

A. ANOTHER GAME

A. Another Game

- Аккуратно прочитать входные данные и выделить значимую часть, т.е. содержание тайлов без рамок
- Выбрать из 4-х поворотов тайла Васи уникальные
- Перебрать все позиции всех уникальных поворотов тайла внутри и на границе карты
- Проверить выполнение требований условия

B. BOMB HAS BEEN PLANTED

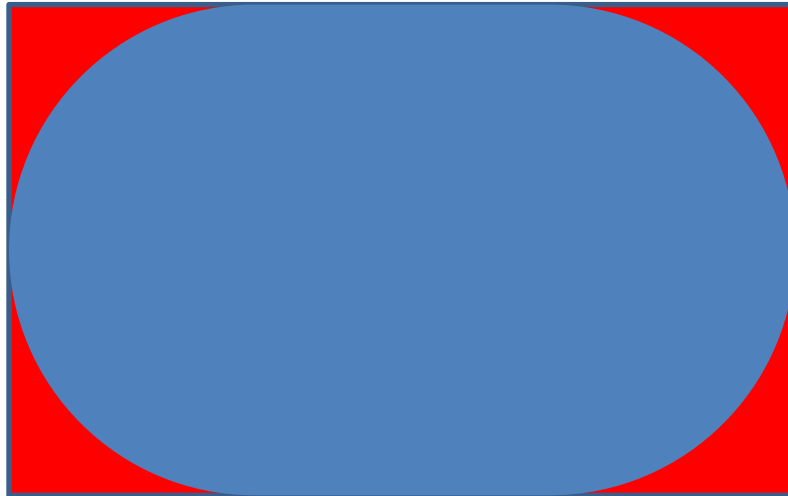
V. Bomb Has Been Planted

- Каким-либо алгоритмом (алгоритм Дейкстры, алгоритм Флойда, поиск в ширину) найти расстояния между вершинами (A,B), (A,C), (B,C)
- Для разминирования над либо сначала пройти из C в A, затем из A в B, либо наоборот
- Ответ на задачу:
$$(\text{Min}(D(A, C), D(B, C)) + D(A, B)) * 4 + 10$$

C. CSS IS AWESOME

C. CSS is Awesome

- Площадь исходного прямоугольника (без скругленных углов) равна $W \cdot H$
- Посчитаем площадь недостающих частей прямоугольника:



C. CSS is Awesome

- Площадь каждого такого «куска» квадрата для скругления радиуса R равна:

$$S = R^2 - \pi \cdot \frac{R^2}{4}$$

D. DECOMPRESSING

D. Decompressing

- По «общему правилу» и «исключениям» построить таблицу – значение красоты для каждой пары яркостей
- Сначала посчитаем значения $C[i][j][k]$ – максимальное значение красоты растянутого отрезка, если исходный пиксель имел яркость i , отрезок начинается с пикселя яркости j и заканчивается пикселем яркости k

D. Decompressing

- Значения таблицы $d1$ не зависят от исходной картинке
- Эти значения можно посчитать при помощи динамического программирования: для каждого значения k независимо заполним таблицу $d1[m][sum][p]$ – максимальная возможная красота, если у нас осталось m ($m \leq K$) позиций отрезка, суммарная яркость должна быть sum ($sum \leq K * 9$) и последний пиксел имеет яркости p ($p \leq 9$)

D. Decompressing

- Каждое значение таблицы можно пересчитать за 10 итераций, выбирая яркость следующего пикселя:

$$d1[m][sum][p] = \max(d1[m-1][sum-p1][p1] + \text{bonus}[p][p1])$$

Где $p1$ – яркость нового пикселя, а $\text{bonus}[p][p1]$ – бонус красоты за соседние пиксели p и $p1$

D. Decompressing

- Получаем $C[i][j][k] = d1[K-1][m*K-j][j]$ независимо для каждого k (не путать с K)
- Общая сложность вычисления таблицы $d1 - K*K*10*10*10 = K^2*1000$
- Теперь можно легко посчитать максимальный бонус красоты для данной текстуры
- Бонусы красоты для строк считаются независимо

D. Decompressing

- Заведем другую таблицу: $d2[m][p]$ – максимальный бонус, если осталось «растянуть» m пикселей и последний пиксель в растянутой текстуре имеет яркость p
- Для каждой строки посчитаем эту таблицу независимо при помощи динамического программирования

D. Decompressing

$$d2[m][p] = \max(d2[m+1][p1] + \\ \text{bonus}[p][p2] + \\ C[S[m]][p2][p1])$$

по всем $p1$ и $p2$ от 0 до 9

$S[m]$ – яркость пикселя, стоящего на месте m текущего ряда

D. Decompressing

- Тогда бонус красоты для текущей строки будет равен

$$\max(d2[1][p1] + C[S[0]][p2][p1])$$

по всем $p1, p2$ от 0 до 9

- Сложность вычисления этой таблицы для всех строк в сумме – $N * M * 10 * 10 * 10$

E. EQUATION

E. Equation

- Если $N = 2$, то решения не существует
- В противном случае можно сделать так, чтобы и сумма, и произведение чисел равнялись 0
- Например:
 - Все целые числа от 0 до $N - 2$
 - Число $\frac{-(N-2)(N-1)}{2}$

F. FORMULA 8

F. Formula 8

- Столкновение возникает, если за одно и то же время один участник проехал целое число восьмерок, а второй – целое и еще одну половину

F. Formula 8

- Моменты столкновений – решения уравнений:

$$A_1 \frac{L_1 + L_2}{V_1} = A_2 \frac{L_1 + L_2}{V_2} + \frac{L_2}{V_2}$$
$$B_2 \frac{L_1 + L_2}{V_2} = B_1 \frac{L_1 + L_2}{V_1} + \frac{L_1}{V_1}$$

F. Formula 8

- Рассмотрим одно из уравнений. Немного преобразуем его:

$$A_1 V_2 (L_1 + L_2) - A_2 V_1 (L_1 + L_2) = L_2 V_1$$

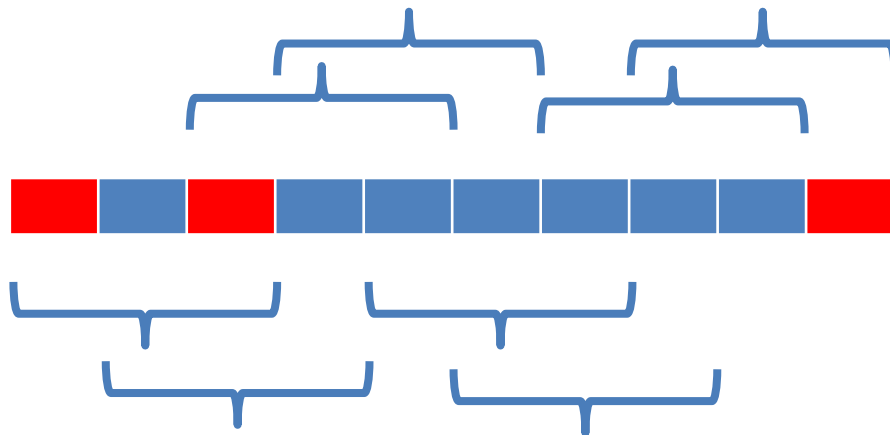
- Столкновение возникает тогда и только тогда:

$$L_2 V_1 \equiv 0 \pmod{(L_1 + L_2) \gcd(V_1, V_2)}$$

G. GRAB YOUR SEAT!

G. Grab your seat!



- Рассмотрим все $(M - N + 1)$ группы из N подряд идущих кресел. Среди этих групп выберем ту, в которой занято наибольшее количество кресел. Данная группа и будет ответом. Основная сложность задачи в том, как найти эту группу быстро.
- Решение которое перебирать все возможные позиции с которых начинается группа работает в лучшем случае за $O(M)$, что слишком медленно.



G. Grab your seat!

- Отсортируем за $O(N \log N)$ позиции в которых изначально находятся участники.
- Будем перебирать позицию pb в которой в начинается группа занятых кресел, и позицию pe в которой заканчивается группа кресел. Если величина $pe - pb - 1 > N$, то двигаем pb , в противном случае можем увеличить pe .

G. Grab your seat!

	Корректно!
	Корректно!
	Не корректно!
	Корректно!
	Не корректно!
	Не корректно!
	Корректно!

H. HEAL

H. Neal

- Моменты столкновений – решения уравнений:

$$A_1 \frac{L_1 + L_2}{V_1} = A_2 \frac{L_1 + L_2}{V_2} + \frac{L_2}{V_2}$$
$$B_2 \frac{L_1 + L_2}{V_2} = B_1 \frac{L_1 + L_2}{V_1} + \frac{L_1}{V_1}$$

I. IS IT TETRIS

I. Is It Tetris

- Любая фигурка из четырех элементов – фигурка Тетриса
- Достаточно посчитать количество связанных фигурок размера четыре
- Например, отмечать фигурки поиском в глубину

J. JUMP!

J. Jump!

- Рассмотрим граф, каждая вершина графа соответствует кувшинке в один из моментов времени t от 0 до 1000. Всего $W \cdot H \cdot 1001$ вершин.
- Ребра в графе – допустимые прыжки между кувшинками
- Необходимо написать алгоритм нахождения кратчайшего расстояния в невзвешенном графе. Например, с помощью поиска в ширину или Дейкстры.

J. Jump!

- Подводные камни:
 - Иногда надо подождать на берегу и прыгнуть первый раз на кувшинку после того как она всплывет не первый, а второй, третий и т.д. раз.
 - Необходимо нескольких секунд стоять на кувшинке.
 - Необходимо сделать шаг влево.

K. KEY NUMBER

K. Key Number

- Читать число как строку и вывести ее задом наперед.
- См. задачу “В” пробного тура.

L. LOOKING FOR NEXT STRING

L. Looking for Next String

- Следующей строки нет только для строки вида

zzzzz...zz

- Найти самый правый не-Z, заменить его на следующий в алфавите, все, что правее заменить на A