ESTRUCTURAS DE DATOS Y ALGORITMOS II
TERCERA ENTREGA DE EJERCICIOS

Se aceptan entregas por Aulas hasta el 9/10/2016 a las 23:50

Para la realización de este trabajo la cátedra provee un proyecto de Visual Studio con las funciones a implementar, algunos datos de prueba y el framework visto en clase.

IMPORTANTE:

1. La definición de cada función incluye siempre indicar sus pre y postcondiciones.
2. Las pruebas provistas por la cátedra constituyen meros ejemplos y no son para nada exhaustivas. Se espera que los alumnos desarrollen pruebas propias para su código.
3. Debe entregarse solamente una carpeta comprimida conteniendo el proyecto "Obligatorio", sin el Framework.

?1. Colas de Prioridad.

En esta tarea vamos a escribir un programa para resolver el famoso *Juego del 8*, también conocido como *Taken*. Desde chicos muchos se han enfrentado a este rompecabezas pero pocos conocen que es posible resolverlo de manera mecánica, usando un algoritmo de búsqueda conocido como A^*[1].

El problema. El juego del 8 es un rompecabezas popularizado por Noyes Palmer Chapman en la década del 1870. Se juega en una grilla de 3x3 con 8 bloques cuadrados que contienen los números del 1 al 8 y un cuadrado libre. El objetivo del juego es reordenar los bloques de forma que queden en orden. Está permitido deslizar los bloques horizontalmente o verticalmente dentro del espacio libre. A continuación se ilustra una secuencia de movimientos legales desde la posición inicial (izquierda) hasta la posición final (derecha).

```
1     3        1  2  3        1  2  3        1  2  3
4  2  5   =>   4     5   =>   4  5     =>   4  5  6
7  8  6        7  8  6        7  8  6        7  8
```

Algoritmo e implementación. A continuación vamos a describir una solución mecánica, i.e. un algoritmo, al rompecabezas conocido como el algoritmo de búsqueda A^*. Definimos un *estado* del juego como: la posición del tablero, el número de movimientos hechos hasta el momento para llegar a esa posición, y el

[1] Pronunciado como "A asterisco" o "A estrella".

estado inmediatamente anterior. Primero, insertar el estado inicial (tablero inicial, 0 movimientos y un estado previo nulo) en una cola de prioridad. Luego, borrar de la cola el estado con la mínima prioridad e insertar en la cola todos sus estados vecinos (es decir, todos a los que se pueden acceder en un sólo movimiento). Repetir este procedimiento hasta que el estado desencolado sea el estado final. No habrá escapado al lector atento que el éxito de este procedimiento depende totalmente de la elección de la prioridad de cada estado. Consideramos dos posibles asignaciones de prioridad:

(1) **Prioridad A**: La cantidad de bloques en una posición equivocada más el número de movimientos hechos hasta el momento. Intuitivamente, un estado con menos bloques en la posición equivocada está más cerca del estado final, y preferimos un estado al que se haya llegado usando la menor cantidad de movimientos.
(2) **Prioridad B**: La suma de distancias (suma de distancais verticales y horizontales) desde cada bloque hasta su posición final, más el número de movimientos hechos hasta el momento para llegar a ese estado.

Por ejemplo, las prioridades A y B para el siguiente estado inicial serían 5 y 10 respectivamente:

```
8  1  3        1  2  3
4     2        4  5  6
7  6  5        7  8
inicio         fin
```

```
1  2  3  4  5  6  7  8        1  2  3  4  5  6  7  8
-------------------------     -------------------------
1  1  0  0  1  1  0  1        1  2  0  0  2  2  0  3

Prioridad A = 5 + 0           Prioridad B = 10 + 0
```

Una observación clave es que para resolver el rompecabezas desde un estado en la cola de prioridad se necesitan al menos tantos movimientos como su prioridad (incluyendo los ya realizados), tanto si se está usando la prioridad A como la B[2].

Como consecuencia de lo anterior, al momento de desencolar un estado, no sólo hemos encontrado una secuencia de movimientos desde el estado inicial hasta éste, sino la secuencia con la menor cantidad de movimientos. (Desafío para los amantes de la Matemática: ¡demostrarlo!).

[2]Para la prioridad A esto es cierto porque cada bloque que esté fuera de lugar deberá moverse al menos una vez para llegar a la posición final. Para la prioridad B es cierto porque cada bloque debe moverse al menos su distancia hasta la posición final. Notar que no contamos el espacio en blanco para computar las prioridades.

Optimización. Luego de implementar el algoritmo descripto anteriormente notarán un efecto un tanto molesto: estados correspondientes al mismo tablero se encolan múltiples veces. Para prevenir una exploración innecesaria de estados inútiles, cuando considere los vecinos de un estado, no encole aquellos cuya posición es igual al estado previo. Por ejemplo:

```
8 1 3       8 1 3       8 1 3
4   2       4 2         4   2
7 6 5       7 6 5       7 6 5

previo        actual      deshabilitar
```

Su tarea. Escriba un programa que lea un tablero inicial e imprima la secuencia de movimientos que resuelve el rompecabezas en la menor cantidad de movimientos posibles. También deberá imprimir el número total de movimientos. La entrada consistirá de la dimensión del tablero N, seguida del tablero inicial de NxN. El formato de entrada utiliza el número 0 para representar el cuadrado en blanco. Por ejemplo:

Entrada:

```
3
0 1 3
4 2 5
7 8 6
```

Salida esperada:

```
  1 3
4 2 5
7 8 6

1   3
4 2 5
7 8 6

1 2 3
4   5
7 8 6

1 2 3
4 5
7 8 6

1 2 3
4 5 6
7 8

Cantidad de movimientos requeridos = 4
```

Note que su programa deberá funcionar para tableros de dimensión arbitraria NxN (con $N \geq 2$).

Detectando tableros imposibles. No todos los tableros iniciales llevan al estado final. Modifique su programa para que detecte estas situaciones. Por ejemplo:

Entrada:

```
3
1  2  3
4  5  6
8  7  0
```

Salida esperada:

```
Sin solucion.
```

Sugerencia: note que las posibles posiciones del tablero pueden dividirse en dos clases de equivalencia: (i) aquellas desde las que es posible alcanzar la posición final y (i) aquellas desde las que es posible alcanzar la posición final sólo si las modificamos intercambiando cualquier par de bloques adyacentes (no vacíos). Por ejemplo, en el caso anterior, intercambiar 3 con el 6 (o cualquier par de bloques adyacentes) resultaría en un tablero con solución. Hay básicamente dos formas de aplicar esta sugerencia:

(a) Correr el algoritmo A^* en simultáneo con dos tableros, uno con la configuración inicial y uno con ésta modificada intercambiando dos pares adyacentes (no vacíos). Exactamente uno de ellos llegará a la posición final.
(b) Derivar una fórmula matemática que le indique si un tablero dado tiene o no solución.

Especificaciones de Tablero y Solucion. Organice su programa usando una cantidad apropiada de tipos de datos. Como mínimo, está obligado a implementar las métodos de las clases `Tablero` y `Sistema`. Está permitido agregar más funciones y clases (como por ejemplo `Estado`) pero no puede alterar la firma de ninguno de los métodos dados.

Entregables.

(1) Especifique un TAD cola de prioridad extendida.
(2) Implemente el TAD anterior utilizando un heap.
(3) Implemente las funciones de las clases `Tablero` y `Sistema`, haciendo uso de la cola de prioridad que especificó. Deberá implementar y probar tanto la prioridad A como la prioridad B.