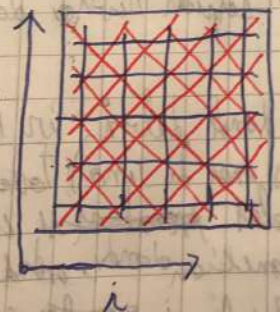
LAB 1

Idea: Primero se consideran las posiciones donde deben ir sucursales (si es que hay más de 1) aplicar fuerza bruta, es decir, todas las otras posibles lugares donde puede ir otra sucursal que cumpla con los requerimientos iniciales, o sea, que no debe existir conexion entre las horizontales, verticales y diagonales, es decir, si es una matriz donde esta i y j. Ver todas las posibilidades que cumplan con que la posicion inicial [i][j] no deben haber otras sucursales en:

* i = i
* j = j
* i - m y j - m
* i + m y j + m
* i + m y j - m
* i - m y j + m

donde m es un numero desde la posicion hasta el largo de la matriz (horizontal o vertical) Mientras se va aplicando la fuerza bruta para cada caso, se irá guardando la cantidad de sucursales que tiene cada caso, ese será la cantidad de sucursales maximo y solo se reemplazara si se encuentra un caso mayor en un caso posterior, esto se realiza hasta que se verifiquen todos los casos, para despues quedarnos con la cantidad de sucursales maxima y su respectivas posiciones que también son almacenadas

Ejemplo de posiciones de matriz 5 X 5



En caso de que no haya sucursales iniciales que deban ir ni o ni el algoritmo comenzara de la posicion inicial [0][0] y vera todas las posibles lugares que cumplan con los requerimientos iniciales

Pseudocódigo: Este algoritmo es inicial, es decir el general, hay funciones que todavía no se realizan, entonces se dan como que funcionan en este que será la función general del código.

En la función principal se ingresa una lista con todas las posiciones de las sucursales que deben ir X o X, de no haber sucursales iniciales se entrega una lista vacía, además se ingresa el largo y el ancho de la matriz.
La salida debe ser la cantidad máxima de sucursales que pueden haber junto a la lista de coordenadas, de haber más de un caso que cumpla con la mayor cantidad de sucursales, ya que no se especifica, el algoritmo guardará el último que encuentre

```
BrudeForce (l: lista, n, m): listaFinal
    listaFinal = []
    Si largo(l) == 0:
        listaFinal = BrudeForce2  #Realiza fuerza bruta sin sucursales iniciales
        devolver listaFinal
    Sino:
        cantidad = 0
        Mientras cantidad < largo(l):          # Para acortar algoritmo
            Para i desde 0 hasta n-1:          pos i y pos j equivalen a la posición
                Para j desde 0 hasta m-1:      de la sucursal inicial a evaluar
                    Si (i =! pos i y j =! pos j):
                        Si (i-val =! pos i-val y j-val =! pos j-val) o (i+val =! pos i+val y j+val =! pos j+val)
                            Si (i+val =! pos i+val y j-val =! pos j-val) o (i-val =! pos i-val y j+val =! pos j+val):
                                lista2 = agregar (pos[i][j])  se agrega posición que cumple

        lista3 = DescartarConjuntos (lista2)  #descartaca y dejara conjunto con + cant de sucursales

        sucursales = largo(lista3)
        Si maxSucursales < sucursales
            listaFinal = lista3 + len(lista3)  #se agrega posiciones de las sucursales mas cantidad de sucursales

    devolver listaFinal
```

val es el número que va variando

Traza: Se dará la traza para una matriz de 5x5 donde tendrá 2 sucursales que deben ir x o x y se va agregando sucursales que cumplan con los requisitos, para después en otra función eliminar las sucursales que no cumplen con los requisitos de las sucursales que se agregaron

	Sucursales	Cantidad de sucursales
Sucursales iniciales	pos[0][4] y pos[3][3]	2
Agregar sucursales	pos[0][4], pos[3][3] y pos[1][2]	3
Quedan el total de sucursales	pos[0][4], pos[3][3], ... hasta	9

Quedan solo las sucursales que cumplen con los requisitos después de pasar por la función DescartarConjuntos