**A. Многоквартирный дом**

|  |  |
| --- | --- |
| Ограничение времени | 1 секунда |
| Ограничение памяти | 256 Мб |
| Ввод | стандартный ввод |
| Вывод | стандартный вывод |

Аркадий возвращался домой поздним вечером и захотел узнать, в скольких квартирах в его доме хозяева всё ещё не спят. Он решил выяснить это, посмотрев, в каких окнах горит свет.

Дом, в котором живёт Аркадий, состоит из n этажей, на каждом этаже которого расположено m квартир. Каждая квартира имеет x окон в высоту и y окон в ширину, то есть в каждой квартире всего x⋅y окон.

Аркадий определил для себя правило: если в квартире горит свет хотя бы в ⌈x⋅y/2⌉ (половине с округлением вверх) окнах, то это значит, что хозяева не спят. Ваша задача — узнать количество таких «бодрствующих» квартир в доме.

Формат ввода

В первой строке входных данных записаны четыре целых числа n, m, x, y (1≤n,m,x,y≤50) — количество этажей, квартир на этаже, окон в высоту и ширину в каждой квартире.

Следующие n⋅x строк длины m⋅y описывают состояние окон в доме. Если в окне горит свет, то оно описывается символом «X», иначе «0».

Формат вывода

Выведите единственное целое число — количество квартир в доме, в которых хозяева всё ещё не спят.

Пример 1

| **Ввод**  Скопировать ввод | **Вывод**  Скопировать вывод |
| --- | --- |
| 2 2 2 2  X000  000X  X000  XX00 | 1 |

Пример 2

| **Ввод**  Скопировать ввод | **Вывод**  Скопировать вывод |
| --- | --- |
| 3 1 2 1  0  0  X  X  0  X | 2 |

**B. Скалярное произведение**

|  |  |
| --- | --- |
| Ограничение времени | 1 секунда |
| Ограничение памяти | 256 Мб |
| Ввод | стандартный ввод |
| Вывод | стандартный вывод |

Скалярное произведение двух векторов одинакового размера — это сумма произведений соответствующих координат.

В поиске одним из шагов генерации выдачи является расчет скалярного произведения двух целочисленных векторов Q и D. Первый вектор вычисляется, исходя из запроса, а второй — исходя из документа. Каждая координата векторов — беззнаковое целочисленное 4-байтовое число. Размер обоих векторов N.

Документные вектора хранятся в поисковом индексе. В целях экономии места их координаты сжимаются до одного байта по следующему алгоритму:

* A=miniDi​
* B=maxiDi​
* Ci={⌊255(Di−A)/B−A⌋, если A<B 0, иначе}

Здесь ⌊x⌋ обозначает округление вниз до ближайшего целого числа (⌊5⌋=5, ⌊6.99⌋=6).

Таким образом, вектор D преобразится в вектор C того же размера, но каждая координата которого будет принимать целые значения от 0 до 255.

Например, если исходный вектор D = (1000, 2000, 3000, 4000, 5000), то A = 1000, B = 5000, и C = (0, 63, 127, 191, 255).

Ваша задача — посчитать скалярное произведение векторов Q и D, но имея на руках только сжатый вектор C (а так же значения A и B) вместо исходного вектора D.

Формат ввода

В первой строчке указан размер векторов N (1 ≤ N ≤ 64).

Во второй строчке через пробел указаны целочисленные координаты исходного запросного вектора Q (0 ≤ Qi ≤ 108).

В третьей строчке через пробел указаны целочисленные координаты сжатого документного вектора C (0 ≤ Ci ≤ 255).

В четвертой строчке через пробел указаны целые значения A и B (0 ≤ A ≤ B ≤ 108).

Формат вывода

В качестве ответа надо вывести единственное целое число — скалярное произведение исходных векторов. Ответ будет считаться корректным, если ∣dot(Q,D) − ans∣ ≤ ∑Q ∗ (B−A)/255​, где dot(Q,D) — истинное скалярное произведение исходных векторов, а ans — ответ участника.

Пример

| **Ввод**  Скопировать ввод | **Вывод**  Скопировать вывод |
| --- | --- |
| 5  1000 2000 3000 4000 5000  0 63 127 191 255  1000 5000 | 55000000 |

Примечания

В тесте из сэмпла Q = D = (1000, 2000, 3000, 4000, 5000). Их скалярное произведение равно 55000000. Допустимая ошибка будет равна (1000 + 2000 + 3000 + 4000 + 5000) ∗ (5000 − 1000)/255 = 235294.117... Соответственно любой целочисленный ответ из промежутка [54764706;55235294] будет засчитан за корректный.

**C. ЕТА**

|  |  |
| --- | --- |
| Ограничение времени | 1 секунда |
| Ограничение памяти | 64 Мб |
| Ввод | стандартный ввод или input.txt |
| Вывод | стандартный вывод или output.txt |

При вызове такси мы не всегда можем сказать точно, за сколько времени таксист может доехать до пользователя. В этой задаче вам нужно разработать алгоритм, который будет определять, какие таксисты смогут **гарантированно** добраться до точки заказа за отведенное время, если примут его моментально. Таксист может стоять на месте или перемещаться по кругу **по часовой стрелке**, поэтому на момент вызова его позиция может измениться по сравнению с последней известной — он мог двигаться с любой скоростью, вплоть до максимально разрешенной. Точка заказа и точки, в которых находятся или могут находиться таксисты, — это точки на круге, перемещаться на вызов такси может только по часовой стрелке.

Формат ввода

В первой строке задано три целых числа: число событий N, длина круга L (в метрах), по которому перемещаются таксисты и на котором происходят заказы, максимальная скорость S с которой такси могут перемещаться по кругу.

В последующих N строках описываются события в следующем формате:

* TAXI <timestamp> <taxi\_id> <taxi\_position>. Команда, оповещающая о том, что таксист с номером taxi\_id в момент времени timestamp прислал координату своего местоположения. Местоположение определяется значением taxi\_position — целым числом в диапазоне [0,L), где 0 означает, что такси стоит в начале круга, а любое положительное число — дальность позиции таксиста от начала круга при движении по часовой стрелке.
* ORDER <timestamp> <order\_id> <order\_position> <order\_time>. Команда, оповещающая о том, что в момент времени timestamp пришёл заказ order\_id на точку А, определяющуюся значением order\_position — целым числом в диапазоне [0,L) (где 0 означает, что заказ произошел в начале круга, а любое положительное число — дальность позиции вызова от начала круга при движении по часовой стрелке), до которого нужно **гарантированно** доехать за order\_time секунд.

Поле timestamp — это unix-like секундный таймстемп, все приходящие команды отсортированы по нему в порядке возрастания. Возрастание нестрогое: в нескольких последовательных командах могут быть одинаковые таймстемпы, но на момент события вызова мы рассматриваем только те события, которые нам известны (то есть не рассматриваем события, которые пришли после вызова, даже если у них такой же таймстемп).

Все заказы пронумерованы (order\_id), начиная с 0 в порядке их появления.

Ограничения:

* N≤5000;
* 0<L≤100000 (в метрах);
* 0<S≤10 (в метрах в секунду);
* 0≤order\_time≤L (в секундах);
* 0≤taxi\_id<N (число от 0 до N).

Формат вывода

Программа должна вывести K строк, где K — количество команд ORDER.

В ответ на каждую команду ORDER программа должна вывести строку с таксистами (набор taxi\_id, разделённых пробелами), которые гарантированно смогут добраться до точки А не более чем за order\_time. Если таких таксистов больше 5, то нужно вывести **5 любых**. Если ни один таксист не может гарантировано приехать в отведённое время, то в строке должен быть единственный идентификатор −1. Идентификаторы в одной строке не должны повторяться.

Пример

| **Ввод**  Скопировать ввод | **Вывод**  Скопировать вывод |
| --- | --- |
| 3 100 1  TAXI 1738300000 0 0  TAXI 1738300000 1 2  ORDER 1738300001 0 1 1 | 0 |

Примечания

Примечание к примеру.

Возможные позиции такси с id 0: [0,1]. В таком случае на момент заказа таксист может добраться до него за 0 или 1 секунду, то есть он гарантированно может добраться за требуемую 1 секунду до точки вызова.

Возможные позиции такси с id 1: [2,3]. То есть таксист может добраться до точки вызова не меньше чем за 98 секунд, поэтому он нам не подходит.

**D. ЖК Плюс Сити**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Язык | Ограничение времени | Ограничение памяти | Ввод | Вывод |
| Все языки | 3 секунды | 512 Мб | стандартный ввод | стандартный вывод |
| Kotlin 2.0.10 (JRE 21) | 4 секунды | 512 Мб |
| Scala 2.13.4 | 4 секунды | 512 Мб |
| Java 21 (Temurin JDK) | 4 секунды | 1024 Мб |
| Python 3.12.3 | 5 секунд | 512 Мб |

Дана прямоугольная карта местности в виде матрицы из n строк и m столбцов. На части клеток расположены жилые объекты. Была запланирована постройка ЖК Плюс Сити, состоящего из одного квадратного дома. Стороны квадрата должны быть параллельны строкам/столбцам матрицы.

Он должен быть отдалён хотя бы на d клеток от ближайшего жилого объекта, чтобы окружить его парковыми зонами. Расстояние считается по манхэттенской метрике. Из подходящих вариантов нужно выбрать дом с наибольшей площадью.

Учтите, что жилые объекты, уже находящиеся на карте, необязательно являются квадратами.

Формат ввода

В первой строке входных данных даны три целых числа — n,m,d (1⩽n,m⩽5000, 2⩽d⩽n+m−2) — размеры предоставленной карты местности и минимальное расстояние от ЖК до других жилых объектов.

Далее следует матрица из n строк, m столбцов, состоящая из символов «X», «O» (для застроенных и свободных клеток соответственно).

Формат вывода

Выведите одно целое число — наибольший размер стороны квадрата ЖК Плюс Сити, подходящий под данную карту местности. В случае невозможности разместить ЖК, выведите 0.

Пример 1

| **Ввод**  Скопировать ввод | **Вывод**  Скопировать вывод |
| --- | --- |
| 10 7 2  ooxoooo  ooxoooo  xoooooo  oooooox  oooxoxx  oooooox  oooxxoo  ooooooo  ooooooo  oooooox | 3 |

Пример 2

| **Ввод**  Скопировать ввод | **Вывод**  Скопировать вывод |
| --- | --- |
| 5 5 2  xxxxx  xxxxx  xxxxx  xxxxx  xxxxx | 0 |

**E. Финикийские сообщения**

|  |  |
| --- | --- |
| Ограничение времени | 4 секунды |
| Ограничение памяти | 512 Мб |
| Ввод | стандартный ввод или input.txt |
| Вывод | стандартный вывод или output.txt |

Учёные нашли остатки сообщений от древней цивилизации финикийцев — создателей одного из первых алфавитов, который использовали для торгов, дипломатии, в религиозных текстах. Финикийцы были морской цивилизацией, активно торговавшей по всему Средиземноморью. Для передачи сообщений между кораблями или поселениями они использовали систему из девяти барабанов, где количество ударов по конкретному барабану определяло букву сообщения, — похожая система добралась до наших дней в кнопочных телефонах с режимом ввода Т9.

| Барабан | Буквы |
| --- | --- |
| 1 | - (не используется) |
| 2 | ABC |
| 3 | DEF |
| 4 | GHI |
| 5 | JKL |
| 6 | MNO |
| 7 | PQRS |
| 8 | TUV |
| 9 | WXYZ |

Например, чтобы набрать U, финикийцы дважды ударяли по барабану номер 8, а чтобы набрать Z, они четырежды били по барабану с номером 9. Сообщение HELLOWORLD от них выглядит так: 443355555566696667775553.

Известно, что финикийцы используют только слова из словаря. Словом является последовательность буквенных символов без пробелов и иных знаков. Сообщение, которое нашли учёные, как раз связано с ритмами барабанов. Ваша задача — при помощи словаря расшифровать сообщение, переданное финикийцами.

Формат ввода

В первой строке задано сообщение s, которое необходимо расшифровать. Строка s состоит из цифр 22 — 99, при этом 1≤∣s∣≤128⋅10^3.

Во второй строке задано натуральное число n (1≤n≤10^3) — количество слов в словаре. В каждой из последующих n строк указано слово из словаря — di​. Гарантируется, что длина каждого слова di​ не превышает 256, а сумма длин всех слов словаря не превышает 64⋅10^3.

Формат вывода

Используя заглавные буквы латинского алфавита и пробелы, выведите расшифрованное сообщение. Гарантируется, что существует хотя бы один способ расшифровать сообщение с использованием слов из словаря. В случае если существует несколько способов расшифровать сообщение, выведите любой из них.

Пример 1

| **Ввод**  Скопировать ввод | **Вывод**  Скопировать вывод |
| --- | --- |
| 443355555566696667775553  3  WORLD  QUANTUM  HELLO | HELLO WORLD |

Пример 2

| **Ввод**  Скопировать ввод | **Вывод**  Скопировать вывод |
| --- | --- |
| 7788266888674499977774442227777  5  PHYSICS  QUANTUM  WORLD  HELLO  PHYS | QUANTUM PHYSICS |