Laboratorio 1

Saira Sharid Sanabria Muñoz

Carlos Andres Suarez Torres

Universidad Santo Tomás

Facultad de Ingeniería Electrónica

Presentado a: Diego Alejandro Barragan Vargas

Bogotá, Colombia

28 de Agosto de 2025

1. Introducción

El presente documento describe el desarrollo del Laboratorio 1, que comprende tres proyectos fundamentales en el área de algoritmos y simulaciones. Estos proyectos exploran diferentes conceptos esenciales de la ciencia de la computación, implementados mediante el lenguaje de programación Python.

Cada proyecto aborda problemáticas específicas: la resolución del clásico 8-Puzzle mediante BFS, la navegación en laberintos con sistema de recompensas, y simulaciones educativas de sistemas de estados finitos. Estas implementaciones demuestran principios algorítmicos fundamentales y su aplicación en contextos prácticos, sirviendo como base para el entendimiento de conceptos más avanzados en inteligencia artificial y planificación.

2. Objetivos

General

Implementar y analizar tres sistemas algorítmicos que demuestren principios fundamentales de búsqueda, planificación de caminos y transiciones entre estados, aplicando conceptos teóricos en soluciones prácticas.

Específicos

1. Desarrollar un solucionador para el problema 8-Puzzle implementando el algoritmo de Búsqueda en Amplitud (BFS) para encontrar la solución óptima, modelando representaciones de estados y transiciones válidas, y gestionando espacios de búsqueda complejos de manera eficiente.
2. Crear un sistema de pathfinding para laberintos implementando un algoritmo BFS para encontrar todos los caminos posibles, diseñando un sistema de recompensas y penalizaciones, desarrollando mecanismos de detección y evitación de obstáculos, y realizando análisis completos de transiciones entre estados.

3. Implementar simulaciones educativas de sistemas de estados finitos modelando la navegación de un barco pirata en una cuadrícula 3x3, simulando el comportamiento de una planta con transiciones de estado, y demostrando conceptos de transiciones entre estados basadas en acciones.

3. Marco Teorico

El espacio de estados es una herramienta fundamental en la resolución de problemas dentro de la Inteligencia Artificial (IA). Se define como el conjunto de todas las configuraciones posibles en las que puede encontrarse un sistema o problema. Cada configuración se denomina estado, y representa una situación particular que puede ocurrir dentro del sistema analizado.

Un problema puede describirse formalmente mediante cuatro componentes principales:

1. Estado inicial: Punto de partida desde el cual comienza la búsqueda de una solución.
2. Estado meta: Configuración final que representa la solución deseada.
3. Espacio de estados: Conjunto de todos los estados alcanzables a partir del estado inicial mediante la aplicación de acciones.
4. Acciones o operadores: Reglas que permiten la transición de un estado a otro.

4. Metodologia

Enfoque de Desarrollo

Se adoptó un enfoque modular para el desarrollo, creando tres proyectos independientes pero conceptualmente relacionados. Cada proyecto fue implementado en Python, aprovechando sus capacidades para estructuras de datos y algoritmos.

Herramientas Utilizadas

1. Lenguaje de programación Python 3.x.
2. Estructuras de datos: listas, diccionarios, tuplas.
3. Algoritmos de búsqueda (BFS).
4. Sistemas de representación de estados y transiciones.

Estrategia de Implementación

1. Análisis del problema: Identificación de estados, acciones y transiciones
2. Diseño de la solución: Definición de estructuras de datos y algoritmos
3. Implementación: Codificación de las soluciones propuestas
4. Pruebas: Validación de los algoritmos con diversos casos de prueba

Estructura del Proyecto

```
Laboratorio1/
 Puzzle_8/                  # Solucionador de 8-Puzzle
 Laberinto_Solver/          # Sistema de pathfinding en laberintos
 Algoritmo-Concepto-Espacio/  # Simulaciones de Barco Pirata y Planta
```

5. Desarrollo

En esta práctica de laboratorio se aplicaron los conceptos de espacio de estados, acciones y estados meta a diferentes problemas representativos.

5.1. Problema del 8-Puzzle

El primer ejercicio consistió en el clásico 8-puzzle, el cual se compone de un tablero con ocho fichas numeradas y un espacio vacío.

Se implementó un algoritmo basado en búsqueda en amplitud (BFS) para encontrar la secuencia de movimientos que lleva desde el estado inicial al estado final.

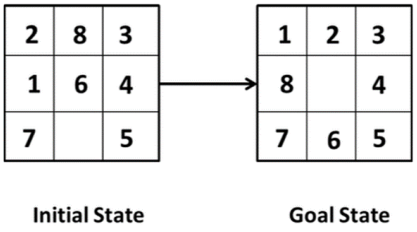

Figura 1: sistema en lazo abierto

Implementación:

Se representó el tablero como una lista de 9 elementos (3x3). Se definieron las acciones posibles: mover el espacio vacío hacia arriba, abajo, izquierda o derecha. Se utilizó una cola para almacenar los estados por visitar y un diccionario para registrar los padres de cada estado y reconstruir la solución.

Estructura del Proyecto

```
Puzze-8/
PuzzleState → Representa el estado del tablero
PuzzleSolver → Implementa el algoritmo BFS para resolver el puzzle
PuzzleGame → Proporciona la interfaz y lógica del juego
     # Ejemplos de uso y demostración
```

Resultado:

El algoritmo encontró una solución en 5 movimientos. La secuencia fue la siguiente:

```
Solución encontrada en 5 pasos:
Estado inicial:
2 8 3
1 6 4
7 0 5

Paso 1: Mover up
2 8 3
1 0 4
7 6 5

Paso 2: Mover up
2 0 3
1 8 4
7 6 5

Paso 3: Mover left
0 2 3
1 8 4
7 6 5

Paso 4: Mover down
1 2 3
0 8 4
7 6 5

Paso 5: Mover right
1 2 3
8 0 4
7 6 5
```

Figura 2: o

5.2. Planta virtual

Se desarrolló un sistema basado en estados finitos para modelar el ciclo de hidratación de una planta, aplicando los conceptos fundamentales de inteligencia artificial como espacio de estados, acciones disponibles y estado meta. El sistema contempló dos estados posibles para la planta:

"HIDRATADAz "SECA", con un estado inicial configurado como "SECAz un estado meta deseado de "HIDRATADA". Las acciones definidas para transitar entre estos estados fueron .AGUAz "SOL", representando las intervenciones básicas que afectan la condición de la planta.

Listing 1: Sistema de estados de la planta

```
# Estados posibles de la planta
estados = ["HIDRATADA", "SECA"]
estado_actual = "SECA"
estado_meta = "HIDRATADA"

def cambiar_estado(estado, accion):
    if accion == "AGUA":
        return "HIDRATADA"
    elif accion == "SOL":
        return "SECA"
    return estado
```

Resultados:

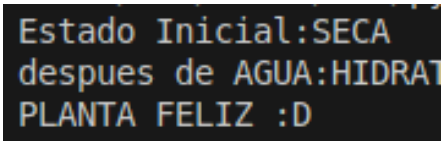

Figura 3: o

5.3. Barco

Se desarrolló un algoritmo de navegación para simular la travesía de un barco pirata en busca de una isla perdida. El modelo representó el océano como una cuadrícula de 3x3, donde cada celda correspondía a una posición geográfica específica. El espacio de estados se definió mediante coordenadas que abarcaban desde la posición "0,0"hasta "2,2", estableciendo el estado inicial en "0,0çomo punto de partida del barco y el estado meta en "2,2çomo la ubicación de la isla tesoro. Las acciones disponibles se determinaron según la posición actual del barco, considerando los cuatro puntos cardinales: norte (↑), sur (↓), este (→) y oeste (←).

Listing 2: Implementación del sistema de navegación marítima

```
estados = ["0,0", "0,1", "0,2",
           "1,0", "1,1", "1,2",
           "2,0", "2,1", "2,2"]

estado_actual = "0,0"
estado_meta   = "2,2"

def acciones(estado):
    x, y = map(int, estado.split(","))
    posibles = []
    if x > 0: posibles.append("NORTE")
    if x < 2: posibles.append("SUR")
    if y > 0: posibles.append("OESTE")
    if y < 2: posibles.append("ESTE")
    return posibles

def cambiar_estado(estado, accion):
    x, y = map(int, estado.split(","))
    if accion == "NORTE":
        return f"{x-1},{y}"
    elif accion == "SUR":
        return f"{x+1},{y}"
    elif accion == "OESTE":
        return f"{x},{y-1}"
```

```
    elif accion == "ESTE":
        return f"{x},{y+1}"
    return estado

print(f"Estado Inicial: {estado_actual}")
ruta_navegacion = ["ESTE", "ESTE", "SUR", "SUR"]

for accion in ruta_navegacion:
    estado_actual = cambiar_estado(estado_actual, accion)
    print(f"Despues de {accion}: {estado_actual}")

if estado_actual == estado_meta:
    print("El barco encontro la isla!")
else:
    print("El barco sigue en busqueda...")
```

Resultados:

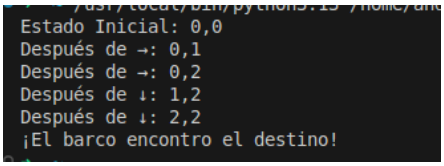

Figura 4: o

5.4. Implementación del Sistema de Laberinto con BFS

El sistema de laberinto implementa un sololvedor de caminos utilizando el algoritmo de Búsqueda en Amplitud (BFS) para encontrar todos los caminos posibles desde un punto de inicio hasta una meta, evitando obstáculos. El sistema incluye un modelo de recompensas que penaliza movimientos no óptimos y premia llegar a la meta.

Estructura del Proyecto

```
Laberinto/
 Laberinto.py              # Clase principal del laberinto
 AnalizadorLaberinto.py    # Clase para análisis y visualización
 EjemplosLaberinto.py      # Ejemplos de uso y demostración
```

Sistema de Recompensas y Transiciones

El modelo implementa un sistema de recompensas que penaliza movimientos ineficientes y premia el éxito:

```python
# Sistema de recompensas
self.recompensas = {
    "inicio": 0,
    "meta": 100,
    "normal": -1,         # Penalizaci n por movimiento
    "obstaculo": -50,     # Penalizaci n fuerte por chocar
    "fuera_limites": -10 # Penalizaci n por salir del laberinto
}
```

Estructura de Clases

La clase Laberinto representa el componente central del sistema de resolución de laberintos. Esta clase se encarga de gestionar toda la lógica fundamental para la creación, configuración y navegación dentro del entorno del laberinto.

Como núcleo del sistema, la clase Laberinto implementa cuatro funcionalidades principales:

1. Inicialización y configuración: Permite crear estructuras de laberinto personalizables mediante parámetros ajustables, facilitando la experimentación con diferentes configuraciones.
2. Gestión de estados: Mantiene un registro completo de todos los estados posibles dentro del laberinto y administra las transiciones válidas entre ellos.
3. Ejecución de movimientos: Implementa la lógica necesaria para transitar entre diferentes estados, validando cada movimiento según las reglas establecidas.
4. Búsqueda de caminos: Ejecuta el algoritmo de Búsqueda en Amplitud (BFS) para encontrar todas las soluciones posibles desde el punto de inicio hasta la meta.

6. Conclusiones

- Se implementó exitosamente el solucionador del 8-Puzzle utilizando el algoritmo BFS, demostrando su capacidad para encontrar la solución óptima mediante la exploración sistemática del espacio de estados.
- Se desarrolló completamente el sistema de pathfinding para laberintos, que permite encontrar todos los caminos posibles con un sistema de recompensas funcional y mecanismos efectivos para evitar obstáculos.
- Se crearon las simulaciones educativas del barco pirata y la planta, que ilustran claramente los conceptos de sistemas de estados finitos y transiciones entre estados basadas en acciones.
- Todos los proyectos cumplieron con el objetivo general de aplicar conceptos teóricos en soluciones prácticas, demostrando principios fundamentales de búsqueda, planificación de caminos y transiciones entre estados.
- Las implementaciones resultaron eficientes y educativas, proporcionando una base sólida para el desarrollo de sistemas más complejos en inteligencia artificial y sistemas autónomos.