

Representación de un estado.

Como la estructura que utilizo para representar los estados está compuesta de la siguiente forma

```
//struct del tablero
typedef struct EstadoTablero EstadoTablero;

struct EstadoTablero{
    int**tablero;//este es el tablero
    //estas dos variables son para tener alma
    int caballoX;
    int caballoY;
    //
    int peonesRestantes;//almacena cuantos p
    int estadoActual;//esta es la id del est
    int estadoAnterior;//esta es la id del e
    char * transicion; //el movimiento reali
};
```

Una estructura de un estado cualquiera seria de la forma:

Tablero=[[0,0,0,0,0],[1,1,1,0,0],[0,2,0,0,0],[0,0,0,0,0],[0,0,0,0,0]] donde los 0 representan espacios vacíos, los 1's representan peones y el 2 representa al caballo

caballoX=2

caballoY=1

peonesRestantes=3

estadoActual =0 si es el estado inicial, en otro caso dependerá de cuantos movimientos se hayan realizado

estadoAnterior dependerá también si este estado es el inicial

transición es el movimiento realizado en caso de este ser el estado inicial no tendrá ningún movimiento realizado

Representación del estado inicial y sus valores en específico, y por qué será el estado inicial.

Ya que al momento de ejecutar el programa se le solicita al usuario que ingrese el caballo y los peones el estado inicial dependerá del usuario, sin embargo, si utilizamos como estado inicial un ejemplo entregado en el PDF de la prueba como puede ser el caso de la figura 2.


0	1	0	0	0
1	0	0	1	0
0	0	1	0	0
0		0	0	1
0	0	0	0	0

Figura 2: Solución posible partiendo desde (3,1)

En este caso la estructura del estado inicial vendría dado de la siguiente forma

Tablero=[[0,1,0,0,0],[1,0,0,1,0],[0,0,1,0,0],[0,2,0,0,1],[0,0,0,0,0]] //donde los 0 representan espacios vacíos, los 1's representan peones y el 2 representa al caballo

caballoX=3 //ya que la coordenada del caballo es 3,1, y esta variable almacena su posición en X

caballoY=1 //ya que la coordenada del caballo es 3,1, y esta variable almacena su posición en Y

peonesRestantes=5

estadoActual =0 //ya que es el estado inicial

estadoAnterior //no tiene valor ya que al ser el inicial no tiene un estado anterior

transición ="" //como es el estado inicial no posee transición

este es el estado inicial ya que es el ingresado por el usuario siguiendo el ejemplo de la figura 2

Representación del estado final y sus valores en específico, y por qué será el estado final.

En este caso la estructura del estado final vendría dado de la siguiente forma

Tablero=[[0,0,0,0,0],[0,0,0,0,0],[0,0,0,0,0],[0,2,0,0,0],[0,0,0,0,0]] //donde los 0 representan espacios vacíos, los 1's representan peones(como es el estado final, no quedan o simplemente no se agregaron peones) y el 2 representa al caballo

caballoX=3 //ya que la coordenada del caballo es 3,1, y esta variable almacena su posición en X

caballoY=1 //ya que la coordenada del caballo es 3,1, y esta variable almacena su posición en Y

peonesRestantes=0

estadoActual = //ya que es el estado final y depende de cuantos movimientos se hayan realizado no se sabe cuantos estados se generaron sin embargo tomara un valor 0 en caso de que no se agreguen peones

estadoAnterior //su valor dependerá del camino que se haya recorrido para llegar a este estado.

transición = //como es el estado final la transición dependerá del movimiento realizado

este es el estado final ya que se cumple que no queden peones, independiente de si no se agregaron estos o no.

Transiciones, explicando cómo pasa de un estado E_i a un estado E_j con esa transición, justificar la existencia de esta transición.

Las transiciones vienen dadas por los posibles movimientos del caballo dentro de un tablero de ajedrez, el caballo posee un movimiento en L el cual le permitiría moverse a una de un total de 8 casillas distintas como se ve en la figura 1 del PDF de la prueba (cada casilla con una x recibe un numero el cual abajo se explicará que nombre toma ese movimiento dentro del código)

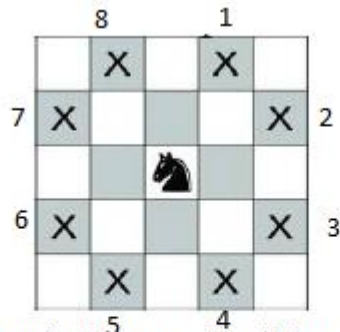


Figura 1: Movimientos posibles del caballo ubicado en (2,2)

Aquí se menciona que nombre toma cada movimiento dentro del código

1	arribaDER1
2	arribaDER2
3	abajoDER1
4	abajoDER2
5	abajoIZQ2
6	abajoIZQ1
7	arribaIZQ2
8	arribaIZQ1

Sin embargo, estos movimientos se ven limitados dependiendo de la posición actual del caballo ya que no puede salirse del tablero.

Esto implica que las transiciones para pasar de un estado a otro están limitadas por los movimientos que pueda realizar el caballo en un momento cualquiera.

El algoritmo utilizado de búsqueda en espacio de estados, señalando si es Búsqueda en Anchura o profundidad, justificando el tipo de búsqueda realizada.

El algoritmo utilizado es Búsqueda en Anchura, debido a que fui creando estados nuevos pero siempre revisaba el primer estado en ser creado, trabajando de esta forma mi array de abiertos de la manera FIFO (First In, First Out)

Código que obtiene la solución.

El código funciona de la siguiente forma:

-Primero se le solicita la posición del caballo al usuario

Se verifica si es valida, en caso de no serla se termina el programa

En caso de ser una posición valida el programa prosigue

-luego se le solicita la cantidad de peones que agregara

Si la cantidad de peones es menor o igual a 0 o mayor que 24 entonces termina el ejercicio dándole un mensaje al usuario de porque termino

-se le pide al usuario agregar cada peón en su respectiva coordenada

-una vez el usuario haya terminado de agregar el tablero inicial, el programa ejecutara movimientos del caballo generando todos los estados posibles trabajando con el array de abiertos de la forma FIFO

-con esto el programa buscara la solución al tablero inicial entregando luego los pasos a realizar.

El esqueleto del programa en pseudocódigo se ve de la siguiente forma

```
estructura{
}

movimientos
verificaciones de movimientos

main{
    crear lista soluciones
    creacion del tablero inicial
    ciclo iterativo while{
        verificar movimiento
        movimiento
        agregar a soluciones
    }
    mostrar pasos
}
```