Universidad San Francisco de Quito

Grupo **06** **Fecha** **11/23/2025**

Integrantes	Tareas Individuales	Tareas Grupo
Santiago Acosta	Torres Hanoi	Completar el documento con la parte investigada y compartir reflexión de los videos
Juan Rodríguez	TSP Sol 1 (Grafos) - Notion	
Fernando Rodríguez	TSP Sol 2 (Held-Karp)	
Danilo Serrano	Problema Granjero – Ensayo	

Resolución de Ejercicios Propuestos

Código: https://github.com/Borreguin/USFQ-Wshop.git

Rama: Grupo_06

Link Notion:

TSP – Travelling Salesman Problem – Problema del Vendedor Viajante
Dada una lista de ubicaciones y las distancias entre cada par de ellas, el objetivo es encontrar la ruta más corta posible que visite cada ubicación exactamente una vez y, al finalizar, regrese a la ciudad de origen.

Solución 2: Algoritmo de Algoritmo de Held-Karp

Selección de la Estrategia

Se seleccionó el enfoque de Programación Dinámica con Máscaras de Bits como alternativa a la búsqueda en grafos tradicional. El objetivo principal de esta elección es optimizar el tiempo de ejecución mediante la técnica de memorización. Mientras que la solución basada en grafos recalcula rutas idénticas repetidamente, este enfoque almacena el costo mínimo de cada subconjunto de ciudades visitadas, garantizando que cada sub-problema se resuelva una sola vez.

Análisis Comparativo: Grafos (Backtracking) vs. Programación Dinámica

"O": Es la **etiqueta de clasificación** para la velocidad de un algoritmo, pero no en segundos, sino en **operaciones**. Es la **tendencia de crecimiento.**

“n”: Es el número de ciudades (en este problema)

1. **Eficiencia Temporal:**

   La solución basada en grafos tiene una complejidad de **O(n!).** Esto la hace inviable para **n > 12**. La solución propuesta reduce esto a **O(n^2 2^n),** permitiendo resolver problemas de hasta 20-22 ciudades en un tiempo razonable.

2. **Uso de Memoria:**

Aquí reside la desventaja de la Programación Dinámica. Mientras que el grafo (DFS) consume poca memoria (**O(n)** para la pila recursiva), la solución propuesta requiere almacenar una tabla de tamaño **n X 2^n**. Esto significa un intercambio: **sacrificamos memoria RAM para ganar velocidad de procesamiento**.

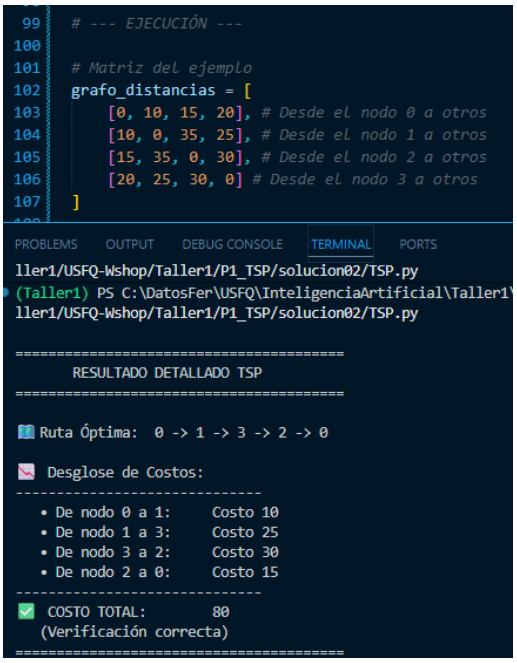
3. **Representación de Datos:**

   Se sustituyeron las estructuras de listas dinámicas por manipulación de bits (bitmasking). Esto reduce la sobrecarga de gestión de objetos del lenguaje, haciendo las verificaciones de "ciudades visitadas" casi instantáneas a nivel de procesador.

Estrategia Técnica Implementada:

1. **Representación del Grafo:** Se utilizó una **Matriz de Adyacencia**, donde la intersección de la fila **i** y la columna **j** representa la distancia entre la ciudad **i** y la **j**. La diagonal principal es 0, indicando que la distancia de un nodo a sí mismo es nula.
2. **Bitmasking (Máscaras de Bits):** En lugar de utilizar estructuras de datos pesadas (como listas o conjuntos) para rastrear las ciudades visitadas, se utilizó la **representación binaria de un número entero**. **Cada bit del número actúa como un interruptor (0 = no visitado, 1 = visitado).**
   a. ***Ejemplo:*** Una máscara `0101` indica que se han visitado las ciudades 0 y 2.
   b. ***Justificación:*** Esto permite realizar operaciones de verificación y actualización de estado en tiempo constante **O(1)** utilizando operadores a nivel de bits **(|, &, <<)**, optimizando drásticamente el uso de CPU.
3. **Memorización (Dynamic Programming):** Se implementó una tabla `memo[máscara][ciudad_actual]`. Antes de explorar una ruta recursiva, el algoritmo verifica si esa combinación específica de "ciudades visitadas + ubicación actual" ya fue resuelta anteriormente. Si es así, reutiliza el valor almacenado, evitando la explosión combinatoria típica de los algoritmos factoriales.

Salida:



El granjero, el lobo, la cabra y la col

Un día, un granjero fue al mercado y compró un lobo, una cabra y una col. Para regresar a casa debía cruzar un río. El granjero tenía una barca, pero en ella solo cabían él y uno de los tres elementos que había comprado.

- Las reglas del problema son:
- Si el lobo se queda solo con la cabra, el lobo se la come.
- Si la cabra se queda sola con la col, la cabra se la come.
- La barca solo puede ser manejada por el propio granjero.

El desafío consiste en lograr que el granjero cruce el río junto con sus compras, asegurándose de que ninguna quede en riesgo durante el proceso.

Informe: Resolución del problema del granjero, el lobo, la cabra y la col mediante técnicas de búsqueda

Introducción

El clásico acertijo del granjero, el lobo, la cabra y la col es un problema de espacio de estados donde un agente (el granjero) debe transportar a tres elementos a través de un río siguiendo reglas que evitan combinaciones peligrosas. Este problema es útil para ilustrar: representación de estados, construcción de un grafo implícito, algoritmos de búsqueda no informada e informada, y uso de estructuras de datos fundamentales en IA clásica.

Representación del problema

Un estado se representó como una tupla: (Granjero, Lobo, Cabra, Col). Cada elemento puede estar en L o R. Se definió una función de validez para descartar estados donde el lobo quede solo con la cabra sin el granjero o la cabra con la col. El grafo se genera de manera implícita mediante la función neighbors().

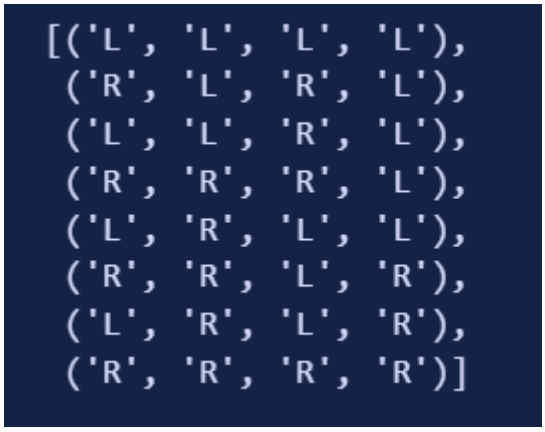
Métodos de solución implementados

- BFS: Encuentra la solución más corta usando una cola y un conjunto de visitados.
- DFS: Explora por profundidad usando una pila; no garantiza optimalidad.
- Backtracking: Explora exhaustivamente todo el espacio; encuentra soluciones pero es más lento.
- A*: Usa heurística (elementos restantes en L) y una cola de prioridad; (típicamente el más eficiente).

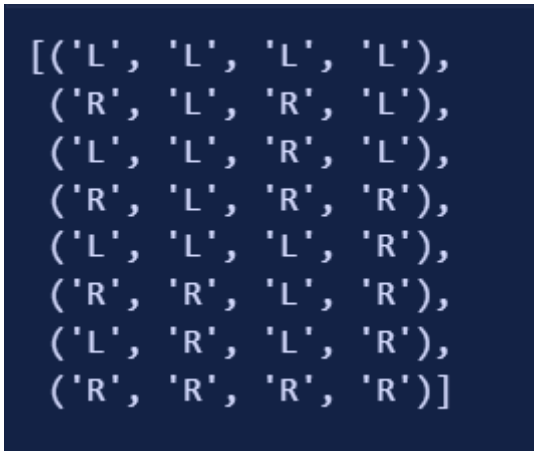
Comparación de desempeño

Se midieron longitud de solución y tiempo de cómputo.

BFS



DFS



Backtracking

```
[('L', 'L', 'L', 'L'),
 ('R', 'L', 'R', 'L'),
 ('L', 'L', 'R', 'L'),
 ('R', 'R', 'R', 'L'),
 ('L', 'R', 'L', 'L'),
 ('R', 'R', 'L', 'R'),
 ('L', 'R', 'L', 'R'),
 ('R', 'R', 'R', 'R')]
```

Conclusiones

- Resultado General:

```
BFS: 8 pasos
DFS: 8 pasos
Backtracking: 8 pasos
Grafo + BFS: 8 pasos
Árbol de búsqueda: 8 pasos
```

- BFS: solución más corta.
- DFS: solución válida pero no óptima.
- Backtracking: método correcto pero costoso.
- El uso de tuplas, colas, pilas, sets y colas de prioridad fue esencial.

La Torre de Hanoi

El objetivo es mover una pila completa de discos desde la torre de origen (primera torre) hasta la torre de destino (tercera torre). Para lograrlo, se deben seguir tres reglas simples:

- Solo se puede mover un disco a la vez.
- Cada movimiento consiste en tomar el disco superior de una de las pilas y colocarlo sobre otra. Es decir, un disco solo puede moverse si es el que está arriba en su torre.
- No está permitido colocar un disco más grande sobre uno más pequeño.

Clasificación

El problema es accesible ya que se conoce exactamente que poste está cada disco y no hay información oculta, toda acción, mover un disco, es determinista, es secuencial, debido a que cada movimiento afecta el estado siguiente y condiciona lo que podrás

hacer después, es estático, es decir el entorno no cambia al mover un disco y finalmente es discreto ya que El número de estados posibles se los puede contar.

Representación

La representación de este problema es el estado de la pila la misma que se puede representar como una lista o pila de discos por torre. Por ejemplo, si hay tres discos (1, 2, 3), la representación sería (A= [3,2,1B= [], C= []).

Para resolver el problema debemos tener en cuenta que el primer paso es llevar el disco base a la torre C de la siguiente forma:

1. Para mover el disco más grande (3) a la torre C, para esto, se debe sacar de encima los dos discos 1,2, (n-1) discos, y llevarlos a la torre auxiliar B para tener libre la torre C, se debe tener la forma *(A = [3], B= [2,1], C= [])*.
2. Sacar el disco 3 de A a C, *(A= [], B= [2,1], C= [3])*.
3. Para reconstruir la pila de necesita mover los discos 1 y 2, (n-1) discos, a la torre C, *(A= [], B= [], C= [3,2,1])*.

Para mover los discos 1, 2 a la torre auxiliar sin violar las reglas del problema se debe realizar 3 movimientos tomados de la siguiente forma:

1. Disco 1 origen=A, destino =C.
2. Disco 2 origen=A, destino =B.
3. Disco 1 origen=C, destino =B

En esta etapa la torre C actúa como auxiliar debido a que el objetivo es tener en B la estructura *B= [2,1]*.

La etapa media es siempre mover el disco base de origen= A, a destino = C.

Para el caso en el que volvemos a armar la pila una vez que el disco 3 este en la torre C tenemos:

1. Disco 1 origen=B, destino =A.
2. Disco 2 origen=B, destino =C.
3. Disco 1 origen=A, destino =C

En este tenemos a la torre A como auxiliar siendo que origen es B y destino es C para resolver el subproblema. Como se puede observar las torres desempeñan un rol pasando de origen a destino y a ser auxiliares dependiendo del movimiento esto para respetar las reglas del problema.

Las etapas para pasar los n-1 discos se repiten 2 veces, tanto para liberar al disco 3 como para armar la pila, es por esto por lo que es un problema recursivo.

El número de movimientos para 3 discos son los siguientes:

Discos	Movimientos
1	4
2	2
3	1

Si aumentamos más discos y consideramos al disco base (el más grande) como n

Discos	Movimientos	Representación
n	1	2^0
n-1	2	2^1
n-2	4	2^2
n-3	8	2^3
n-4	16	2^4
.		.
.		.
……..		.
Disco más pequeño		2^{n-1}

La suma me movimientos será 1+2+4+8+…….2^{n-1}, nos dará el número minio de movimientos

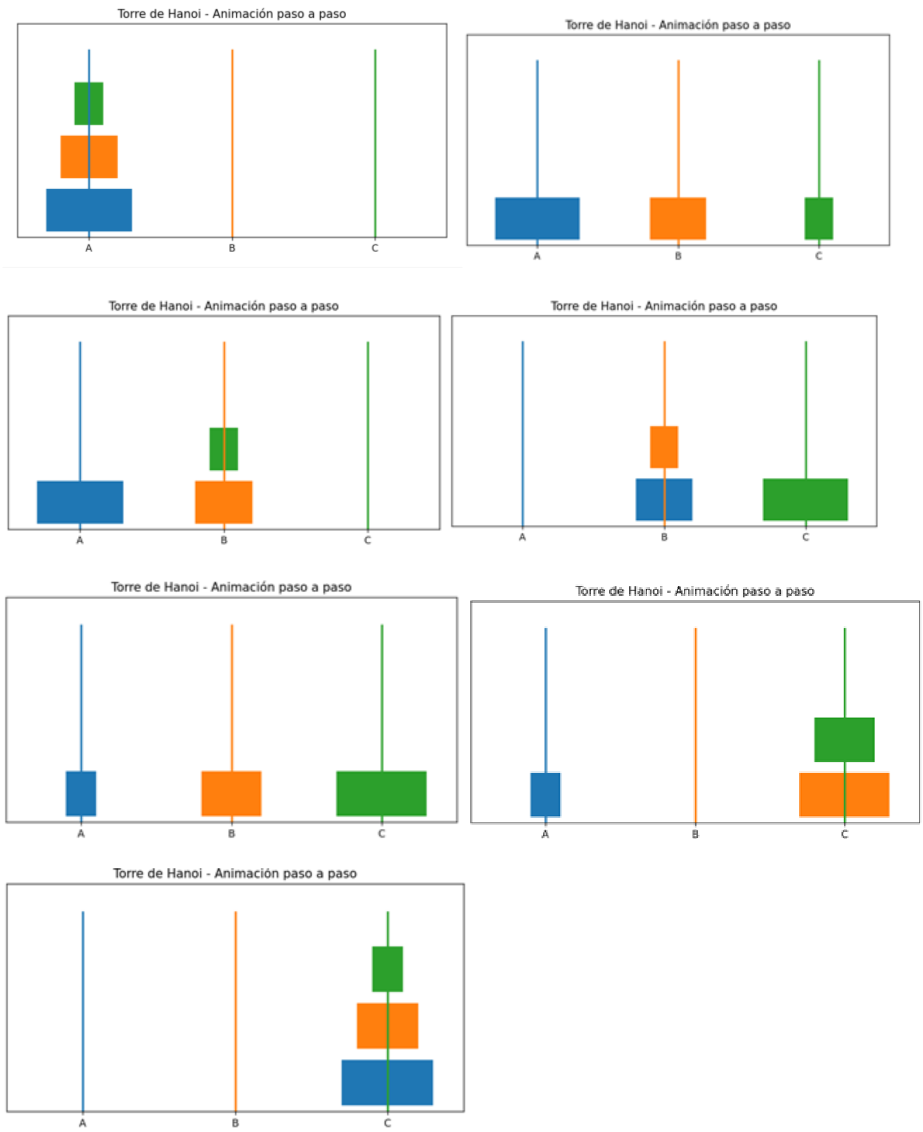
$$\sum_{i=0}^{n-1} 2^i = 2^n - 1$$

Por tanto, dependiendo de cuantos discos tengamos en la torre de origen los movimientos crecen de forma exponencial, y así también el tiempo.

Pseudocódigo

```
hanoi(n, origen, auxiliar, destino):
    si n == 1:
        mover origen → destino
        retornar
    hanoi(n-1, origen, destino, auxiliar)
    mover origen → destino
    hanoi(n-1, auxiliar, origen, destino)
```

Animación



CONCLUSIONES

1.- El problema de las torres de Hanoi presenta recursividad es decir se tiene que resolver un problema dentro de otro problema en este caso pasar los discos que estén encima de la base a una torre auxiliar y viceversa para volver a armar la pila en la torre destino.

2.- El problema presenta una naturaleza exponencial en su resolución por número discos en movimientos, recursividad y tiempo.

3.- La dificultad de representarlo es baja, se lo puede representar mediante tuplas o listas.

Link distribución en Notion

https://www.notion.so/TALLER-1-2ba8ab982b7a80de917bc96d9bf18d38?source=copy_link