Pregunta 2

Implementación:

Para el desarrollo de este problema se utilizó un código de búsqueda en espacio de estados por anchura, ya que necesitamos conocer el "camino" recorrido para llegar a la solución, los estados son una mejor opción para identificar cada paso que se recorrió para llegar a la solución final y un método de anchura para así conocer la solución que resuelve el problema en la menor cantidad de pasos.

En este caso el estado inicial estará representado de la siguiente manera:

- Tablero: Esta será una matriz de 5x5 que representa el tablero actual, en este caso, será el tablero leído desde un archivo de texto por la función entregada por los profesores. Este tablero tendrá 3 valores diferentes, 1 representado la posición donde se encuentra un peón, 2 representado donde se encuentra el caballo y 0 representado las casillas vacías.

- PosX: Variable de tipo entero que indica la posición en x (columna) que tiene el caballo inicialmente en este caso. El valor de esta variable dependerá del tablero inicial entregado.

- PosY: Variable de tipo entero que indica la posición en y (fila) que tiene el caballo inicialmente en este caso. El valor de esta variable dependerá del tablero inicial entregado.

- Id: Variable para identificar cada estado. En este caso tendrá el valor de 0 por ser el estado inicial del problema.

- Id estado anterior: Variable para identificar de que estado proviene dicho estado. En este caos tendrá el valor de 0, ya que no hay un estado anterior al estado inicial.

Mientras que el estado final estará representado de la siguiente manera:

- Tablero: El tablero del estado final no deberá tener peones (1's) dentro de él, ya que esto significa que logro resolver el problema ("comerse" a todos los peones), esta condición será comprobada externamente por una función llamada **peonesRestantes** más adelante.

- PosX: Deberá ser la posición en x (columna) del ultimo peón que se "comió", por lo que su valor depende del problema.

- PosY: Deberá ser la posición en y (fila) del ultimo peón que se "comió", por lo que su valor depende del problema.

- Id: Deberá ser el identificador del estado final (solución), por lo que su valor depende del problema.

- Id estado anterior: Deberá ser el identificador del estado que generó al estado final, por lo que su valor depende del problema.

El algoritmo utilizado para la resolución de este problema comienza leyendo un archivo de texto que contiene cada tablero para este problema, esta tera será realizada por la función entregada por los profesores. Luego por medio de la función **Bee** comenzamos la resolución del problema, como argumentos esta función recibe la variable **mapa** que es una matriz de caracteres obtenida al leer el archivo de texto, **colCaballo** que contiene la columna en la que se encuentra inicialmente el caballo y **filaCaballo** que contiene la fila en la que se encuentra el caballo inicialmente.

Dentro de la función **Bee** comenzamos definiendo la variable **idNuevoEstado** que nos servirá para establecer el id que tendrá cada estado, la variable **cantidadAbiertos** iniciada en 0, junto con una lista de estados abiertos llamada **abiertos** que almacenará todos los estados por revisar y la variable **cantidad Cerrados** iniciada en 0, junto con una lista de estados cerrados llamada **cerrados** que almacenará todos los estados ya revisados.

Después se creará el estado inicial por medio de la función **tableroInicial**, esta función recibe como argumentos la matriz de caracteres **mapa** que contiene el tablero leído, **colCaballo** y **filaCaballo**, esta función se encargar de crear el estado inicial previamente explicado y retornarlo para ser almacenado en la variable **estadoInicial**, antes de comenzar a buscar la respuesta, verificamos que el tablero leído tenga peones para ser comidos dentro de él, en caso de no tenerlos, se saldrá de la función y se entrega un mensaje indicando que no existen peones en el tablero inicial.

Una vez creado el estado inicial, este será agregado a la lista de estados abiertos por medio de la función **agregarEstado**, retornando una nueva lista con el estado agregado, ahora que el estado se encuentra dentro de la lista, comenzará un ciclo while que se ejecutará mientras queden estados abiertos por revisar, dentro del ciclo comenzaremos extrayendo el primer estado de la lista de estados abiertos con la función **extraerEstado**, almacenándolo en la variable **estadoActual** y a su vez agregar el estado a la lista de estados cerrados con **agregarEstado**.

Ahora comenzaremos a realizar todos los movimientos que puede realizar el caballo, estos movimientos son **arribaIzquierda** (2 casillas arriba y una a la izquierda), **arribaDerecha** (2 casillas arriba y una a la derecha), **izquierdaArriba** (2 casillas a la izquierda y una arriba), **derechaArriba** (2 casillas a la derecha y una arriba), **abajoIzquierda** (2 casillas abajo y una a la izquierda), **abajoDerecha** (2 casillas arriba y una a la derecha), **izquierdaAbajo** (2 casillas a la izquierda y una abajo), **derechaAbajo** (2 casillas a la derecha y una abajo); cada una de estas funciones (movimientos) recibe como argumento la variable **nuevoEstado** y retorna un nuevo estado en donde el nuevo tablero tendrá marca en la nueva posición del caballo (en caso de haber un 1 o un 0, este será cambiado por un 2. En caso de ya haber un 2 lo remplazará con un 3, esto para permitir que el caballo pase una vez más por una casilla por la que ya pasó), la nueva posición en X e Y que tendrá el caballo, el nuevo identificador del estado generado y el id anterior será el id de la variable **estadoActual** (argumento de la función).

Antes de realizar cada movimiento se comprobará que dicho movimiento sea posible, esto se realizará por medio de las funciones **puedoArribaIzquierda**, **puedoArribaDerecha**, **puedoIzquierdaArriba**, **puedoDerechaArriba**, **puedoAbajoIzquierda**, **puedoAbajoDerecha**, **puedoIzquierdaAbajo** y **puedoAbajoDerecha**. Cada una de estas funciones recibe como argumento la variable **estadoActual** y revisa que la posición que ocupará el caballo en el estado que generará cada movimiento respectivamente, sea una posición valida, esto quiere decir, que la posición se encuentre dentro del tablero y con no sea una posición por la que ya ha pasado 2 veces (posición = 3). Si es que la nueva posición cumple con los requisitos, la función retornará un 1 indicando que se puede realizar el movimiento y un 0 indicando que no se puede.

Si el estado que se genera se encuentra en una posición valida, se procederá a crear este estado mediante la función de movimiento en cada caso, este estado será almacenado en la variable **nuevoEstado**, una vez creado este estado, se revisará que este estado no se encuentre en la lista de estado abiertos o en la lista de estado cerrados, esto se realiza mediante la función **noEsta**, esta siendo llamada primero para revisar dentro de los estados abiertos y luego para revisar dentro de los estados cerrados. Si es que el estado no se encuentra en la lista de estados abiertos ni en la de cerrados, se verifica que no sea un estado final, esto se hace contando la cantidad de peones restantes dentro del tablero mediante la función **peonesRestantes**, si es que el estado final (0 peones restantes) se procederá a imprimir las posiciones ocupadas por el caballo para llegar al estado final mediante la función **imprimirSolucion**, para luego salir de la función **Bee** retornando 0 que indica que la función termino correctamente. En caso de no ser un estado final, este **nuevoEstado** será agregado a la lista de estado abiertos y se seguirá realizando el resto de los movimientos en caso de que se pueda.

Si es que el programa no sale de la función **Bee** mediante un **return** dentro de la comprobación de estado final, este saldrá del ciclo while, ya que habrá revisado todos los estados generados por el algoritmo sin encontrar una solución, si esto pasa la función retornará un 1 indicando que la función termino, pero no puedo encontrar una solución. Esto puede pasar por el problema no tiene una solución o más probablemente, porque la forma en la que se hizo este programa no fue la correcta para encontrar todas las soluciones.