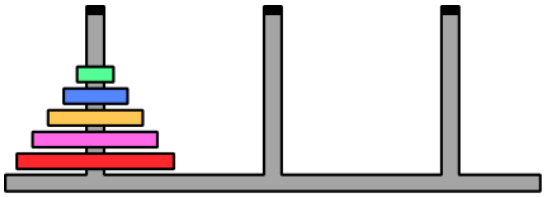
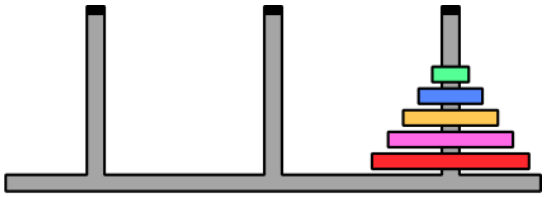
Carlos Ramón Méndez Tineo DNI: 96103939

TP1: Algoritmos de búsqueda en Torre de Hanoi.

En clase presentamos el problema de la torre de Hanoi. Además, vimos diferentes algoritmos de búsqueda que nos permitieron resolver este problema. Para este trabajo práctico, deberán implementar un método de búsqueda para resolver con 5 discos, del estado inicial y objetivo que se observa en la siguiente imagen:



Estado inicial



Estado final

Tareas y preguntas a resolver:

1. ¿Cuáles son los PEAS de este problema? (Performance, Environment, Actuators, Sensors)

   Performance o la medida de rendimiento para este problema sería la colocación correcta (respetando las reglas) de los discos.

   Entorno: tablero de juego compuesto por tres varillas.

   Actuators: Los movimientos permitidos por el juego, mover entre varillas, desde la fuente (o de partida) a la auxiliar (o intermedia), desde la fuente a la de destino, desde la auxiliar a la fuente, desde la auxiliar a la de destino, desde la de destino a la auxiliar o desde la de destino a la fuente.

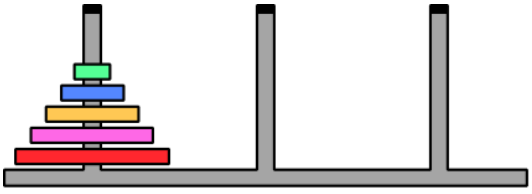
   Sensors: Proceso de evaluar la posición en las varillas de los discos.

2. **¿Cuáles son las propiedades del entorno de trabajo?**

   Sistema de varillas: 3 en total

   Discos: 5

   Como se muestra en la figura



Solo se puede mover un disco a la vez.

Cada movimiento consiste en tomar el disco superior de una de las varillas y colocarlo encima de otra varilla.

Ningún disco puede colocarse encima de un disco más pequeño.

3. **En el contexto de este problema, establezca cuáles son los: estado, espacio de estados, árbol de búsqueda, nodo de búsqueda, objetivo, acción y frontera.**

   Estado representa una configuración posible de los discos en alguna de las varillas (para una interacción dada)

   Espacio de estados: Representa el conjunto total de combinaciones posibles entre discos y varillas.

4. **Implemente algún método de búsqueda. Puedes elegir cualquiera menos búsqueda en anchura primero (el desarrollado en clase). Sos libre de elegir cualquiera de los vistos en clases, o inclusive buscar nuevos.**

   Se implementó un algoritmo recursivo, para encontrar los movimientos válidos que permitan pasar del estado inicial al estado final. Recursivo porque para resolver el problema llama a versiones más pequeñas del mismo proceso.

5. **¿Qué complejidad en tiempo y memoria tiene el algoritmo elegido?**

   En términos de tiempo, el algoritmo seleccionado hace llamadas sucesivas al mismo método. Luego para tres discos se tiene

   $T(3) = T(2) + T(1)$

   $T(4) = T(3) + T(2)$

   $T(5) = T(4) + T(3)$

   Por lo que, para el caso de n, se realizan dos llamadas recursivas. Se deduce que la complejidad temporal se duplica con cada iteración y es **$o(2^n)$**

   En términos de memoria no tiene impacto significativo comparado con los algoritmos de búsqueda considerados en clase, principalmente porque no guarda los estados de transición.

6. **A nivel implementación, ¿qué tiempo y memoria ocupa el algoritmo? (Se recomienda correr 10 veces y calcular promedio y desvío estándar de las métricas).**

Se diseñó un código en Python (.ipynb) donde se iteró 10 veces – variando la cantidad de discos (hasta un máximo de 14) -, obteniendo los siguientes resultados,

```
data_frame.describe()
```

	1 Disco	2 Discos	3 Discos	4 Discos	5 Discos	6 Discos	7 Discos	8 Discos	9 Discos	10 Discos	11 Discos	12 Discos	13 Discos	14 Discos
count	9.000000	9.000000	9.000000	9.000000	9.000000	9.000000	9.000000	9.000000	9.000000	9.000000	9.000000	9.000000	9.000000	9.000000
mean	0.000470	0.000170	0.001075	0.009855	0.017647	0.038028	0.087519	0.252357	0.239399	0.755965	2.005253	3.360738	8.403060	13.881593
std	0.001231	0.000024	0.001978	0.024417	0.037346	0.073342	0.118088	0.332488	0.211504	0.900525	2.103501	2.999731	2.706178	2.839780
min	0.000034	0.000108	0.000256	0.000708	0.001211	0.002459	0.017957	0.032644	0.071798	0.184810	0.277159	0.746385	2.710091	10.266616
25%	0.000051	0.000170	0.000424	0.000953	0.001946	0.004412	0.021138	0.038264	0.117477	0.243228	0.496187	1.080402	7.581109	12.338054
50%	0.000054	0.000174	0.000437	0.001016	0.002133	0.004836	0.025268	0.040692	0.187797	0.285204	0.558856	1.930938	8.437683	13.187500
75%	0.000060	0.000182	0.000462	0.001195	0.014827	0.026202	0.083988	0.572199	0.217263	0.577106	4.390447	5.833011	10.409825	16.683605
max	0.003752	0.000188	0.006347	0.074746	0.115937	0.228138	0.380466	0.885057	0.732449	2.385626	5.197632	9.284750	11.156437	18.509912

7. **Si la solución óptima es 2^k-1 movimientos con *k* igual al número de discos. Qué tan lejos está la solución del algoritmo implementado de esta solución óptima (se recomienda correr al menos 10 veces y usar el promedio de trayecto usado).**

   El algoritmo recursivo usado representa la opción óptima realiza la cantidad mínima de movimientos para cada n (número de discos).

```
data_movimientos = pd.DataFrame(serie_movimientos,columns=cols)
data_movimientos.head()
```

	1 Disco	2 Discos	3 Discos	4 Discos	5 Discos	6 Discos	7 Discos	8 Discos	9 Discos	10 Discos	11 Discos	12 Discos	13 Discos	14 Discos
0	1	3	7	15	31	63	127	255	511	1023	2047	4095	8191	16383
1	1	3	7	15	31	63	127	255	511	1023	2047	4095	8191	16383
2	1	3	7	15	31	63	127	255	511	1023	2047	4095	8191	16383
3	1	3	7	15	31	63	127	255	511	1023	2047	4095	8191	16383
4	1	3	7	15	31	63	127	255	511	1023	2047	4095	8191	16383