DOCUMENTO HECHO POR MAGODY

Preparación

- Pre generar funciones de utilidad: factorial, fibonacci, imprimir, copiar matriz, etc..
- Generar un template para leer, procesar datos y testear.
- Llevar las librerías adecuadas como boost/multiprecision en el caso de c++, consultar si en la competencia se pueden usar.
- Usar un IDE con debugger.
- Familiarizarse con el inglés y tener un traductor abierto. A veces el no entender bien el lenguaje frustra y hace que se demore mucho la primera fase de la ejecución.

Ejecución

Análisis en papel

Fase 1: Entender la lógica del enunciado

- Hay que traducir el problema a nuestro lenguaje. Realizar anotaciones en una hoja física de dibujos, condiciones o variables que creamos importantes (no copiar el problema en la hoja, solo lo importante).
- Un problema casi siempre tiene un objetivo al cual apuntar para llegar a una solución. Anotar el objetivo en lo posible.
- Traducir el inglés a palabras más familiares.
- Comprobar las restricciones del enunciado y condiciones especiales.

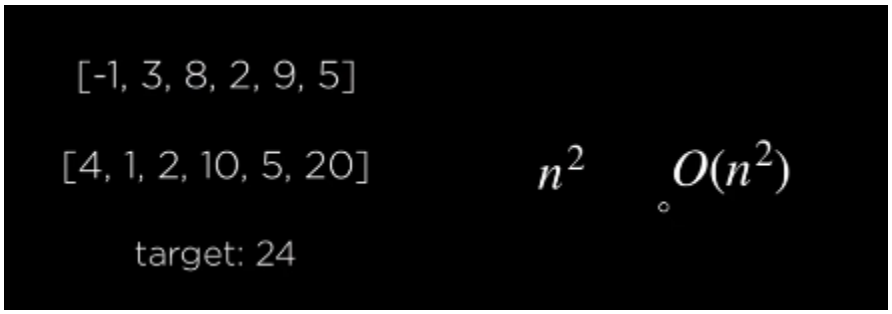
Fase 2: Analizar y crear casos de prueba

- Si existen casos de prueba, analizarlos para REFORZAR el entendimiento del problema de la fase 1.
- Crear uno mismo una versión simplificada del problema.
- Crear uno mismo una versión ampliada del problema.
- Hay que crear varias versiones ya que si tenemos pocos ejemplos podemos crear una solución que solo sirve para esos ejemplos, pero no para casos particulares.

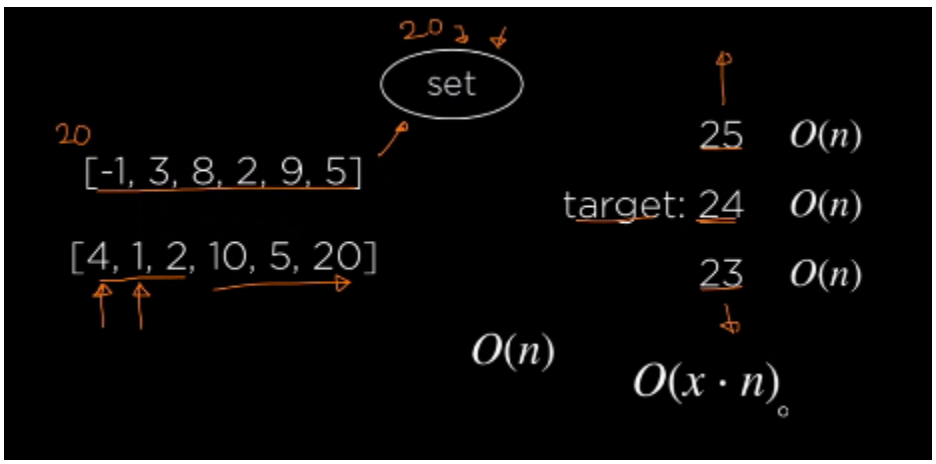
Fase 3: Generar visualizaciones

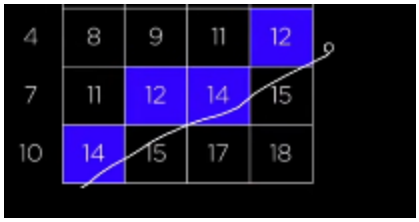
1. Probar varias representaciones de los datos, aunque el problema parezca cómodo de ser representado en una matriz, puede que la mejor opción sea representarlo en un vector o un diccionario. No descartar ninguna representación solo porque “no es cómoda”.

2. A veces puede ser buena idea comenzar con una solución de fuerza bruta para observar mucho mejor el flujo



Y a partir de esto generar una versión más simple del problema para detectar patrones

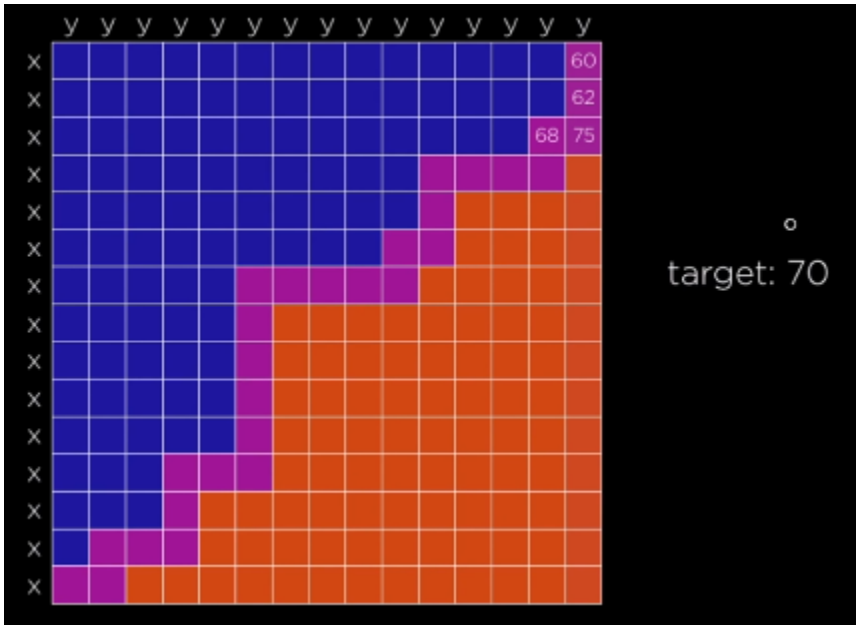




3. Generar visualizaciones de los casos de estudio, se debe poder escribir y dibujar sobre esas visualizaciones. Recomendación: Utilizar una hoja en blanco aparte para esto, dibujar con esfero el caso de prueba y sobre el esfero con lápiz hacer dibujos de posibles patrones e hipótesis.

Al graficar el problema considerar todos los pasos, para elaborar un posible algoritmo. Esto ayuda a previsualizar todo el problema. Luego se puede resaltar las partes que contienen la solución

Si es posible, utilizar graficadores de geogebra o similares para ver todo aún más gráfico.



Fase 4: Determinar las condiciones que se necesitan para que ocurra la respuesta

1. Buscar patrones en la visualización y flujo de resolución de los casos de prueba. Al buscar el patrón obviar muchos pasos intermedios con el procesamiento de datos, solo imaginar que se pasa de un estado a otro mágicamente. Hay que tener una mente abierta y creativa para imaginar soluciones poco comunes.

Nota: Algunos problemas tienen soluciones “mágicas” . Esto es una operación matemática que puede o no relacionarse con el tema, pero llega directo a la respuesta.

Por ejemplo para contar un número de caminos posibles en donde se puede ir solo en derecha y hacia abajo en una matriz la solución normal es generar todos los caminos posibles de forma recursiva e ir contando cuando llega al final

La solución mágica es utilizar la fórmula de “Permutaciones con repetición” sobre una lista de n valores que tienen repetido la dirección derecha y abajo. La solución mágica no hace nada por generar los caminos pero llega directamente a la respuesta.

2. Para evitar razonamientos duplicados hay que anotar algunas deducciones importantes, **que hayamos confirmado que sean verdad,** anotar una deducción con una verdad a medias es perjudicial; sería mejor no anotar.

Algunos problemas tienen distractores y hacen lucir al problema más complejo de lo que realmente es.

3. Si las condiciones para encontrar la respuesta son muy complejas (como involucrar integrales y ecuaciones diferenciales) o realizar operaciones muy pesadas como fuerza bruta, descartar esa rama de razonamiento y pensar en otra. A veces los problemas complejos se resuelven mediante operaciones sencillas matemáticas o con patrones muy especiales.
4. Generar hipótesis para llegar a la solución y buscarle un contraejemplo

En el análisis encontraremos varias posibles condiciones que pueden determinar el llegar de un punto inicial a la solución. Para cada condición encontrada hay que buscarle casos de prueba o restricciones que pueden hacer que no se cumpla, es decir, hay que generar contraejemplos.

Verificar que efectivamente una condición se cumple, por ejemplo para medir si un punto está dentro de una elipse es necesario considerar si está rotada o no, si se usa la fórmula de ellipse que se encuentra por ahí, no funcionará para rotadas. En ese caso la condición se cumple si la distancia entre el punto y los dos foci (su suma) es menor o igual que el diámetro de la elipse del eje más ancho. Pero no cuando al reemplazar la inecuación determinamos si el valor es menor o igual.

5. Representar la conexión del problema y la solución con toda estructura de datos que parezca útil. La representación de los datos debe tener un objetivo **claro** que sale a partir del análisis de las hipótesis..

Una matriz tiene muchas propiedades matemáticas que se pueden utilizar para ahorrar recursos, pero no siempre es la solución. En esta parte se debe seleccionar el método de representación.

Visualizar el problema como un árbol puede ser beneficioso, si llega a ser posible con recursividad y programación dinámica se puede llegar a una solución.

Priorizar el uso de la memoria ram en lugar de extender el tiempo. No importa si la estructura de datos pueda crecer enormemente, mientras sea útil esos cálculos realizados.

(opcional): Prototipar con algoritmo en lugar de pseudocódigo

- NO usar pseudocódigo muy similar al lenguaje de programación.

Por ejemplo en lugar de

```
Int a=3; b=0;
while(b<=3){
        if(a>=4){ b+=1; }
}
b += sqrt((c-a)²+ (c-b)²)
```

Escribir en papel el algoritmo lo más cercano al humano posible

1. b>3? Sumar distancias a los focos en ‘b’
2. Usar la distancia en a...

Escribir un algoritmo en lugar de pseudocódigo ahorra mucho tiempo y además permitirá hacerle cambios de forma más facil, de modo que implementar el algoritmo sea la parte fácil de todo.

Programación

Programar es la parte fácil.

Con la planificación hecha en papel esta es la parte fácil. Si la planificación fue errónea todo se cae.
Si fallas al planificar, estás planificando fallar.

- Considerar que tanto van a crecer los números, para determinar si usar una precisión de datos diferente en orden de que todo se calcule correctamente.
- No usar float nunca, solo precisiones altas.
- Pensar si esta representación crece adecuadamente con los límites impuestos o si en caso involucra una operación factorial, puede que no sea una buena solución para números excesivamente grandes.

Test

- Probar el programa primero con todos los casos realizados a mano.
- NO depurar con mensajes en lo posible, siempre usar el debugger de un IDE con inspector de variables como VSCODE.
- Utilizar las cotas iniciales y finales de las restricciones. A veces el programa funciona bien solo cuando son números pequeños, pero cuando son grandes se puede desbordar todo. O puede funcionar para valores grandes pero no para pequeños.
- Automatizar la ejecución del programa y de las pruebas.
- Si el algoritmo resuelve los casos de ejemplo pero no los reales que no podemos ver, significa que debemos crear más casos de prueba con más combinaciones extrañas y analizar con la herramienta debug que los cálculos se hacen correctamente.

(Situacional) Optimización

Si y solo si la solución es correcta pero el programa se demora mucho hay que:

- Si la eficiencia es relativamente buena, buscar bajar la complejidad Big-O del programa.

Para esto se puede jugar con los valores de la entrada, cambiar la forma de representación o simplemente almacenar cálculos que están siendo hechos varias veces. Por ejemplo un programa simulación de montecarlo puede estar recalculando varios valores una y otra vez, almacenar esos valores en la RAM y continuar.

- Si la eficiencia es mala, buscar una integración de la programación dinámica y aumentar el uso de memoria RAM.
- Si la eficiencia no mejora, entonces implementar la solución en un lenguaje de más bajo nivel como c++.
- Si la eficiencia no mejora a partir de este punto, significa que en la planificación se eligió una solución semejante a fuerza bruta. Si con este ejercicio se generaron puntos se recomienda abandonarlo, llegar al 100/100 ya no es posible.

Futuro, aprendizaje

Por cada ejercicio resuelto es importante documentar para reforzar la lógica de programación en nuestros cerebros. No hay que memorizar nunca.

Errores comunes

- A veces cuando una respuesta es incorrecta pero el algoritmo es correcto, el número del cálculo pudo haberse desbordado. Lenguajes como python no tienen este problema pero c++ definitivamente si, despues de cierto número se desborda y se vuelve negativo o valor basura.
- El cálculo flotante suele acumular error, incluso variables de tipo double van acumulando un error. Si ese error se lo deja pasar en millones de iteraciones el resultado final puede diferir del correcto por varias décimas pero esto ya puede hacer que la plataforma nos marque como error

Temas

Bitwise operations

Data structures

Dynamic programing

Memoization

Tabulation

Hashmap

Lists

Sets

The Two pointers tecnic

Sliding window

String processing

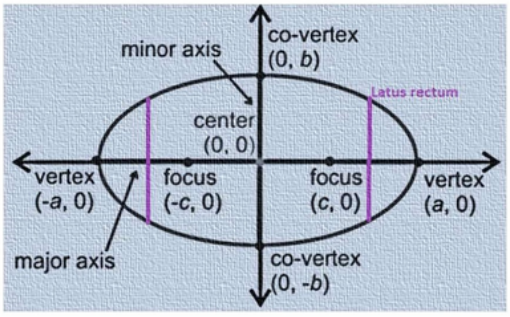
Palindrome

Beetle bag

Montecarlo

Geometry

Ellipse



The standard form of the equation of an ellipse with center $(0,0)$ is

$$\frac{x^2}{a^2}+\frac{y^2}{b^2}=1$$

where

- $a > b$
- the length of the major axis is $2a$
- the coordinates of the vertices are $(\pm a, 0)$
- the length of the minor axis is $2b$
- the coordinates of the co-vertices are $(0, \pm b)$
- the coordinates of the foci are $(\pm c, 0)$, where $c^2 = a^2 - b^2$

Ellipse Equation

Area of ellipse = πab

Perimeter of ellipse = $2\pi\sqrt{\frac{(a^2+b^2)}{2}}$

The standard form of the equation of an ellipse with center $(h,\ k)$ is

$$\frac{(x-h)^2}{a^2}+\frac{(y-k)^2}{b^2}=1$$

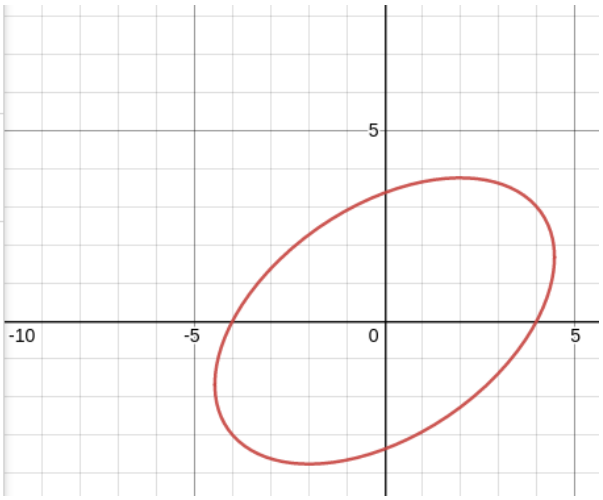
where

- $a > b$
- the length of the major axis is $2a$
- the coordinates of the vertices are $(h \pm a, k)$
- the length of the minor axis is $2b$
- the coordinates of the co-vertices are $(h, k \pm b)$
- the coordinates of the foci are $(h \pm c, k)$, where $c^2 = a^2 - b^2$

$$\frac{(x\cos(a)-y\sin(a))^2}{25}+\frac{(x\sin(a)+y\cos(a))^2}{9}=1$$

$a = -0.6$

−10 10

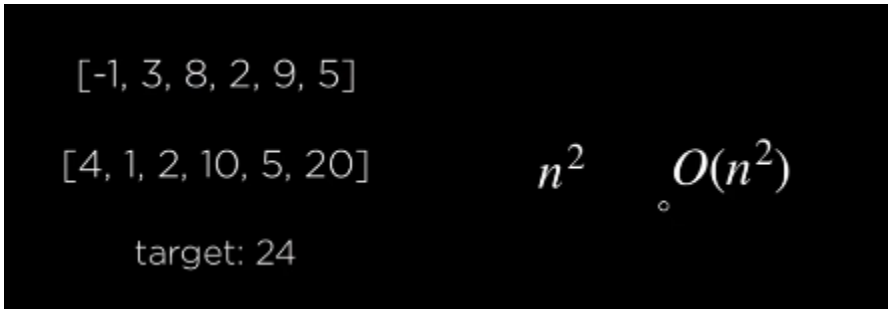


Un punto está dentro de la elipse si la distancia entre el punto y los dos foci (su suma) es menor o igual que el diámetro de la elipse del eje más ancho.

Example solution flow

First, come up with a brute-force solution

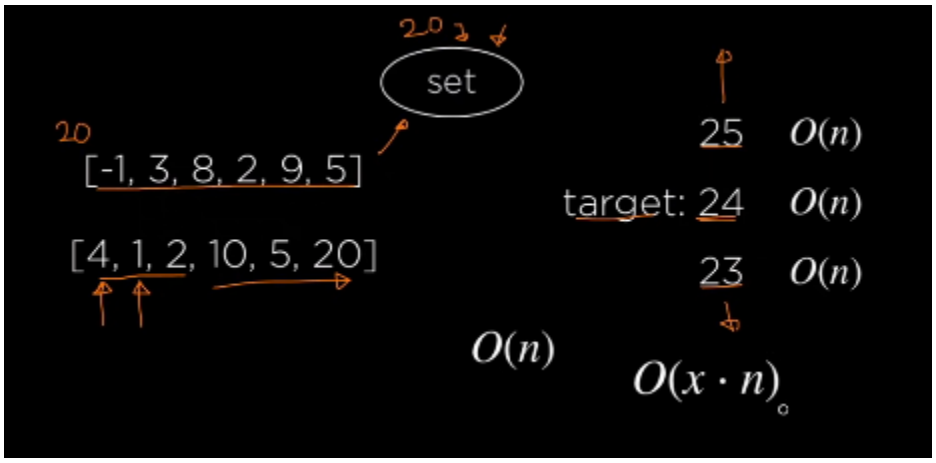
For example check every single pai



Think of a simpler version of the problem

If brute-force is not enough, go to next step
A simpler version could be: define a pair of number that sum 24 or near

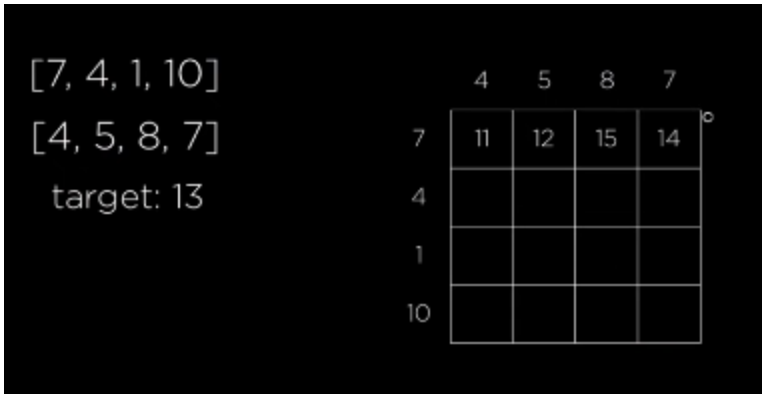
For each element ask if exist the complement in the other side



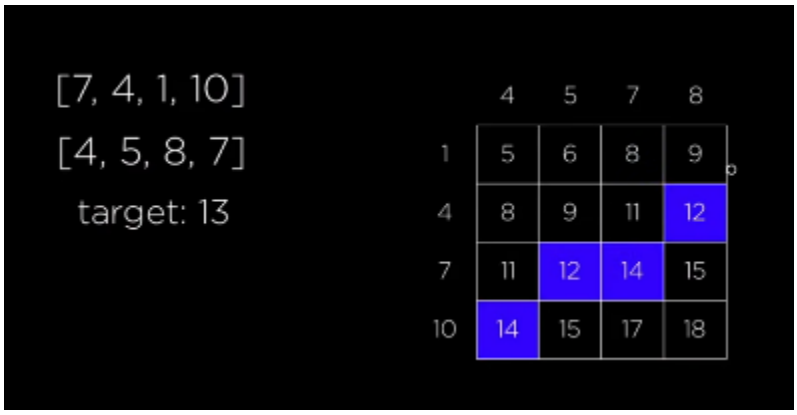
Think with simpler examples, try noticing a pattern

So instead of 6 we can use 4 elements

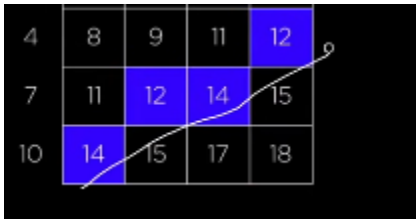
And target is small to



We then notice is better to sort this index on matrix

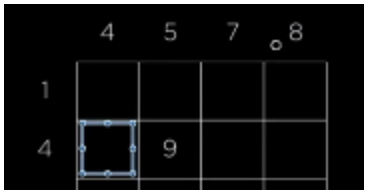


The pattern can be the alignment



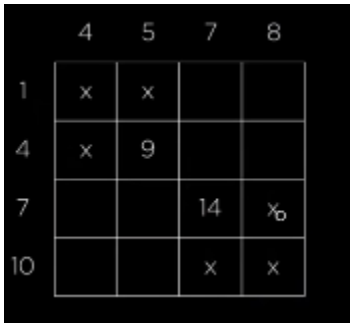
We can draw one by one to notice something:

We also can notice that if we compute a value in matrix, we haven't to compute the previous values



If 4+5 is 9, we dont need to compute 4+4

Also for upper part

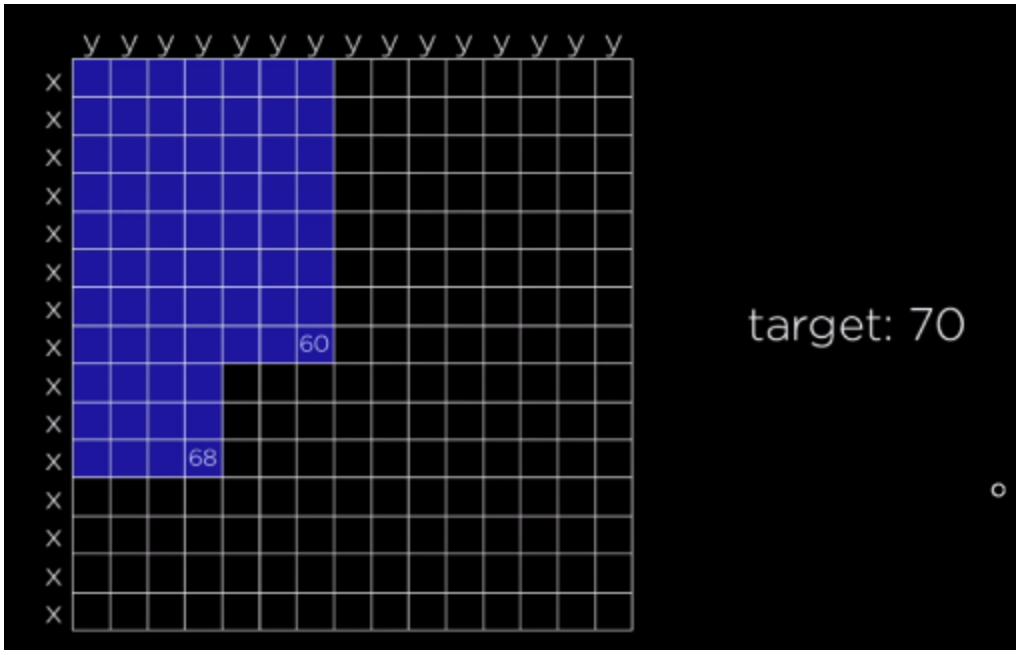


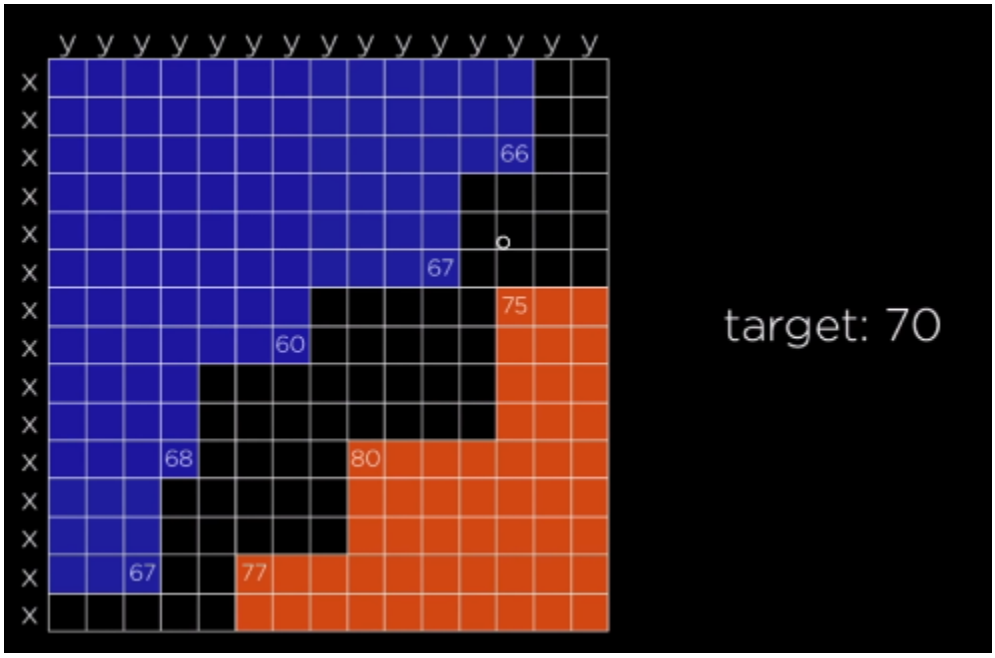
If is 14 we don’t need to compute the nexts

Use some visualization

To visualize we need a bigger example

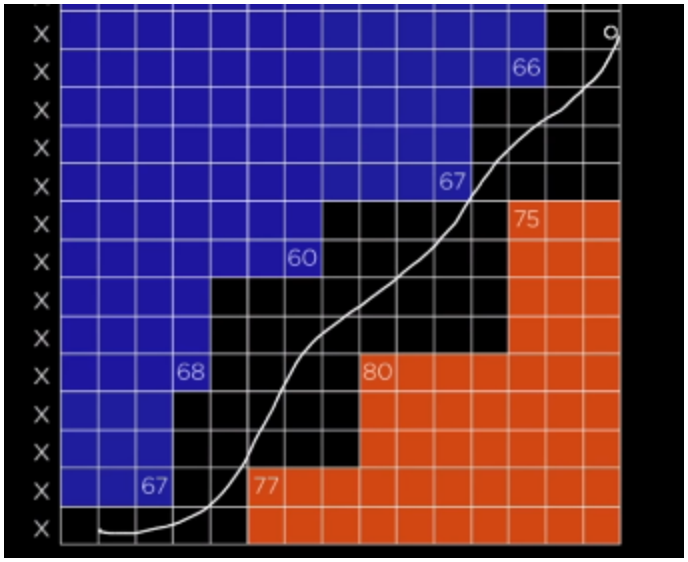
Then we mark what to no compute



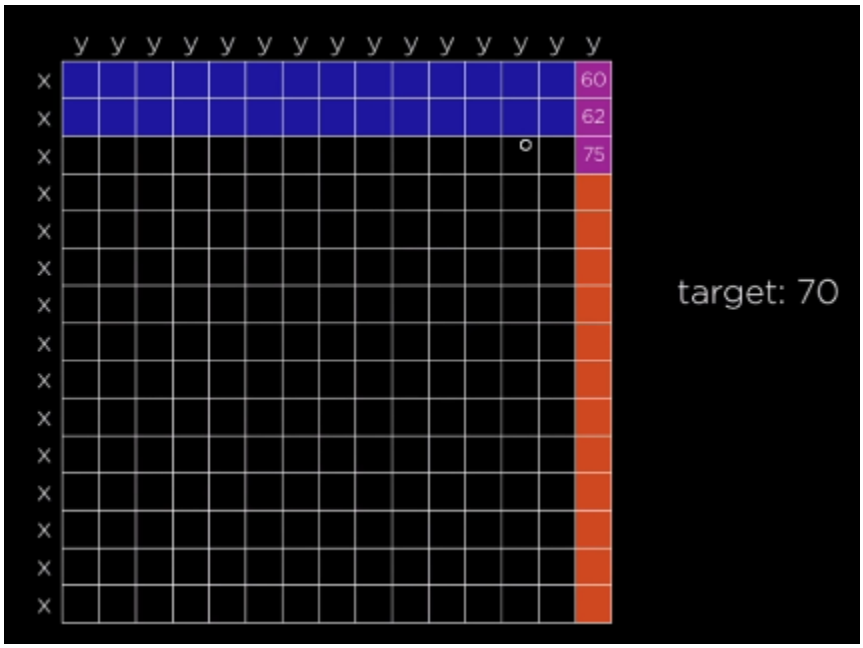


Blue and orange are cell we don’t want to check

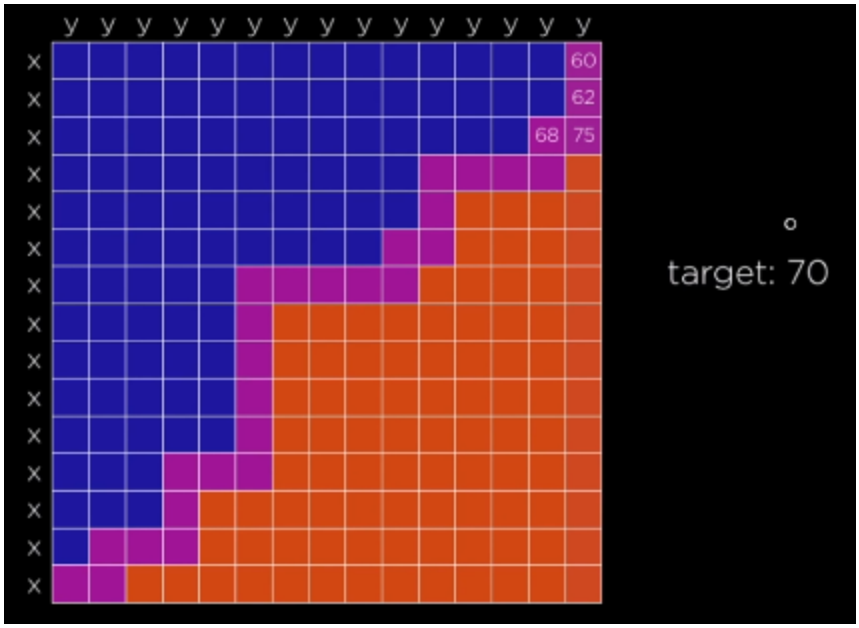
And this has a patter similarly to the first example



We now make a proceduran and controllable form to discard

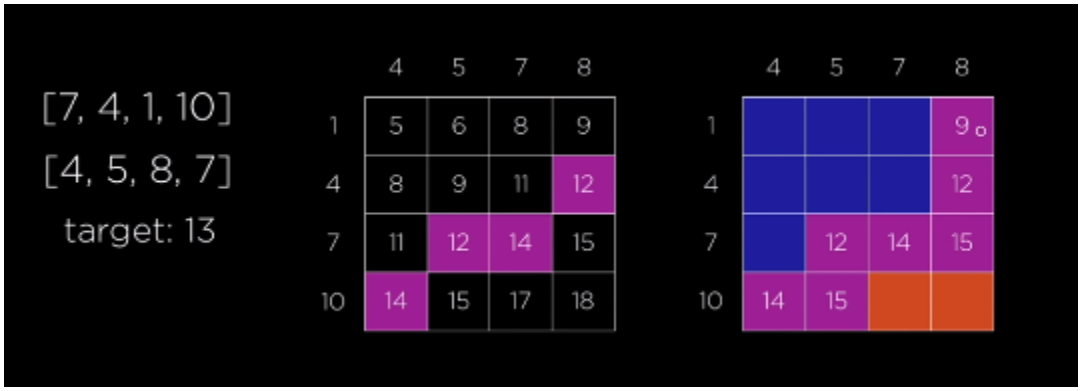


We can start at the end, and go throug the numbers

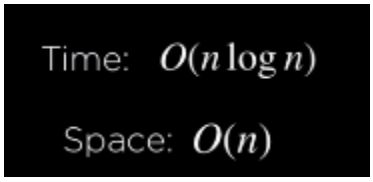


And this numbers ar closer to the target

Test the solution on a few examples



We can keep track



Now we see the interviewer face and see if we can begin the coding

-