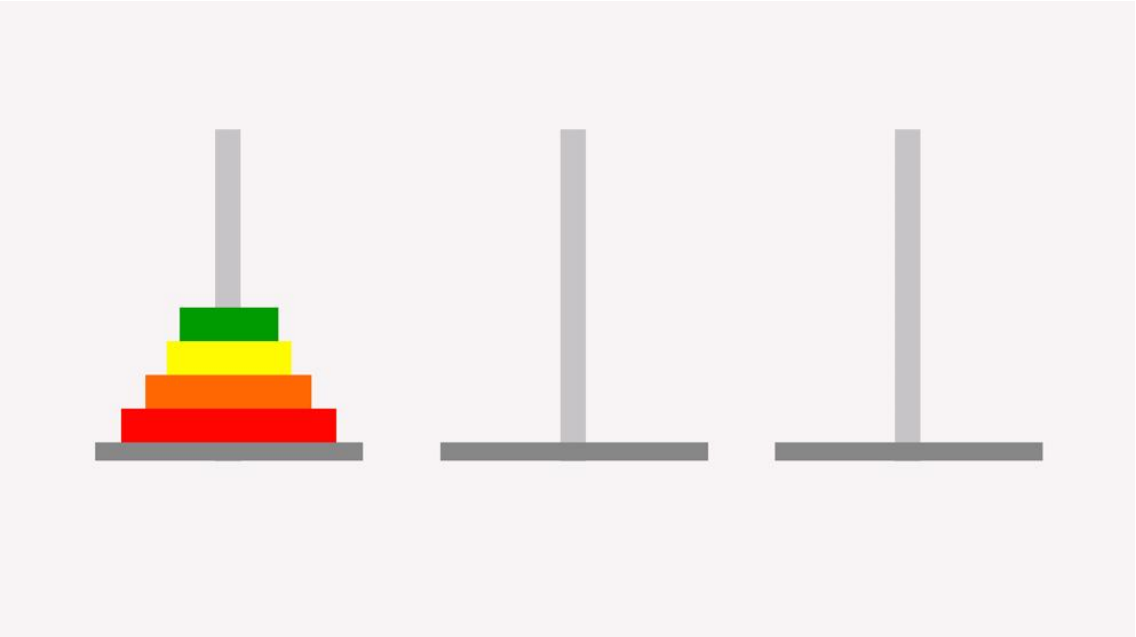
YERAY BELLANCO CASARES

Proyecto 1 – Hanoi Puzzle

DESCRIPCIÓN DEL PROBLEMA



Según la leyenda, unos monjes budistas estaban cambiando de lugar 64 discos y que cuando terminen será el fin del mundo. Esos discos están apilados de forma que los discos mas pequeños están encima de los mas grandes y los monjes no tienen permitido colocar nunca un disco grande encima de uno pequeño.

Este puzzle en su forma clásica, consiste en que tenemos tres soportes como podemos observar en la imagen de arriba. El objetivo es pasar todos los discos que tenemos en un soporte a otro soporte, pero siguiendo una serie de reglas:

- Solo puedes mover un disco en cada movimiento
- No puedes colocar un disco encima de otro que sea mas pequeño.
- Siempre mueves el disco que este mas arriba en un soporte.

Es un puzzle que suele verse en asignaturas como programación para dar el tema de recursividad.

Y por ultimo, haciendo caso a la leyenda de los monjes budistas se necesitarían 2^64-1 movimientos lo que equivale aproximadamente a 584000 millones de años, muy superior al tiempo de vida que se le da al universo.

SCRIPT UTILIZADOS

- **Main-template.R**: Contiene el script que resuelve el problema. En este archivo básicamente cargamos el script del problema y lo metemos en los distintos algoritmos para ver los resultados.

```
[1] 6 6 6
                                      name solution    runtime length cost expanded max_depth
1                        Breadth First Search    FALSE  1.83 secs     -1   -1     1000         4
2                      Breadth First Search + GS     TRUE  0.80 secs      5    5      207         5
3                          Depth First Search    FALSE  7.60 secs     -1   -1     2500      2499
4                        Depth First Search + GS     TRUE  0.42 secs     45   45       46        44
5                  Depth First Search - Limit=6    FALSE  2.52 secs     -1   -1     4000         7
6            Depth First Search - Limit=6 + GS    FALSE  0.88 secs     -1   -1      757         7
7                Depth First Search - Limit=10    FALSE  2.24 secs     -1   -1     3500        11
8           Depth First Search - Limit=10 + GS     TRUE  0.33 secs      9    9      370        11
9       Iterative Deeping Search - Iteration:11    FALSE 12.16 secs     -1   -1        0      -Inf
10 Iterative Deeping Search - Iteration:7 + GS     TRUE  2.35 secs      7    7      227         8
   max_frontier
1          9756
2           116
3         27491
4           137
5            68
6            33
7           112
8            52
9          -Inf
10           36
```

- **Problem-template.R:** Contiene los métodos que resuelven el problema. Los métodos principales son:
  - **Initialize.problem:** Funcion que tiene por parámetros la barra (soporte) y el disco. Cuenta con los siguientes atributos:
    - **Nombre de el problema:** Torres de Hanoi y numero de discos y soportes utilizados.
    - **Estado inicial:** Carga todos los valores a 1.
    - **Estado Final:** Los valores son iguales al numero del soporte.
    - **Acciones posibles:** Permutaciones del conjunto de soportes en grupos de 2. Contiene todos los movimientos posibles.
  - **Is.applicable:** Tiene por parámetros una **acción** que se define con un soporte de origen y un soporte de destino **,** un **estado** que es un vector que indica en que soporte se encuentra un disco y el **problema** que cuenta con los atributos mencionados en Initialize.problem. El objetivo de esta función es comprobar si se puede realizar el movimiento desde el soporte de origen al soporte destino.
  - **Effect:** Tiene por parámetros acción y estado. El objetivo de esta función es que cuando is.applicable devuelva TRUE crear una copia del estado y modificar el soporte del disco que este en la parte superior del soporte origen para moverlo al soporte destino.
  - **Is.final.state:** Invoca a la función all().

- **Algorithms:** Implementación de los algoritmos y funciones para analizar los resultados. Los algoritmos utilizados son:
  - Breadth first search
  - Depth first search
  - Iterative deepening search

PRUEBAS UNITARIAS

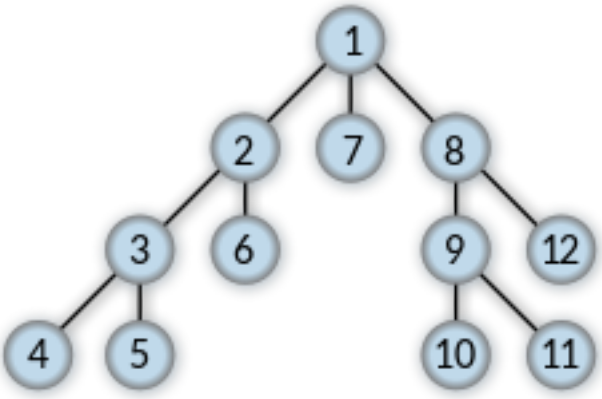
Utilizando 3 estados de prueba y un estado final e inicializando el problema con 6 barras y 3 discos. Comprobamos mediante pruebas unitarias:

- Si un movimiento puede (si es posible) realizarse o no.
- Si un movimiento se realiza o no.
- Verificar si un estado es final o no.

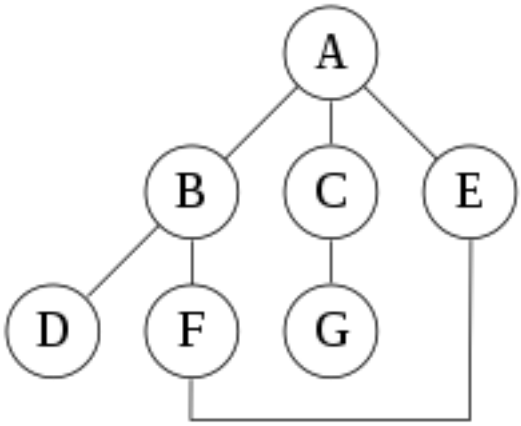
ANALISIS DE LOS RESULTADOS

```
[1] 6 6 6
                                      name solution    runtime length cost expanded max_depth
1                          Breadth First Search    FALSE  1.83 secs     -1   -1     1000         4
2                       Breadth First Search + GS     TRUE  0.80 secs      5    5      207         5
3                            Depth First Search    FALSE  7.60 secs     -1   -1     2500      2499
4                         Depth First Search + GS     TRUE  0.42 secs     45   45       46        44
5                    Depth First Search - Limit=6    FALSE  2.52 secs     -1   -1     4000         7
6               Depth First Search - Limit=6 + GS    FALSE  0.88 secs     -1   -1      757         7
7                   Depth First Search - Limit=10    FALSE  2.24 secs     -1   -1     3500        11
8              Depth First Search - Limit=10 + GS     TRUE  0.33 secs      9    9      370        11
9         Iterative Deeping Search - Iteration:11    FALSE 12.16 secs     -1   -1        0      -Inf
10 Iterative Deeping Search - Iteration:7 + GS     TRUE  2.35 secs      7    7      227         8
   max_frontier
1          9756
2           116
3         27491
4           137
5            68
6            33
7           112
8            52
9          -Inf
10           36
```

- El algoritmo mas rápido es el **Depth First Search** o búsqueda en profundidad. Expande, todos y cada uno de los nodos que va localizando, en un camino concreto. Cuando no quedan mas nodos en este camino regresa (backtracking) y repite el proceso con cada uno de los nodos hermanos que ha procesado. Podemos ver en la tabla que quien mas nodos expande es este algoritmo.



- El algoritmo mas lento es el **Iterative Deeping Search** en cada iteración visita los nodos en el árbol de búsqueda en el mismo orden que una búsqueda en profundidad, pero el orden con que los nodos son visitados finalmente se corresponde con la búsqueda en anchura.



- **Breadth first search** o el algoritmo de búsqueda en anchura recorre un árbol de búsqueda comenzando por la raíz y explorando todos los vecinos de este nodo. Cada uno de los vecinos explora sus respectivos vecinos adyacentes, hasta que el árbol es totalmente recorrido.