```
cin >> s;
if (s=="freeze") || (s=="heat" && troom>tcond) || (s=="freeze")
{
    cout << troom;
} else {
    cout <<tcond;
}
```

Второе решение.

Заметим, что при каждом режиме работы кондиционер реализует некоторую функцию, которая вычисляет результат по двум аргументам troom и tcond.

В режиме "freeze" кондиционер реализует функцию $\min(x, y)$, в режим "heat" – функцию $\max(x, y)$, в режиме "auto" – функцию $f(x, y) = y$ (возвращает второй аргумент), а в режиме "fan" – функцию $g(x, y) = x$ (возвращает первый аргумент).

Приведем фрагмент программы на языке C++, реализующий данную идею:

```
if (s == "freeze") cout << ((troom > tcond) ? tcond : troom);
if (s == "heat") cout << ((troom < tcond) ? tcond : troom);
if (s == "fan") cout << troom;
if (s == "auto") cout << tcond;
```

Задача 6. «Праздничный ужин»

Пусть некоторому программисту предожили p вариантов ужина, а следующему – q.

$$p = a_1 \times a_2 \times \ldots \times a_i \times \ldots \times a_k, \quad q = a_1 \times a_2 \times \ldots \times (a_i - 1) \times \ldots \times a_k, \quad p / q = a_i / (a_i - 1),$$

откуда

$$a_i = p / (p - q).$$

Таким образом, когда программисту предложили p вариантов ужина, одно из блюд предлагалось в $p / (p - q)$ вариантах.

Постепенно анализируя входные данные, мы будем получать новую информацию о количестве блюд того или иного вида.

Заведем два массива: в одном (start) будем хранить изначальное количество каждого из видов блюд (сначала массив будет пустой, а по мере получения новой информации мы будем его расширять). Во втором массиве (current) в элементе с тем же номером мы будем хранить текущее количество вариантов того же вида блюда (на самом деле решений может быть несколько: мы будем хранить количества, соответствующие одному из них).

Пусть в некоторый момент мы обнаружили, что было a_i вариантов некоторого блюда. Возможны две ситуации:

1) в массиве current уже есть число a_i. Тогда будем считать, что это то же самое блюдо, и уменьшим этот элемент массива current на 1.
2) в массиве current нет числа a_i. Это означает, что мы получили информацию о блюде, про которое ранее не было известно ничего. Добавим в массив start число a_i, а в массив current – число $a_i - 1$.

(Заметим, что если в пункте 1 мы бы создали новое блюдо, то в итоге могли бы получить количество блюд, большее, чем указано в условии.)

После обработки всех входных данных мы получим массив start, длина которого меньше или равна количеству блюд k. Если она равна k, то задача решена. Если длина меньше k, посмотрим на количество вариантов r, которое было предложено на выбор последнему программисту. В массиве current уже есть информация о количестве блюд

некоторых видов к этому моменту: разделим r на произведение элементов массива current и будем считать, что у нас есть еще один вид блюд с таким изначальным количеством.

Если же и теперь количество видов блюд меньше, чем требуется, добавим оставшиеся виды блюд с изначальным количеством, равным 1.

Например, если последнему программисту остался выбор из 4 комбинаций блюд, всего блюд было 5, и в массиве start лежат числа 2 и 3, будем считать, что изначально был выбор из 2-х первых блюд, 3-х вторых блюд, 4-х третьих блюд, 1-го четвертого и 1-го пятого.

Приведем пример программы на языке python3:

```
types = 0
start = []
current = []
for i in range(1, n):
    y = a[i - 1] // (a[i - 1] - a[i])
    for j in range(types):
        if current[j] == y:
            current[j] = y - 1
            break
    else:
        types += 1
        start.append(y)
        current.append(y - 1)
last = a[n - 1]
for value in cur:
    last //= value
print(" ".join(map(str, start)), end = " ", file = fout)
if len(start) < k:
    print(last, '1 ' * (k - 1 - types), file = fout)
```

Задача 7. «Космический кегельбан »

Переформулируем задачу на геометрическом языке. Дано несколько кругов радиуса r (кегли) и полоса ширины $2q$ (траектория шара). Требуется подсчитать, сколько окружностей имеют хотя бы одну общую точку с полосой.

Заменим окружности на точки, а полосу ширины $2q$ на полосу ширины $2(q + r)$. Тогда задача сведется к эквивалентной: сколько точек (центров кеглей) лежат в расширенной полосе, то есть находятся на расстоянии не более $(q + r)$ от прямой, по которой движется центр шара. Заметим, что поскольку по условию задачи каждая точка шара лежит изначально ниже любой точки каждой кегли, то можно рассматривать не часть полосы выше шара, а всю полосу: на ответ это не повлияет.

Запишем неравенством условие принадлежности точки полосе. Пусть A – начальное положение центра шара, вектор AB имеет координаты (v_x, v_y), P – центр некоторой кегли. Тогда расстояние от точки P до прямой AB равно $|[AP, AB]| / |AB|$ (квадратными скобками обозначено псевдоскалярное произведение векторов), а условие принадлежности точки полосе запишется так:

$$|[AP, AB]| / |AB| \le q + r.$$

Приведем теперь три решения исходной задачи, отличающиеся по времени работы.

Первое решение (асимптотическая сложность порядка n^2).

Проверим отдельно принадлежность центра $P(x_p, y_p)$ каждой кегли нашей полосе. Для этого запишем полученное выше неравенство в координатах. Пусть точки A и B имеют координаты (x, y) и $(x + v_x, y + v_y)$ соответственно (B – точка на прямой, по которой движется центр шара). Тогда неравенство примет вид:

$$|(x_P - x)\, v_y - (y_P - y)\, v_x)| \,/\, \sqrt{v_x^2 + v_y^2} \le q + r. \quad (1)$$

Домножим обе части части на знаменатель и возведем в квадрат:

$$((x_P - x)\, v_y - (y_P - y)\, v_x))^2 \le (v_x^2 + v_y^2)\, (q + r)^2.$$

Подставляя последовательно в неравенство центры всех кеглей, находим ответ.

Последнее условие требует вычислений только в целых числах, что с одной стороны избавляет нас от всех возможных проблем, связанных с точностью вычислений, но с другой может создать проблемы с переполнением: правая часть неравенства в ограничениях задачи может достигать $((10^{12})^2 + (10^{12})^2) * (10^9)^2 = 10^{32}$, что превосходит границу чисел, которые можно хранить даже в 64-битных целых типах.

Приведем фрагмент программы на языке python3 (где длина целого числа ограничена лишь объемом доступной памяти; на других языках можно воспользоваться длинной арифметикой, либо написать решение с вещественными числами):

```
result = 0
for yp in range(n): # yp = 0, 1, … , n - 1
    res_line = 0
    for xp in range(-i, i + 1, 2): # xp = -i, -i + 2, …, i - 2, i
        if abs(a * (i - y) - b * (j - x)) ** 2 <=
                                  (q + r) ** 2 * (a ** 2 + b ** 2):
            res_line += 1
        result += res_line
print(result, file = outfile)
```

Заметим, что общее количество кеглей равно $1 + 2 + \ldots + n = n(n + 1)/2 \sim n^2$, поэтому асимтотическая сложность приведенного алгоритма – порядка n^2. Такое решение набирает 40 баллов.

Второе решение (асимптотическая сложность порядка n).

Заметим, что для каждого горизонтального ряда кеглей достаточно найти самую левую, и самую правую кеглю, которые собьет шар. Для этого решим неравенство

$$((x_P - x)\, v_y - (y_P - y)\, v_x))^2 \le (v_x^2 + v_y^2)\, (q + r)^2$$

относительно x_p для каждого значения y_p. Это квадратичное неравенство, решением которого является отрезок с концами

$$left = k(y_p - y) - c + x, \quad right = k(y_p - y) + c + x,$$

где $k = v_x/v_y$, $c = (q + r)\, \sqrt{v_x^2 + v_y^2} \,/\, v_y$.

Теперь нам необходимо найти самый левый и самый правый центр кегли на этом отрезке, то есть самое левое/правое четное или нечетное целое число (четность зависит от четности y_p), а также ограничить координату по модулю числом y_p. Запишем соответствующий фрагмент программы на языке python3:

```
left = math.ceil(left) #округляем "вверх"
right = math.floor(right) #округляем "вниз"
if left % 2 != yp % 2:
    left += 1
if right % 2 != yp % 2:
    right -= 1
left = max(left, -yp)
```

```
right = min(right, yp)
```

Теперь количество сбитых кеглей в горизонтальном ряду легко вычислить по формуле max((*right* – *left*) // 2 + 1, 0) (в процессе вычисления left и right может оказаться, что right < left: в этом случае шар не сбивает ни одной кегли в данном ряду).

Третье решение (целочисленное; ассимптотическая сложность n log n**).**

Преимущество этого решения в том, что оно, оставаясь достаточно быстрым, оперирует лишь целыми числами, а значит, лишено возможных проблем, возникающих из-за ошибок, связанных с округлением вещественных чисел при вычислениях.

Перепишем неравенство (1), избавившись от модуля:

$$-(q+r)\sqrt{v_x^2+v_y^2} \le ((x_P-x)\,v_y-(y_P-y)\,v_x) \le (q+r)\sqrt{v_x^2+v_y^2}.$$

Заметим, что при движении по горизонтальному ряду кеглей слева направо средняя часть неравенства возрастает (x_p возрастает, v_y = const > 0). Следовательно, искать целочисленные решения этого неравенства можно бинарным поиском: найдем самое левое и самое правое целое число, принадлежащее отрезку, и прибавим/вычтем 1, если число окажется ненадлежащей четности.

Мы пока еше не достигли цели, поскольку в неравенстве присутствует нецелое число – квадратный корень, а при возведении в квадрат средняя часть неравенства перестает быть возрастающей функцией. Для решения этой проблемы воспользуемся таким трюком. Перепишем условие

$$(x_P-x)\,v_y-(y_P-y)\,v_x) \le (q+r)\sqrt{v_x^2+v_y^2}$$

эквивалентным образом (пользуясь тем, что $q + r > 0$):

$$(x_P-x)\,v_y-(y_P-y)\,v_x) < 0 \quad \text{or} \quad ((x_P-x)\,v_y-(y_P-y)\,v_x))^2 \le (v_x^2+v_y^2)(q+r)^2.$$

Для проверки этого условия требуются уже только вычисления в целых числах, а значит, цель достигнута. Аналогично переписывается и левая часть двойного неравенства.

Опять же заметим, что вычисления нельзя провести в 64-битном целом типе из-за больших ограничений в условии задачи.

Задача 8. «Abracadabra»

Преобразуем данный нам словарь в такой вид, чтобы в нем можно было быстро искать слово с нужным супрефиксом. Для этого каждую строку $s[0..n-1]$ из словаря преобразуем в такую:

$$\underline{s[0]}\ s[n-1]\ \underline{s[1]}\ s[n-2]\ \underline{s[2]}\ s[n\text{-}3]\ \ldots\ \underline{s[n-1]}\ s[0].$$

Например, слово 'table' превратится в такое: 'tealbblaet'.

Отсортируем обновленный словарь в лексикографическом порядке (по алфавиту). Заметим, что теперь все слова с одинаковым супрефиксом идут подряд.

С каждым из образцов проделаем такое же преобразование. Теперь для каждого обновленного образца *s* найдем бинарным поиском самую первую позицию (left) в словаре, куда его можно вставить, не нарушая лексико-графический порядок. Возьмем строку, полученную из *s* заменой последнего символа на следующий в таблице ASCII:

```
s[n] = chr(ord(s[n] + 1)),
```

и найдем самую левую позицию в словаре, куда его можно вставить (right). Заметим, что все слова, имеющие супрефикс *s*, лежат в промежутке [left, right), поэтому искомое количество слов равно right – left.

Продемонстрируем сказанное на примере. Пусть словарь после обработки имеет вид:

s[0] = 'aa';