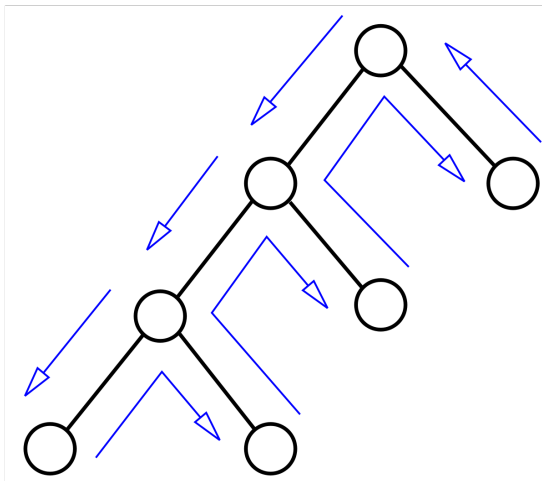


# **Recursividad usando Backtracking**

Computer Science

**CS1100 - Introducción a Ciencia de la Computación**

# Algoritmo Recursivo usando Backtracking?



# Logro de la Sesión

Al finalizar esta sesión, estarás en la capacidad de:

- Conocer Backtracking como estrategia algorítmica para resolver problemas.

# Logro de la Sesión

Al finalizar esta sesión, estarás en la capacidad de:

- Conocer Backtracking como estrategia algoritmica para resolver problemas.
- Utilizar Recursividad.

# Logro de la Sesión

Al finalizar esta sesión, estarás en la capacidad de:

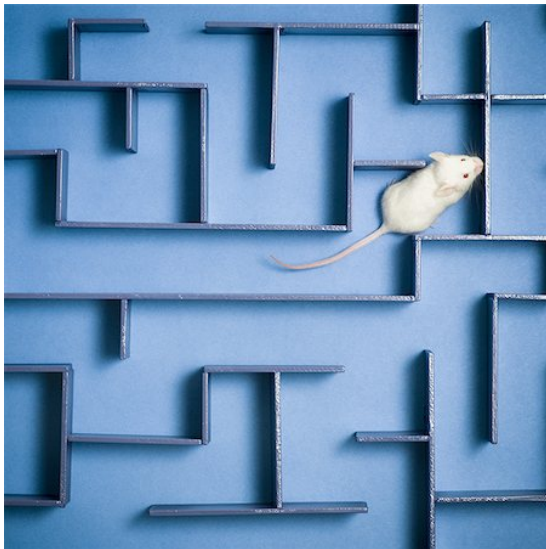
- Conocer Backtracking como estrategia algorítmica para resolver problemas.
- Utilizar Recursividad.
- Construir soluciones incrementalmente.

# Logro de la Sesión

Al finalizar esta sesión, estarás en la capacidad de:

- Conocer Backtracking como estrategia algorítmica para resolver problemas.
- Utilizar Recursividad.
- Construir soluciones incrementalmente.
- Utilizar Backtracking para solucionar problemas de laberintos.

# Rata en Laberinto

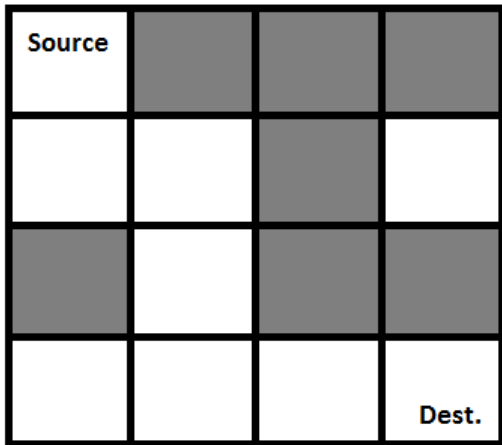


# Rata en Laberinto

- Un laberinto es una matriz binaria  $N \times N$ , donde  $N$  es el lado de la matriz cuadrada.
- El bloque origen esta ubicada en la parte superior izquierda. (e.g. `laberinto[0][0]`)
- EL bloque destino esta ubicada en la parte inferior derecha.(e.g. `laberinto[N-1][N-1]`)
- La rata empieza desde el origen y tiene que llegar al destino.
- La rata solo puede moverse en dos direcciones: Hacia adelante y hacia abajo.
- Si el bloque del laberinto guarda el valor de **0** significa que esta en un callejon sin salida.
- Si el bloque del laberinto guarda el valor de **1** significa que puede ser usada en el camino del origen al destino.



# Rata en Laberinto

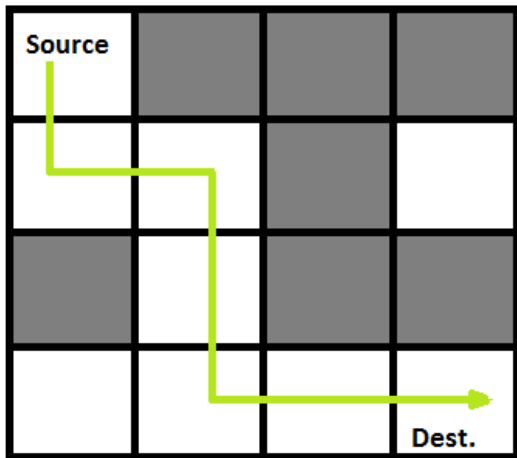


# Rata en Laberinto

```
1 {1, 0, 0, 0}
2 {1, 1, 0, 1}
3 {0, 1, 0, 0}
4 {1, 1, 1, 1}
```

Listing 1: Representación Binaria del Laberinto

# Rata en Laberinto



# Rata en Laberinto

```
5 {1, 0, 0, 0}
6 {1, 1, 0, 0}
7 {0, 1, 0, 0}
8 {0, 1, 1, 1}
```

Listing 2: Solución

# Rata en Laberinto

```
9  while Existen caminos no explorados{  
10      generar un nuevo camino  
11      if el nuevo camino tiene todos los bloques en 1 {  
12          imprimir este camino  
13      }  
14  }
```

Listing 3: Solución sencilla

# Rata en Laberinto

```
16  if destino ha sido encontrado
17      imprimir la matriz solución
18  else
19      a) Marcar el bloque actual de la matriz solución como 1
20      b) Moverse hacia adelante horizontalmente y recursivamente
           verificar
21          si el movimiento hecho llega a la solución
22      c) Si el movimiento escogido en "b)" no llega a la solución
           entonces
23          realizar movimiento hacia abajo y verificar si se llega a
              la solución.
24      d) Si ningún movimiento realizado en "b)" o "c)" funciona
           entonces
25          desmarcamos el bloque como 0 (BACKTRACK) y retornar
              False
```

Listing 4: Solución usando Backtracking

# Rata en Laberinto

```
26 # Tamanho del laberinto
27 n = 4
28
29 def is_safe(laberinto, x, y):
30     #Funcion utilitaria que verifica si x, y son indices validos
31     if x >= 0 and y >= 0 and x < N and y < N and laberinto[x][y]
        == 1:
32         return True
33     return False
```

Listing 5: Solución

# Rata en Laberinto

```
1 def print_solution(sol):
2     #Funcion utilitaria para imprimir la solución de la matriz
3     for i in sol:
4         for j in i:
5             print(str(j) + " ", end = "")
6         print("")
```



# Rata en Laberinto

```
34 def solve_laberinto(laberinto):  
35     #Crear a 4*4 2D list  
36     sol = [[0 for _ in range(N)] for _ in range(N)]  
37  
38     if solve_laberinto_util(laberinto, 0, 0, sol) == False:  
39         print("No existe Solución")  
40         return False  
41     print_solution(sol)  
42     return True
```

Listing 6: Solución

# Rata en Laberinto

```
43 #Función Recursia utilitaria para resolver el problema del
    laberinto
44 def solve_laberinto_util(laberinto, x, y, sol):
45     #if (x, y is goal) return True
46     if x == N-1 and y == N-1:
47         sol[x][y] = 1
48         return True
49     #verificar si laberinto[x][y] es valido
50     if is_safe(laberinto, x, y) == True:
51         #marcar x, y como parte de la solución
52         sol[x][y] = 1
53         #movemos hacia adelante
54         if solve_laberinto_util(laberinto, x+1, y, sol) == True:
55             return True
56
57         #si moviendo hacia adelante no nos da la solución
58         #entonces movemos hacia abajo
59         if solve_laberinto_util(laberinto, x, y+1, sol) == True:
60             return True
61
62         #si ninguno de los movimientos funciona
63         #BACKTRACK: desmarcamos x, y como parte de la solución
64         sol[x][y] = 0
```



# Ejercicio 1

## Enunciado

Escribir una función recursiva que calcule la multiplicación de un número por 5

## Ejercicio 2

### Enunciado

¿Cuál será el capital de 10K Soles despues de 10 años si el interés anual es del 8%? Programe la solución con una función recursiva

## Ejercicio 3

### Enunciado

La cantidad de bacterias en un cultivo se triplica cada hora. ¿Cuántas bacterias habrán después de 10 horas? Programe la solución con una función recursiva

# Cierre

En esta sesión aprendiste:

- Qué es Backtracking

# Cierre

En esta sesión aprendiste:

- Qué es Backtracking
- Cómo se relaciona Backtracking con algoritmos recursivos

# Cierre

En esta sesión aprendiste:

- Qué es Backtracking
- Cómo se relaciona Backtracking con algoritmos recursivos
- Resolver problemas usando backtracking