся відповідна інформація втрачається. При цьому слід вивести два слова через пробіл "`RESET DONE`".

`JOIN` *j k* — з'єднати підмножини, яким належать елемент *j* та елемент *k*. Якщо елементи і так належали одній підмножині, вивести слово "`ALREADY`", після нього через пробіли ті самі числа *j* та *k* в тому самому поряку. Якщо елементи досі належали різним підмножинам, то дія відбувається лише зі структурою даних у пам'яті, нічого не виводиться.

`CHECK` *j k* — перевірити, одній чи різним підмножинам у даний момент належать елемент *j* та елемент *k*; вивести слово "`YES`" (якщо одній) або слово "`NO`" (якщо різним).

Вхідні дані. У вхідних даних записано послідовність запитів RESET, JOIN та CHECK — кожен у окремому рядку, згідно вищеописаного формату. Гарантовано, що перший рядок містить запит RESET, а загальна кількість запитів RESET невелика. Також гарантовано, що в кожному запиті JOIN та в кожному запиті CHECK обидва числа будуть в діапазоні від 0 до *n*–1, де *n* — параметр останнього виконаного запиту RESET.

Результати. Для запитів CHECK та тих запитів JOIN, де елементи і так належать одній підмножині, виведіть результати. Результат кожного такого запиту виводити в окремому рядку, слова писати великими латинськими буквами без лапок.

Приклади:

Клавіатура (stdin)	Екран (stdout)
RESET 15 JOIN 14 10 JOIN 13 8 JOIN 0 9 JOIN 8 3 JOIN 4 1 JOIN 10 5 JOIN 8 4 CHECK 2 11 JOIN 4 1 JOIN 2 6 CHECK 9 1 JOIN 6 5 CHECK 10 5	RESET DONE NO ALREADY 4 1 NO YES

Примітка.

Відповіді «NO» даються на запити «CHECK 2 11» та «CHECK 9 1», відповідь «ALREADY 4 1» — на «JOIN 4 1» (10-й запит), «YES» — на «CHECK 10 5».

Реалізувати структуру даних disjoint sets слід самостійно (наскільки відомо автору задачі, у стандартних бібліотеках поширених мов програмування її все одно нема).

Тести та обмеження підбиралися так, щоб проходила реалізація, що містить оптимізацію «при з'єднаннях, слід приєднувати множину меншого рангу до множини більшого рангу, а не навпаки», але не містить жодної іншої оптимізації. Чи будуть проходити реалізації, де замість цієї оптимізації вжита деяка інша — невідомо (як вийде, так і буде). Реалізації, де нема ні вищезгаданої оптимізації, ні якоїсь альтернативної, по ідеї повинні не проходити.

Задача 0.0:59. «ads059»

Напишіть програму, яка знаходитиме ОДМВ (остовне дерево мінімальної ваги) неорієнтованого зваженого графу з додатними довжинами ребер. Програма повинна працювати швидко для великих розріджених графів (максимальна кількість вершин — десятки тисяч, ребер — сотні тисяч).

Вхідні дані. Перший рядок містить два числа *N* та *M*, розділені пробілом — кількість вершин та кількість ребер графа. Далі йдуть *M* рядків, кожен з яких містить по три цілі числа, розділені пробілами. Перші два з них різні, у межах від 0 до *N*–1 кожне, і позначають кінці відповідного ребр*а*, третє — у межах від 1 до 10^9 і позначає довжину цього ребр*а*. Гарантовано, що всі ребра мають різні довжини.

Результати. Виведіть єдиний рядок, який має містити: якщо граф незв'язний — єдине словосполучення "NON-CONNECTED" (великими латинськими буквами, без лапок), якщо зв'язний — сумарну довжину ребер ОДМВ.

Приклади:

Клавіатура (stdin)	Екран (stdout)
5 7 1 2 5 1 3 2 2 3 4 2 4 3 3 4 6 0 3 20 0 4 10	19
100 1 17 42 111	NON-CONNECTED

Примітка.

Зверніть увагу, що при вказаних обмеженнях на довжини та кількість ребер можлива ситуація, коли сума довжин ребер ОДМВ не поміщається у тип int. Щоб не відбулося ні переповнення, ні втрати точності, скористайтеся цілим 64-бітовим типом.

Алгоритму Краскала абсолютно не потрібно, щоб граф був поданий списками суміжності. Тому, на відміну від попередніх графових задач, формувати при читанні графа списки суміжності недоцільно. Натомість, потрібно сортувати ребра за довжиною, як у задачі «Сортування ребер за довжиною». І взагалі, $\approx 95\%$ розв'язку цієї задачі полягає в тому, щоб правильно поєднати задачі «Сортування ребер за довжиною» та «Система неперетинних множин (disjoint sets)».

Задача 0.0:60. «ads060»

Відрізняється від попередньої версії **виключно** *тим, що результати потрібно вивести більш детально. Якщо граф незв'язний, то, як і в попередній задачі, відповідь має бути просто рядком з єдиним словосполученням "NON-CONNECTED". А для зв'язного графу відповідь повинна описувати саме дерево.*

Напишіть програму, яка знаходитиме ОДМВ (остовне дерево мінімальної ваги) неорієнтованого зваженого графу з додатними довжинами ребер. Програма повинна працювати швидко для великих розріджених графів (максимальна кількість вершин — десятки тисяч, ребер — сотні тисяч).

Вхідні дані. Перший рядок містить два числа *N* та *M*, розділені пробілом — кількість вершин та кількість ребер графа. Далі йдуть *M* рядків, кожен з яких містить по три цілі числа, розділені пробілами. Перші два з них різні, у межах від 0 до *N*–1 кожне, і позначають кінці відповідного ребра, третє — у межах від 1 до 10^9 і позначає довжину цього ребра. Гарантовано, що всі ребра мають різні довжини.

Результати. Якщо граф незв'язний, виведіть єдиний рядок з єдиним словосполученням "NON-CONNECTED" (великими латинськими буквами, без лапок).

Якщо граф зв'язний, то перший рядок має містити єдине ціле число — сумарну довжину всіх ребер ОДМВ; далі мають бути *N* рядків, кожен з яких має містити список суміжності відповідної вершини. Формат списку суміжності: пишеться, яка вершина, пробіл, двокрапка, далі усі

вершини, куди йдуть ребра з даної, у вигляді: пробіл, номер вершини, куди йде ребро, відкривна дужка “(”, довжина ребра, закривна дужка “)”. Рядки мають бути виведені згідно порядку номерів вершин (тих що зліва від “ : ”). Переліки всередині кожного рядка теж повинні бути впорядковані за номерами вершин. Між кожними сусідніми різними ребрами одного переліку повинні бути кома і пробіл. Закінчувати рядок комою чи комою з пробілом (ставити їх після опису останнього в рядку ребра) заборонено.

Приклади:

Клавіатура (stdin)	Екран (stdout)
5 7 1 2 5 1 3 2 2 3 4 2 4 3 3 4 6 0 3 20 0 4 10	19 0 : 4(10) 1 : 3(2) 2 : 3(4), 4(3) 3 : 1(2), 2(4) 4 : 0(10), 2(3)
100 1 17 42 111	NON-CONNECTED

Примітка.

Див. примітки до попередньої задачі «Алгоритм Краскала» та до задачі «Алгоритм Дейкстри за M log N».