Кто мяукает?!

Краткое условие

По условию задачи, дан набор высказываний одного из четырёх видов:

- `A: B is x!`;
- `A: B is not x!`;
- `A: I am x!`;
- `A: I am not x!`.

Где «`A`» и «`B`» могут различаться между высказываниями и означают имена людей, а «`x`» остаётся одинаковым между высказываниями и означает какое-то действие.
По этим высказываниям надо составить «рейтинговую таблицу», руководствуясь следующими правилами:

1. Высказывание вида «`A: B is x!`» прибавляет 1 очко к счёту B.
2. Высказывание вида «`A: B is not x!`» отнимает 1 очко со счёта B.
3. Высказывание вида «`A: I am x!`» прибавляет 2 очка к счёту A.
4. Высказывание вида «`A: I am not x!`» отнимает 1 очко со счёта A.

В ответ нужно вывести имена тех, кто набрал наибольшее количество очков, в формате «`A is x.`», а если таких несколько, то отсортировать в алфавитном порядке.

Решение с помощью регулярных выражений

Для успешного решения достаточно сделать то, что описано в условии, однако нужно учитывать тонкости разбора высказываний.

Нужно грамотно определять в каждом высказывании говорящего («A»), про кого он говорит («B» или «I»), глагол («is» или «am») и возможное отрицание («not»). Один из вариантов — использовать регулярные выражения, чтобы определять нужные шаблоны, однако в таком случае особое внимание нужно уделить оптимизации, чтобы не превысить ограничение по времени.

Регулярные выражения от одного из участников:

```
var (
  selfPositive  = regexp.MustCompile(`^([A-Z][a-z]*): I am ([a-z]+)!$`)
  selfNegative  = regexp.MustCompile(`^([A-Z][a-z]*): I am not ([a-z]+)!$`)
  otherPositive = regexp.MustCompile(`^([A-Z][a-z]*): ([A-Z][a-z]*) is ([a-z]+)!$`)
  otherNegative = regexp.MustCompile(`^([A-Z][a-z]*): ([A-Z][a-z]*) is not ([a-z]+)!$`)
)
```

Проверяя на каждое из них, можно не только определить, что высказывание удовлетворяет какому-то из четырёх шаблонов, но и найти нужные части (говорящий, субъект, глагол и действие) с помощью механизма групп (в регулярных выражениях группы обозначаются круглыми скобками):

```
func parseStatement(line string) Statement {
  if m := selfPositive.FindStringSubmatch(line); m != nil {
    return Statement{Speaker: m[1], Subject: m[1], IsAboutSelf: true, IsPositive: true, Action: m[2]}
  }
  if m := selfNegative.FindStringSubmatch(line); m != nil {
    return Statement{Speaker: m[1], Subject: m[1], IsAboutSelf: true, IsPositive: false, Action: m[2]}
  }
  if m := otherPositive.FindStringSubmatch(line); m != nil {
    return Statement{Speaker: m[1], Subject: m[2], IsAboutSelf:
```

```
    false, IsPositive: true, Action: m[3]}
  }
  if m := otherNegative.FindStringSubmatch(line); m != nil {
    return Statement{Speaker: m[1], Subject: m[2], IsAboutSelf:
    false, IsPositive: false, Action: m[3]}
  }

  return Statement{}
}
```

Решение с помощью ручного разбора строк

Альтернативный подход — это ручной разбор таких выражений. Достаточно разбить строку по пробелам, и мы получим такие массивы:

- `['A:', 'B', 'is', 'x!']`
- `['A:', 'B', 'is', 'not', 'x!']`
- `['A:', 'I', 'am', 'x!']`
- `['A:', 'I', 'am', 'not', 'x!']`

Учитывая, что ни «`A`», ни «`B`», ни «`x`» не могут содержать пробелов, можно заметить несколько закономерностей:

1. Каждый массив состоит не меньше чем из 4 элементов.
2. Каждый массив начинается с имени говорящего («`A`») *и двоеточия*.
3. Каждый массив заканчивается действием («`x`») *и восклицательным знаком*.
4. На третьей позиции каждого массива находится либо «`is`», либо «`am`».
5. На четвёртой позиции каждого массива находится либо действие («`x!`»), либо отрицание («`not`»).

Обратите внимание, что в этом списке не упоминается имя человека, про которого говорят (назовём его субъектом), так как возможно как высказывание «`A: I am x!`», так и высказывание «`A: I is x!`», то есть само по себе значение субъекта ни на что не влияет. Однако, если глагол равен «`am`», то субъект по условию всегда равен «`I`».

Исходя из этого, можно построить алгоритм определения, кто должен получить сколько очков на основе очередного высказывания.

Заведём словарь, где ключом будет имя человека, а значением — количество его очков. Количество очков по итогу может стать отрицательным. **Очень важно**, чтобы изначально у всех субъектов *и* говорящих в словаре было сохранено 0 очков, так как в таком тесте:

```
1
A: B is not x!
```

У человека «`B`» станет -1 очко (минус 1), а у человека «`A`» — 0 очков. Соответственно, ответ будет «`A is x.`»

После инициализации такого словаря, пройдёмся по всем высказываниям и начнём разбор:

1. У элемента на первой позиции уберём один символ справа (отрезаем двоеточие).
2. Если субъект (элемент на второй позиции) — это «`I`», а глагол (элемент на третьей позиции) — это «`am`»:
   (a) Если есть частица отрицания (элемент на четвёртой позиции равен «`not`»): отнимаем у говорящего (элемент на первой позиции без двоеточия на конце) одно очко.
   (b) Иначе, если нет частицы отрицания — прибавляем говорящему два очка.

3. Иначе, если есть частица отрицания — отнимаем у субъекта одно очко.

4. Иначе — прибавляем субъекту одно очко.

```
switch {
case subject == "I" && verb == "am" && isNot:
    scores[speaker]--
case subject == "I" && verb == "am" && !isNot:
    scores[speaker] += 2
case isNot:
    scores[subject]--
default:
    scores[subject]++
}
```

При этом нужно хотя бы один раз сохранить действие (последний элемент массива), убрав у него последний символ (восклицательный знак), так как он понадобится для формирования ответа.

Остаётся только найти в сформированном словаре все ключи с максимальным значением. Однако здесь нужно помнить, что максимальное значение может быть отрицательным, например здесь:

```
4
A: B is not x!
B: C is not x!
C: A is not x!
C: I am not x!
```

У «A», «B» и «C» отрицательное количество очков, но у «C» -2 (минус 2) очка, а у остальных по -1 (минус одному) очку. Поэтому здесь ответ:

```
A is x.
B is x.
```

K-уменьшение

Краткое условие

По условию задачи дан массив целых неотрицательных чисел $a[0..N-1]$. Можно выполнить операцию любое число раз:

- выбрать подмассив длины K, начиная с индекса «`i`», то есть $[a_i, a_{i+1}, ..., a_{i+K-1}]$, только если все элементы > 0;
- после этого операция уменьшает каждый элемент подмассива на 1.

Задача: можно ли после некоторого количества операций сделать все элементы массива равными нулю?

Размышления

1. Одна операция уменьшает K подряд идущих положительных элементов на 1.
2. Мы можем применять её к любому подмассиву длины K, пока все его элементы положительные.
3. Каждый элемент a_i должен быть уменьшен до 0.

Решение

Задачу решаем с помощью жадного алгоритма. Если на позиции «`i`» у нас $a_i > 0$, то:

1. Единственный способ уменьшить a_i — это применить операции, начинающиеся не позже «`i`» и охватывающие «`i`».
2. Самая ближайшая такая операция — это операция, начинающаяся в «`i`» и действующая на $[i, i+K-1]$.

Если $a_i > 0$, то:

1. Мы обязаны применить a_i операций с началом в позиции «`i`», иначе a_i никогда не обнулится. **Но!**
2. Если $i + K > N$, то операция не помещается в массив — значит, обнулить невозможно.

Проблема производительности

Допустим, $a_i = 1\,000\,000\,000$ и таких элементов много. Если каждый раз будем уменьшать вручную K элементов — получим медленный алгоритм (до 10^9 операций). Поэтому вместо реального вычитания используем технику разностного массива:

- Заводим массив `delta[0..N]`, где `delta[i]` — изменение количества вычитаний, начинающееся в позиции «`i`».
- При проходе:
  - `cur_sub`: сколько вычитаний накопилось на текущей позиции («`i`») от предыдущих операций.

Алгоритм

1. `cur_sub` $= 0$.
2. Для «`i`» от 0 до $N - 1$:
   (a) `cur_sub += delta[i]`
   (b) если `a[i] - cur_sub < 0`: → вычли больше, чем нужно → ошибка → `NO`
   (c) если `[i] - cur_sub > 0`:
   - `d = a[i] - cur_sub` #сколько ещё нужно вычесть
   - если $i + K > N$: → операция не поместится → `NO`
   - применяем d операций: `cur_sub += d`

- `delta[i + K] -= d` #отменим влияние d через K шагов.

Пример с пояснениями

Пример: N = 5, K = 3, a = [2, 3, 6, 4, 3].

Подготовка

- `delta := make([]int, N+K+1)` → `delta[0..7]`, все нули;
- `curSub := 0` — текущая сумма всех вычитаний, действующая на `a[i]`.

Решение

1. i = 0:
   - `curSub += delta[0]` → 0
   - `d = a[0] - curSub` = 2 → нужно вычесть ещё 2
   - Применяем операции для подмассива $[0, 1, 2]$
     - `curSub += 2` → 2
     - `delta[3] -= 2` → `delta` = $[0, 0, 0, -2, 0, 0, 0]$
2. i = 1:
   - `curSub += delta[1]` → `delta[1]` = 0 → `curSub` = 2
   - `d = a[1] - curSub` = $3 - 2 = 1$ → нужно вычесть ещё 1
   - Применяем операции для подмассива $[1, 2, 3]$
     - `curSub += 1` → 3
     - `delta[4] -= 1` → `delta` = $[0, 0, 0, -2, -1, 0, 0]$
3. i = 2:
   - `curSub += delta[2]` → `delta[1]` = 0 → `curSub` = 3

- `a[2]` - `curSub` $= 6 - 3 = 3 \rightarrow$ нужно вычесть ещё 3
- Применяем операции для подмассива $[2, 3, 4]$
  - `curSub += 3` $\rightarrow 6$
  - `delta[5] -= 3` $\rightarrow$ `delta` $= [0, 0, 0, -2, -1, -3, 0]$

4. i $= 3$:
   - `curSub += delta[3]` $= -2 \rightarrow$ `curSub` $= 6 - 2 = 4$
   - `d = a[3] - curSub` $= 4 - 4 = 0 \rightarrow$ всё обнулено, ничего делать не нужно

5. i $= 4$:
   - `curSub += delta[4]` $= -1 \rightarrow$ `curSub` $= 4 - 1 = 3$
   - `a[4] - curSub` $= 3 - 3 = 0 \rightarrow$ всё обнулено, ничего делать не нужно

Финальный результат

- все `a[i]` обнулились без выхода за границы;
- ни один `a[i]` не стал меньше нуля;
- все применённые операции были допустимы.

Оригами

Краткое условие

Дана форма клетчатого листа бумаги — основы для оригами. Лист бумаги может быть не прямоугольным. Лист бумаги вводится как таблица из $n \times m$ символов. Каждая ячейка таблицы может содержать или не содержать участок листа. Гарантируется, что все символы таблицы равны «#» или «.».

Все остальные символы появляются в процессе складывания оригами. Лист может быть несвязным, также внутри листа могут быть «дырки».

Таким образом, по данной таблице можно восстановить исходный лист единственным образом. Но в процессе складываний листа бумаги могут понадобиться дополнительные символы для обозначения клеток, которые заполнены участком листа частично. В таблице представлен список символов и соответствующих им обозначений. В них красным цветом показано, как участок листа заполнил клетку, а белым — пустое пространство.

Символ	Клетка
.	
#	
^	 или 
>	 или 
v	 или 
<	 или 
/	 или 
\	 или 
x	или

С этим листом бумаги проводится некоторое количество операций

складывания. Для каждой операции выбирается прямая: горизонтальная, вертикальная или диагональная. Все участки бумаги, что находятся с правой стороны от прямой, отражаются относительно этой прямой и перекладываются на левую сторону от этой прямой. Прямая обозначается двумя точками. Участок бумаги находится по правую сторону от прямой, если он находится по правую сторону от луча, проведённого от первой из данных точек ко второй. Прямая может быть проведена по границе листа.

После каждой операции складывания необходимо вывести получившееся оригами.

В первой группе тестов во входных данных нет символов «.», то есть оригами — это прямоугольник. С ним производится ровно одна операция относительно некоторой вертикальной или горизональной, *но не диагональной* прямой.

Во второй группе тестов с оригами производятся операции относительно некоторых вертикальных или горизональных, *но не диагональных* прямых.

В третьей группе тестов дополнительных ограничений нет.

Разбор примера

Входные данные:

```
1
8 9 3
#########
#########
#########
#########
#########
#########
#########
#########
21 7
14 28
23 7
```

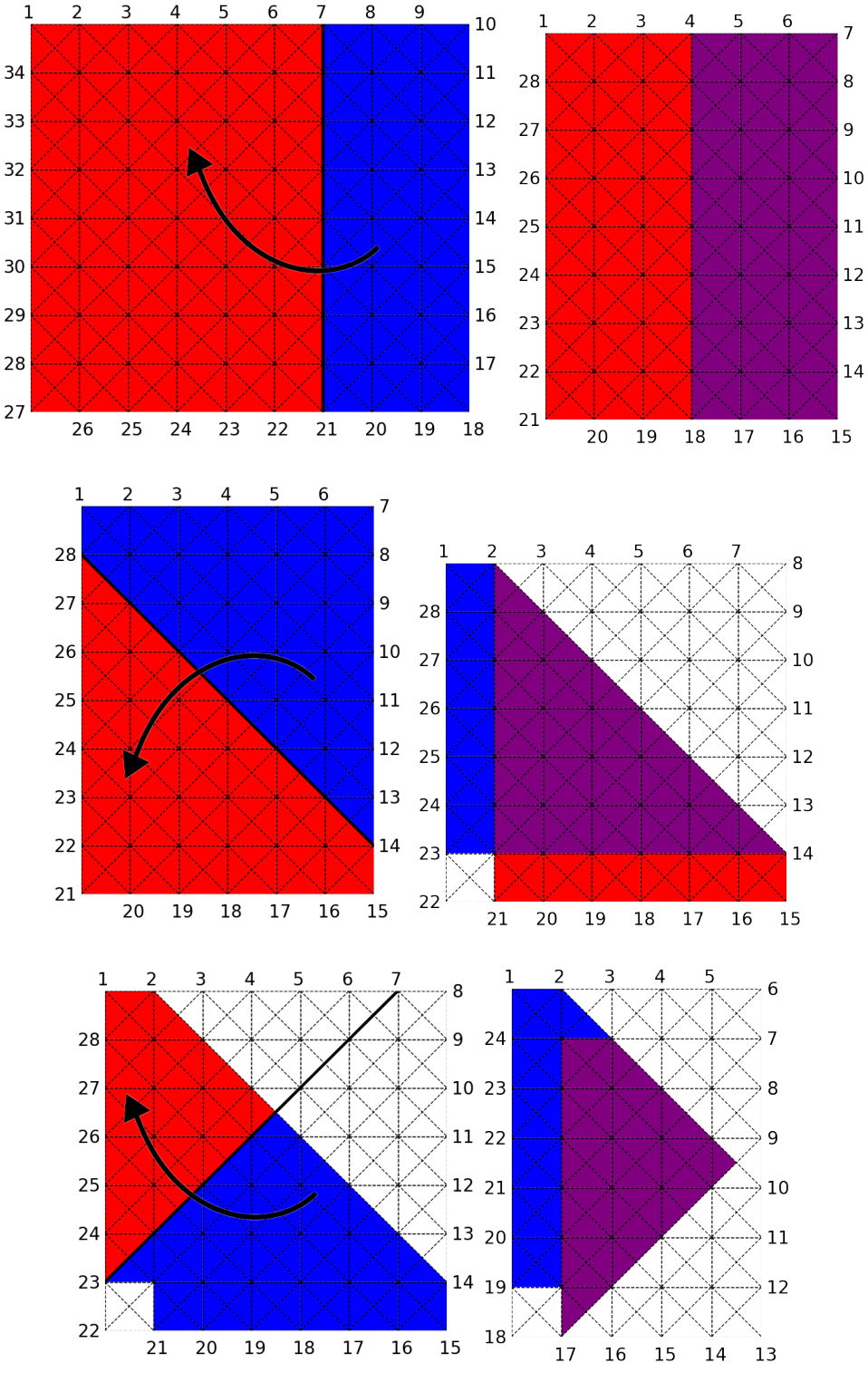
Выходные данные:

```
######
######
######
######
######
######
######
######

#\.....
##\....
###\...
####\..
#####\.
######\
.######

#\...
##\..
###\.
####>
###/.
##/..
./...
```


В каждой паре изображений левое показывает состояние оригами до операции, правое — после. Синим обозначена часть оригами, которая отражается относительно прямой, красным — часть, которая остаётся. На правых изображениях фиолетовым обозначена часть оригами, в которой синяя часть наложилась на красную.


1 2 3 4 5 6 7 8 9 10
34 11
33 12
32 13
31 14
30 15
29 16
28 17
27
26 25 24 23 22 21 20 19 18
1 2 3 4 5 6 7
28 8
27 9
26 10
25 11
24 12
23 13
22 14
21
20 19 18 17 16 15
1 2 3 4 5 6 7
28 8
27 9
26 10
25 11
24 12
23 13
22 14
21
20 19 18 17 16 15
1 2 3 4 5 6 7 8
28 9
27 10
26 11
25 12
24 13
23 14
22
21 20 19 18 17 16 15
1 2 3 4 5 6 7 8
28 9
27 10
26 11
25 12
24 13
23 14
22
21 20 19 18 17 16 15
1 2 3 4 5 6
24 7
23 8
22 9
21 10
20 11
19 12
18
17 16 15 14 13

Решение

Первая группа тестов

Разберём решения для каждой группы тестов отдельно.

В первой группе тестов оригами на входе имеет форму прямоугольника и все клетки листа заполнены. Нам нужно сделать только одну операцию относительно некоторой вертикальной или горизонтальной прямой.

Введём координатную систему, где точка $(0, 0)$ соответствует левому нижнему углу листа, а точка (N, M) — правому верхнему углу.

Давайте определим по номеру точек K_1 и K_2 их координаты (X_1, Y_1). Для удобства перейдём в 0-индексацию.

- Если $K < N$, то $X = K$, $Y = M$.
- Иначе если $K < N + M$, то $X = N$, $Y = M - (K - N)$.
- Иначе если $K < 2N + M$, то $X = N - (K - N - M)$, $Y = 0$.
- Иначе ($K < 2N + 2M$), то $X = 0$, $Y = K - 2N - M$.

По координатам точек мы можем определить, вертикальная это прямая или горизонтальная. Если $X_1 = X_2$, то это вертикальная прямая, иначе горизонтальная. Если операция вертикальная, то наше оригами уменьшится в ширину до $\max(X_1,\ N - X_1)$. Аналогично, если операция горизонтальная, то высота оригами уменьшится до $\max(Y_1,\ M - Y_1)$.

Вторая группа тестов

Во второй группе тестов не все клетки оригами заполнены. Нам нужно сделать несколько операций относительно некоторых вертикальных или горизонтальных прямых.

В этом случае нам нужно симулировать процесс складывания явно. Аналогично первой группе, определим прямую, относительно которой происходит операция.

Здесь нам будет удобно хранить заполненные клетки оригами в виде множества координат.

Если операция вертикальная и луч от первой точки до второй идёт вверх, все клетки с координатами $(x,\ y)$, где $x > X_1$, будут отражены относительно прямой $X = X_1$.

В таком случае клетка $(x,\ y)$ будет отражена в клетку $(2 \cdot X_1 - x,\ y)$. Все остальные случаи разбираются аналогично.

После каждой операции не забудьте пересчитать размер оригами

$$N = max(x) - min(x), M = max(y) - min(y)$$

и их координаты

$$(x, y) \to (x - min(x), y - min(y)).$$

Третья группа тестов

В третьей группе тестов у нас могут быть операции и относительно диагональных прямых.

Обратите внимание, что в этом случае клетки могут быть заполнены не полностью, то есть в каждой клетке может быть заполнен один или несколько треугольников.

Здесь нам будет удобно хранить заполненные клетки как множество треугольников. Для удобства умножим координаты на 2, чтобы все вершины треугольников имели целые координаты.

Таким образом, каждый треугльник можно представить как кординаты его вершины при прямом угле и направления (верх, низ, лево, право). Обратите внимание, что в таком случае мы легко можем восстановить все координаты вершин треугольника из его представления.

Например, треугольник с вершиной $(x,\ y)$ и направлением вверх будет иметь вершины $(x,\ y)$, $(x-1,\ y+1)$, $(x+1,\ y+1)$.

При операции относительно прямой, идущей из точки $(X_1,\ Y_1)$ в точку $(X_2,\ Y_2)$, мы можем определить, с какой стороны от этой прямой лежит треугольник.

Очевидно, что треугольник не может иметь вершины и слева, и справа от прямой.

Также очевидно, что хотя бы одна из трёх вершин треугольника не лежит на прямой.

Давайте найдем любую из трёх вершин треугольника, которая не лежит на прямой. Чтобы проверить, с какой стороны от луча $(X_1,\ Y_1) \to (\mathrm{X}_2,\ Y_2)$ лежит точка $(X,\ Y)$, мы можем использовать векторное произведение.

Если знак векторного произведения вектора $(X_2 - X_1,\ Y_2 - Y_1)$ и вектора $(X - X_1,\ Y - Y_1)$ положителен, то точка лежит слева от луча, иначе справа.

Если вы не хотите использовать векторное произведение, то можно разобрать все случаи отдельно.

Чтобы отразить треугольник относительно диагональной прямой, идущей из точки $(X_1,\ Y_1)$ в точку $(X_2,\ Y_2)$, таких что $X_1 + Y_1 = X_2 + Y_2$, координата его вершины $(x,\ y)$ будет преобразована в $(X_1 + Y_1 - y,\ X_1 + Y_1 - x)$.

Если же $X_1 - Y_1 = X_2 - Y_2$, то координата вершины $(x,\ y)$ будет преобразована в $(X_1 - Y_1 + y,\ X_1 - Y_1 + x)$.

Направление треугольника при отражении относительно прямой также меняется только в зависимости от его направления.

После операции не забудьте пересчитать размер оригами и координаты треугольников.

Чтобы вывести получившееся оригами, переберём каждую доступную координату клетки и, в зависимости от множества треугольников в этой клетке, выведем соответствующий символ.