INTELIGENCIA ARTIFICIAL

TALLER #1

GUILLERMO FERNÁNDEZ – 345802
MARIO PAULO SALGADO – 345833
ERIKA VILLAGÓMEZ – 345008
GABRIELA ZUMARRAGA – 345769

QUITO 26 DE ABRIL 2025

TALLER #1

REPOSITORIO Y PLANIFICACION

- **Link GitHub:** Borreguin/WS-USFQ
- **Rama del grupo:** Borreguin/WS-USFQ at feature/grupo-3
- **Link a ClickUp:** Taller 1 | USFQ (List)

1. USO DE LA INTELIGENCIA ARTIFICIAL – LOW CODE ENGINEERING

A. TSP – Travelling salesman problema – Problema del vendedor viajante

Entendimiento del problema:

El Travelling Salesman Problem (TSP) es un problema de optimización combinatoria. Consiste en encontrar la ruta más corta que permita visitar una lista de ubicaciones una única vez y regresar al punto de origen, formando un ciclo cerrado. Este problema es conocido por su complejidad computacional, ya que el número de posibles rutas crece exponencialmente con el número de ubicaciones.

El objetivo es implementar algoritmos de búsqueda como BFS, DFS, A Star, utilizando la IA, para encontrar la solución óptima para encontrar la ruta más corta y posteriormente comparar los resultados obtenidos y eficiencia entre los algoritmos utilizados para los dos casos definidos: 10 ciudades y 100 ciudades.

Solución:

La implementación de este problema se encuentra dentro de la carpeta P1_TSP, en el archivo TSP.py; la implementación de los algoritmos de búsqueda se encuentra en el archivo útil.py.

Tomando en cuenta la estructura planteada, se realizó la implementación de 4 algoritmos de búsqueda: BFS, DFS, A star (vistos en clase) y el algoritmo de Vecinos cercanos, utilizando la IA. Para esto, se tomó en cuenta una estructura con grafos.

A continuación, se detalla la estructura de la solución planteada:

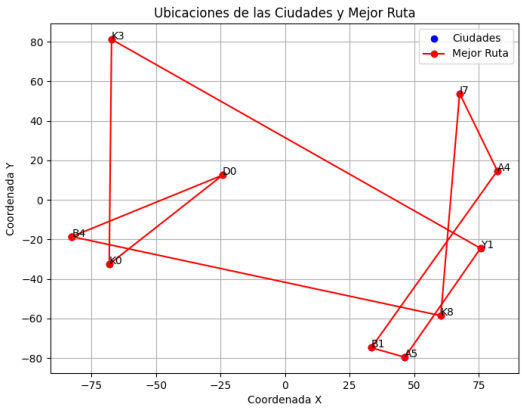
1. **Clases y Métodos**:
    - La clase TSP maneja las ciudades y las distancias entre ellas. Proporciona métodos para calcular la ruta más corta utilizando algoritmos de búsqueda, los cuales se encuentran implementados en el archivo util.py. Los algoritmos implementados son:

- BFS (Breadth-First Search)
- DFS (Depth-First Search)
- A* (A-Star)
- Vecinos Cercanos (Nearest Neighbor)

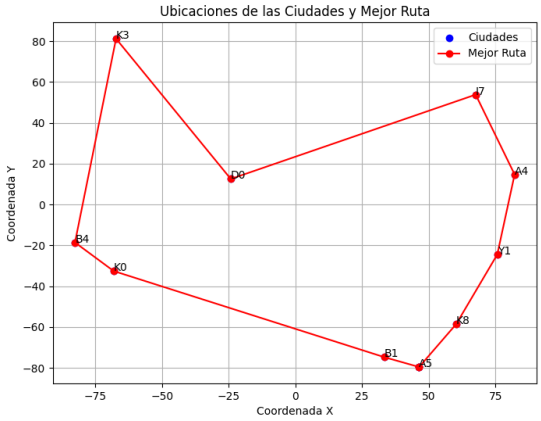
2. **Funcionalidad**:
   - El programa presenta un menú para que el usuario seleccione el algoritmo que desea utilizar para resolver cada caso de estudio del TSP. Al escoger uno de los métodos, el programa mostrara la ubicación inicial de las ciudades y sus distancias. Al cerrar la ventana se mostrará el camino encontrado por el algoritmo seleccionado para el caso de estudio 1. A continuación el programa volverá a pedir que se escoja un algoritmo para resolver el caso de estudio 2 y el mismo proceso se repite. Se recomienda escoger el algoritmo de vecinos cercanos, ya que con los otros no se encuentra una solución en un tiempo optimo.
3. **Casos de Estudio**:
   - **Caso de Estudio 1**: Resuelve el TSP para un conjunto de 10 ciudades generadas aleatoriamente.
   - **Caso de Estudio 2**: Resuelve el TSP para un conjunto más grande de 100 ciudades.
4. **Visualización**:
   - Se muestra las ciudades iniciales.
   - Después de calcular las rutas, el programa muestra las ciudades y las rutas obtenidas.
5. **Medición de Desempeño**:
   - Cada algoritmo mide el tiempo que tarda en encontrar la solución, lo que permite comparar su eficiencia.

Conclusiones:

Para el *Caso de Estudio 1 (10 ciudades)*, se inicializaron las ciudades de la siguiente manera:

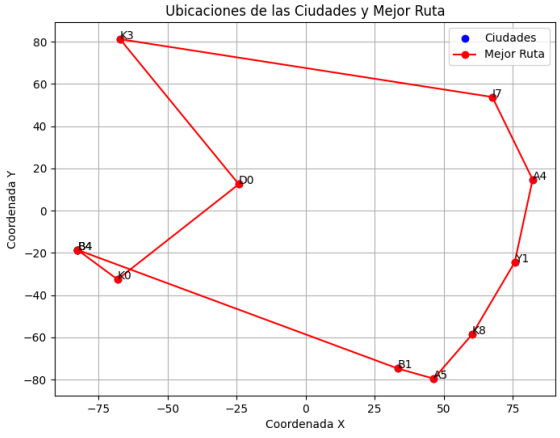


Los algoritmos *BFS*, *DFS* y *A** encontraron la misma ruta más corta, la cual se adjunta a continuación. Sin embargo, los tiempos de respuesta de los algoritmos cambian significativamente.



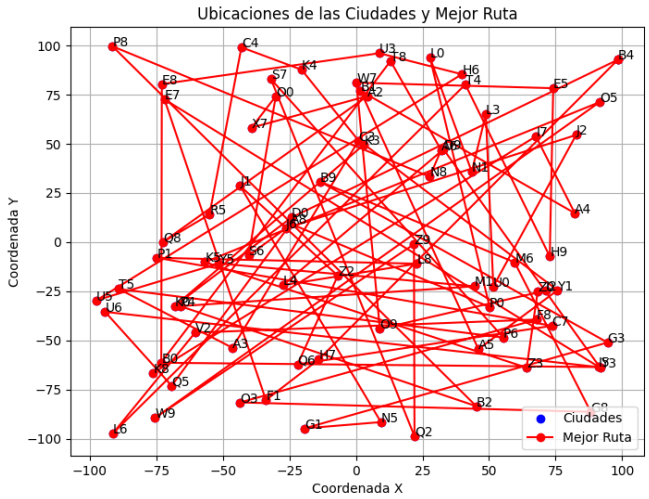
Algoritmo	**Caso 1 (10 ciudades)**
BFS	35.2420 [s]
DFS	20.9615 [s]
A*	290.6966 [s]
Vecinos Cercanos	0.0000 [s]

El algoritmo de *Vecinos Cercanos* encontró una ruta distinta en un tiempo significativamente menor, demostrando ser más eficiente en este caso. Se adjunta la ruta encontrada por este algoritmo.

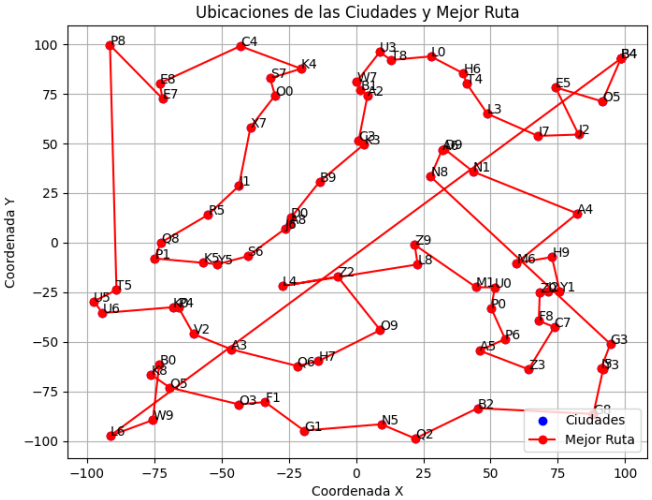


Algoritmo	**Caso 1 (10 ciudades)**
Vecinos Cercanos	0.0000 [s]

Para el *Caso de Estudio 2 (100 ciudades)*, se inicializaron las ciudades de la siguiente manera:



Los algoritmos *BFS*, *DFS* y *A** no lograron obtener una solución probablemente debido a la complejidad computacional. Sin embargo, el algoritmo de *Vecinos Cercanos* encontró una solución en un tiempo significativamente corto, lo que lo hace adecuado para problemas de mayor escala. Se adjunta la ruta mas corta obtenida para este caso.



Algoritmo	**Caso 2 (100 ciudades)**
BFS	No se obtuvo respuesta
DFS	No se obtuvo respuesta
A*	No se obtuvo respuesta
Vecinos Cercanos	0.114 [s]

En conclusión, el algoritmo de vecinos cercanos es el más rápido en ambos casos; para este problema es el más optimo. Para problemas de gran escala, Vecinos Cercanos es una solución práctica debido a su eficiencia computacional. Los algoritmos, BFS, DFS y A* son precisos en problemas pequeños, pero su desempeño se degrada con el aumento del número de ciudades.

B. El acertijo del granjero y el bote

Entendimiento del problema:

El problema plantea que existen cuatro sujetos que deben cruzar de una orilla a otra de un río.

Sujetos:

1: Granjero
2: Lobo
3: Oveja
4: Col

Para esto, se entiende que existen tres posiciones en la que los sujetos de interés pueden estar:

Posiciones:

P1: Orilla 1 (de partida)
P2: En el bote (tránsito)
P3: Orilla 2 (de destino)

Además, se definen condiciones a cumplir para que los sujetos puedan alcanzar el objetivo de que el granjero vuelva a su casa con el lobo, la oveja y la col:

Condiciones:
Se dispone de una barca que puede movilizar dos sujetos a la vez.
El granjero debe llevar a un sujeto a la vez, dejando a los otros dos en una orilla.
El lobo y la oveja no pueden estar solos en una posición: NO 2 y 3 en P1 o P3
La oveja y la col no pueden estar solos en una posición: NO 3 y 4 en P1 o P3

Solución:

Con base en el planteamiento detallado previamente, el problema se representa así:

Tabla 1. Representación por tuplas del problema del granjero

P1	P2	P3
[1, 2, 3, 4]	[0, 0, 0, 0]	[0, 0, 0, 0]
[0, 2, 0, 4]	[1, 0, 3, 0]	[0, 0, 0, 0]

[0, 2, 0, 4]	[0, 0, 0, 0]	[1, 0, 3, 0]
[0, 2, 0, 4]	[1, 0, 0, 0]	[0, 0, 3, 0]
[1, 2, 0, 4]	[0, 0, 0, 0]	[0, 0, 3, 0]
[0, 2, 0, 0]	[1, 0, 0, 4]	[0, 0, 3, 0]
[0, 2, 0, 0]	[0, 0, 0, 0]	[1, 0, 3, 4]
[0, 2, 0, 0]	[1, 0, 3, 0]	[0, 0, 0, 4]
[1, 2, 3, 0]	[0, 0, 0, 0]	[0, 0, 0, 4]
[0, 0, 3, 0]	[1, 2, 0, 0]	[0, 0, 0, 4]
[0, 0, 3, 0]	[0, 0, 0, 0]	[1, 2, 0, 4]
[0, 0, 3, 0]	[1, 0, 0, 0]	[0, 2, 0, 4]
[1, 0, 3, 0]	[0, 0, 0, 0]	[0, 2, 0, 4]
[0, 0, 0, 0]	[1, 0, 3, 0]	[0, 2, 0, 4]
[0, 0, 0, 0]	[0, 0, 0, 0]	[1, 2, 3, 4]

Para la resolución del problema se utilizará el método de grafos. En este sentido, los nodos representarán a los estados, es decir, a los sujetos presentes en la posición 1 o 3, después de que el granjero se mueva en la barca (acción representada por una arista).

El código para la resolución de este problema se puede visualizar en el repositorio del Taller (grafos_2.ipynb -revisar rama de GitHub-)

De manera gráfica la solución se encuentra así:

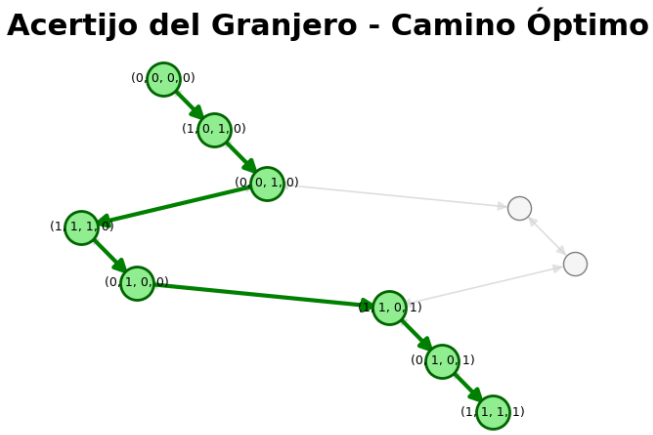


Ilustración 1. Representación de la solución del problema del granjero

Conclusiones:

Debido a la naturaleza del problema, en el que las iteraciones, sujetos y posibles soluciones tienen un número pequeño, se pudo resolver a través de un grafo que lo representó sin

problemas. Además, el consumo de recursos computacionales como el tiempo, para este problema fue bajo.

El grafo permitió representar cada estado de los sujetos después de una acción (movimiento a través del río en barca) representado por las aristas del grafo; pasando de la posición 1 a la posición 3, los cuatro sujetos cumpliendo las condiciones establecidas.

C. La torre de Hanoi

El objetivo es mover una pila completa de discos desde la posición de origen (primera torre) a la posición de destino (tercera torre). Se siguen tres reglas simples:
1. Solo se puede mover un disco a la vez.
2. Cada movimiento consiste en tomar el disco superior de una de las pilas y colocarlo encima de otra pila. En otras palabras, un disco solo se puede mover si es el disco superior de una pila.
3. No se puede colocar un disco más grande encima de un disco más pequeño.

Entendimiento del problema:

Situación actual

Se dispone de 3 torres y 3 discos de diferente tamaño apilados en la torre 1 en orden del grande abajo, mediano y pequeño arriba

Objetivo

el objetivo es trasladar los discos de la torre 1 a la torre 3 ubicando el disco más pequeño en la parte inferior de la torre

Condiciones

- Que en ningún momento un disco de mayor tamaño este sobre un disco de menor tamaño al moverlos.
- Solo se puede mover el disco que se encuentre en la parte superior de la torre.

Conceptualmente se pude representar el problema de la siguiente manera :

Tabla 2. Estados inicial y final (esperado)

Estado Inicial	Estado final
Torre 1: [disco 3(abajo), disco 2, disco 1(arriba)] Torre 2: [] Torre 3: []	Torre 1: [] Torre 2: [] Torre 3: [disco 1 (abajo), disco 2, disco 1(arriba)]

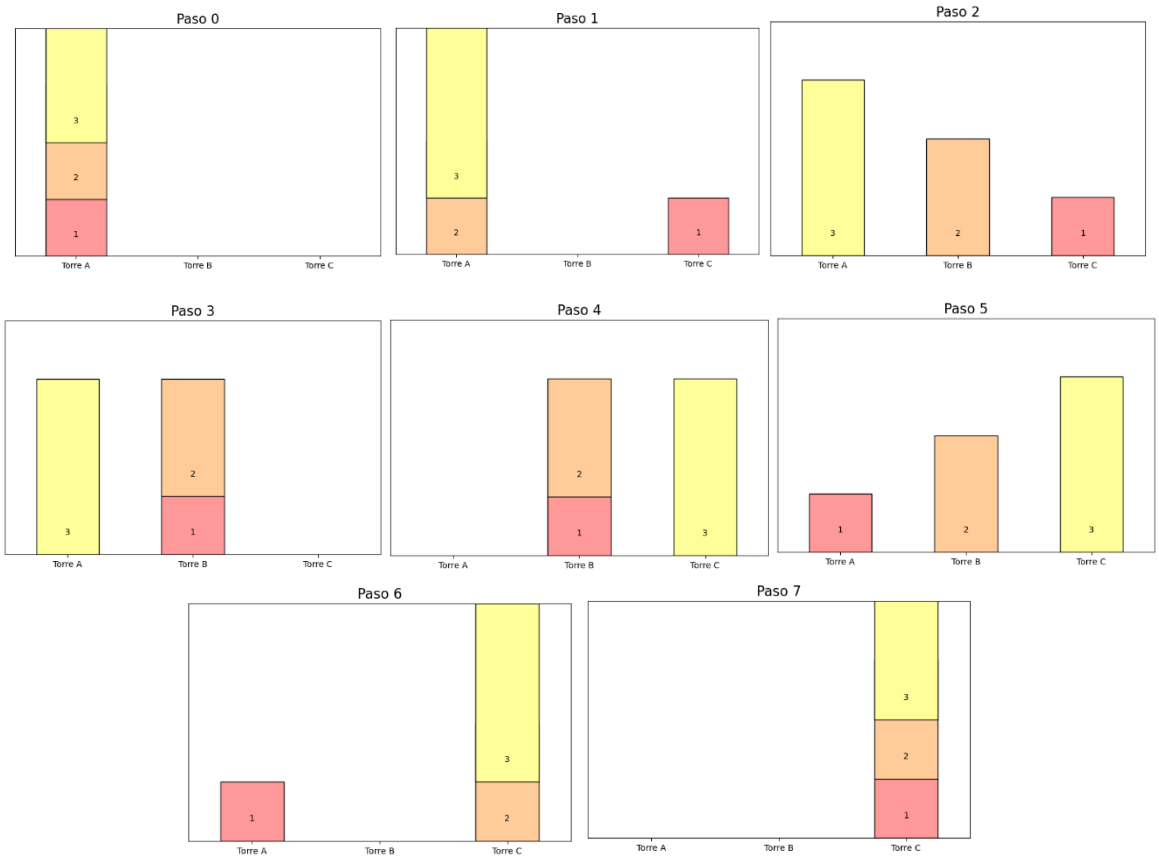
Solución:

En base a las instrucciones y a la revisión bibliográfica realizada[1] se puede identificar que el problema se puede resolver en 7 pasos (sin contar otras alternativas que se tomen) por lo tanto, no requiere una capacidad de cómputo excesiva. En ese sentido, como primera opción se optó por resolver el problema con el algoritmo de inteligencia artificial A*(A-star) que se encuentra en el repositorio de Github de Taller1\P3_Torres

1. A* (A-Star)

   Se plantea la presentación de resultados en dos esquemas diferentes:

   Representación gráfica en Matplotlib, en este esquema presenta de una manera gráfica y secuencial los pasos definidos por el algortimo para resolver el problema de la torre de hanoi.



[1] https://www.freecodecamp.org/espanol/news/como-resolver-el-problema-de-la-torre-de-hanoi-una-guia-ilustrada-del-algoritmo/

Por otro lado, se representa los estados de solución de la torre de hanoi por medio de grafos. Para esta representación es importante mencionar que la posición inicial definida por python para esta representación inicia desde cero.

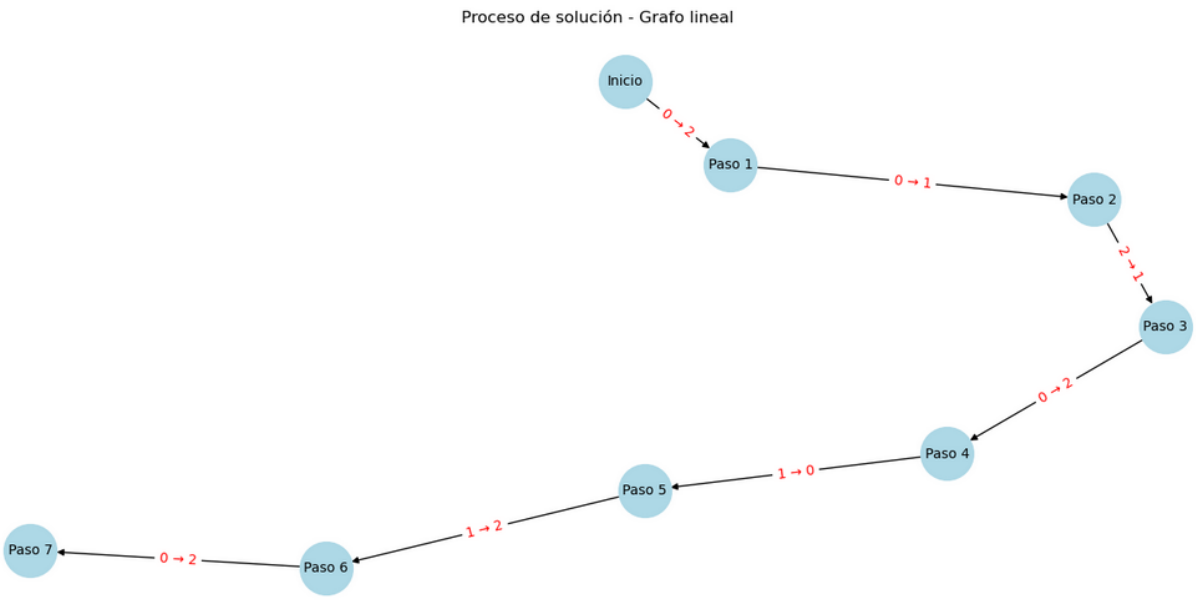


Ilustración 2. Solución por grafo-problema Torre de Hanoi

2. El segundo modelo que fue utilizado para resolver el ejercicio se lo planteó como un árbol de decisión de clasificación, por medio de tuplas donde tenemos un estado inicial que se define como: ((3,2,1), (), ()) y el estado final o punto de llegada lo definimos como: ((), (), (3,2,1)). Dentro del árbol de decisión vamos a tener estados de transición y estados donde los podríamos decir incorrectos donde la misma codificación del árbol da un step back y sigue analizando la siguiente variación. A continuación, se detalla una tabla con los estados del árbol.

Tabla 3. Estados (tuplas) de la solución del árbol de decisión- problema Torre de Hanoi

Estado	Tupla	Valor
Inicial	((3,2,1), (), ())	0
Final	((), (), (3,2,1))	1
Átipico	((x,x,x), (x,x,x), (x,x,x))	-1
Transición	((3,2), (), (1))	1
	((3), (2), (1))	1
	((3), (2,1), ())	1
	((), (2,1), (3))	1

((1), (2), (3))	1
((1), (), (3,2))	1

Al tener claro los puntos iniciales, finales, de transición e incorrectos graficamos el árbol para poder apreciar los caminos y decisiones que toma el árbol valga la redundancia.

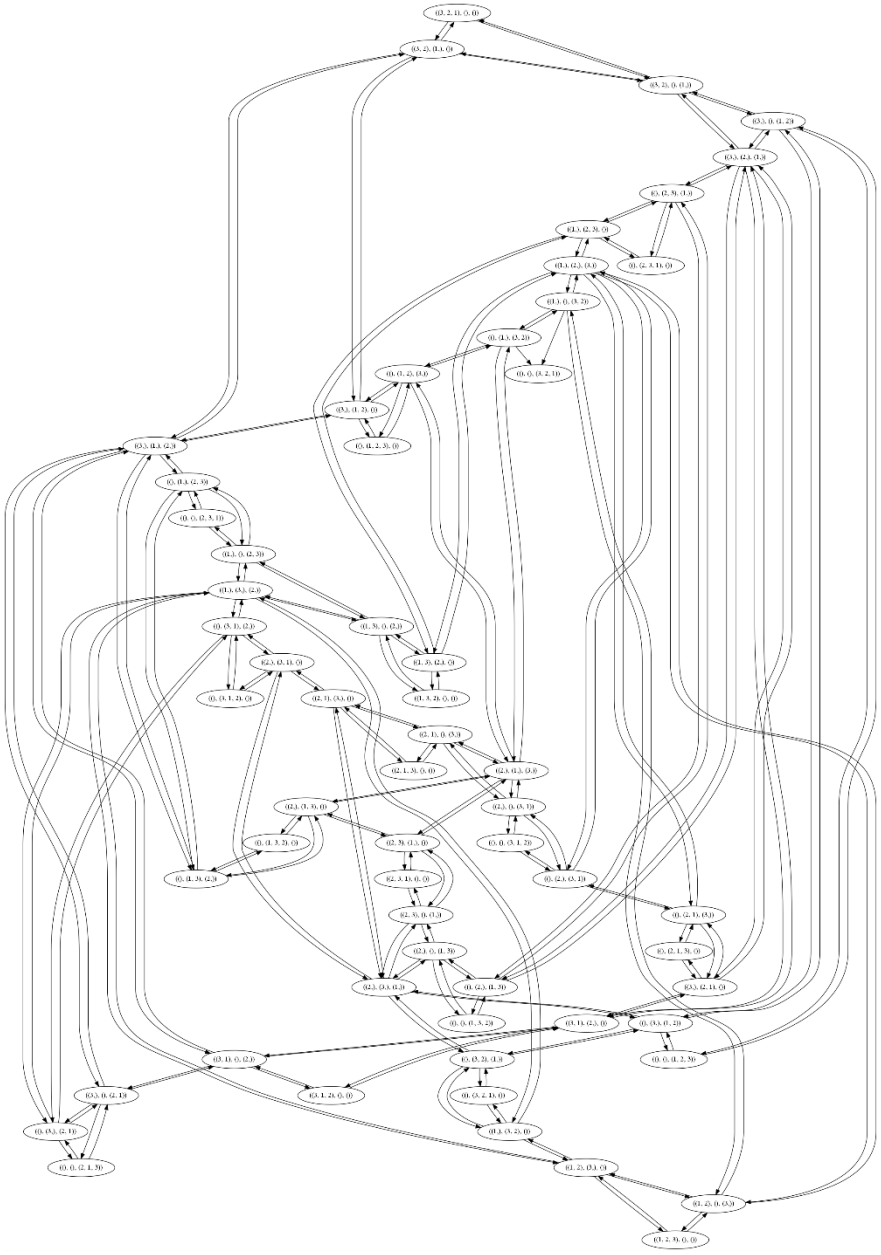


Ilustración 3. Solución árbol de decisión- problema Torre de Hanoi

Conclusiones:

- Debido a las limitadas variables los modelos utilizados no tienen una gran exigencia o diferencia en el tiempo de procesamiento.
- Para facilitar la visualización por medio de grafos, el script solamente muestra el camino que se toma para resolver el problema
- La visualización por medio de matplotlib presenta una interfaz secuencial facil de entender para un público que no tenga mucho conocimiento sobre cienca de datos. En este sentido, se presenta como una solución que se puede presentar a todo el público de manera didáctica, complementándose con una visión técnica (esquema de grafos).
- El árbol de decisión resulta un modelo también viable en el que se debe programar claramente los límites o los estados para que se genere correctamente el algoritmo.

2. USO DE LA INTELIGENCIA ARTIFICIAL – PLANEAMIENTO DE TAREAS

La planificación del taller y distribución de tareas para los integrantes del grupo se encuentra detallada en el siguiente enlace de ClickUp.

- **Link:** Taller 1 | USFQ (List)

3. LA EVOLUCIÓN DE LA INTELIGENCIA ARTIFICIAL

Los avances en inteligencia artificial mediante chips analógicos: eficiencia energética y nuevas fronteras

La inteligencia artificial enfrenta un desafío crítico: su insostenible consumo energético. Según IBM Research, modelos como GPT-3 generan mayores emisiones que un automóvil promedio durante su vida útil. Ante esto, los chips analógicos emergen como una solución revolucionaria, combinando eficiencia energética y capacidad de procesamiento paralelo para tareas de inferencia de IA.

Avances recientes en chips analógicos para IA

En 2025, IBM presentó un prototipo de chip analógico que utiliza memoria de cambio de fase para codificar pesos de redes neuronales directamente en el hardware. Este diseño ejecuta operaciones de multiplicación-acumulación (MAC) dentro de la memoria, eliminando la transferencia de datos entre componentes y reduciendo el consumo energético un 80-90% en comparación con chips digitales.

Los experimentos de IBM incluyeron reconocimiento de voz y transcripción, donde los chips

mostraron precisión equivalente a sistemas digitales, pero con mayor velocidad y menor uso de energía. Aunque actualmente soportan modelos con hasta 17 millones de parámetros (frente a los 175 mil millones de GPT-3), su escalabilidad mediante interconexión de múltiples chips sugiere potencial para aplicaciones más complejas.

Ventajas clave de los chips analógicos

1. Eficiencia energética:
   Al realizar cálculos directamente en memoria no volátil, evitan el movimiento constante de datos, principal fuente de gasto energético en chips tradicionales.
2. Procesamiento paralelo:
   Realizan múltiples operaciones MAC simultáneamente, acelerando tareas como el procesamiento de lenguaje natural o visión por computadora.
3. Escalabilidad:
   La integración de múltiples chips analógicos podría abordar modelos de IA masivos, superando limitaciones actuales de tamaño.
4. Latencia reducida:
   En aplicaciones como asistentes de voz, permiten respuestas instantáneas sin consumo energético en modo de espera.

Desafíos y perspectivas futuras

- La precisión en la computación analógica es inherentemente menos exacta que la digital, lo que exige nuevos algoritmos tolerantes a errores.
- La fabricación requiere de procesos especializados para producir memorias resistivas no volátiles (NVM) a gran escala.
- La integración de la combinación de componentes analógicos y digitales en un solo sistema híbrido sigue siendo un obstáculo técnico.

A pesar de estos retos, IBM proyecta que los chips analógicos podrían igualar el rendimiento de los digitales en la próxima década, especialmente si se optimizan los flujos de trabajo de entrenamiento de modelos. Empresas como Huawei también avanzan en chips de IA especializados, aunque con enfoques digitales, lo que subraya la diversidad de estrategias para abordar la crisis energética de la IA.

Conclusión

Los chips analógicos representan un salto cualitativo en hardware de IA, ofreciendo una alternativa sostenible para aplicaciones que van desde dispositivos IoT hasta centros de datos. Su éxito dependerá superar limitaciones técnicas, pero el trabajo de IBM y otros actores ya marca un camino viable hacia una IA más eficiente y escalable.

Referencias:

OLA GG. (n.d.). OLAGG App. Recuperado el 26 de April de 2025, from https://olagg.io/es/novedades/nuevos-chips-de-ibm-para-inteligencia-artificial

Rodríguez, H. (2025, February 25). Huawei mejora producción de chips de inteligencia artificial. La Ecuación Digital. https://www.laecuaciondigital.com/empresas/mercado/huawei-mejora-produccion-de-chips-de-inteligencia-artificial/

El Chip Analógico: La Revolución en la IA. (n.d.). Toolify.ai. Recuperado el 26 April de 2025, from https://www.toolify.ai/es/ai-news-es/el-chip-analgico-la-revolucin-en-la-ia-1536603

IBM Research. (2023, 23 de agosto). New analog AI chip design uses much less power for AI tasks. https://research.ibm.com/blog/analog-ai-chip-low-power