Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación

Tarea 3

IIC2133 - Estructuras de datos y algoritmos

Primer semestre, 2016

Entrega: Lunes 6 de Junio. Esta tarea se puede hacer en parejas

Objetivos

- Resolver un problema de optimización
- Diseñar una función de hash ad-hoc a un problema
- Optimizar el uso en memoria de un programa

Problema

Se tiene una grilla rectangular de n x m bloques de colores como muestra la figura:

Se puede modificar el estado de esta grilla mediante las siguientes operaciones:

- Desplazamiento de la j-ésima fila hacia la izquierda.
- Desplazamiento de la j-ésima fila hacia la derecha.
- Desplazamiento de la i-ésima columna hacia la arriba.
- Desplazamiento de la i-ésima columna hacia la abajo.

De aquí en adelante se referirá a los desplazamientos como `shift dirección índice`, ya que la dirección indica si se trata de una fila o de una columna.[1] Lo anterior queda entonces como

- `shift L j`
- `shift R j`
- `shift U i`
- `shift D i`

`0 <= j < n , 0 <= i < m`

[1] Las direcciones en ingles serían R = Right, L = Left, U = Up, D = Down

Al desplazar una fila o una columna, el bloque que sale del tablero aparece por el otro lado, como se puede ver en la figura:

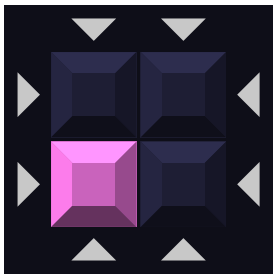
Estado inicial

`shift R 0`

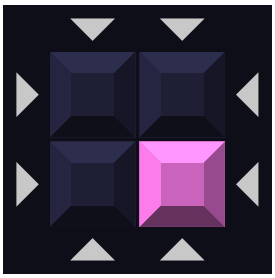
`shift D 0`

Tu objetivo es escribir un programa en C que encuentre la secuencia de pasos que lleva de un estado a otro del tablero, sujeto a la restricción de que solo se puede operar sobre determinadas filas o columnas, conocidas como "activas". Estas son específicas para cada instancia del problema. A continuación se muestran dos ejemplos:

Estado inicial 1

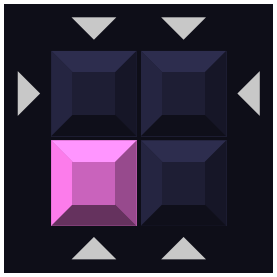


Estado deseado 1

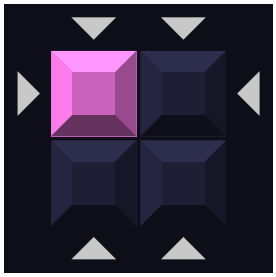


`shift R 1`
ó
`shift L 1`

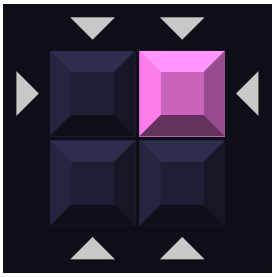
Estado inicial 2



(Fila 1 inactiva)

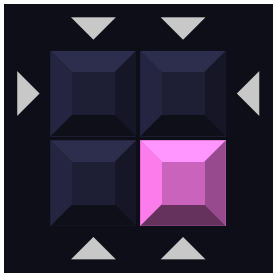


`shift U 0`
ó
`shift D 0`



`shift R 0`
ó
`shift L 0`

Estado deseado 2



`shift D 1`
ó
`shift U 1`

Como podrás notar, la secuencia de pasos que resuelve el problema no es única, y además varía considerablemente según qué filas y columnas permiten desplazamiento. Basta con que tu programa encuentre una de estas secuencias para ser correcto. Esta serie de pasos no debe ser óptima, pero sí debe estar bajo una cantidad de pasos determinada, específica para cada puzzle.

Input

Tu programa deberá recibir como primer parámetro la ruta a un archivo de prueba. Este archivo sigue la siguiente estructura:

Las primeras dos líneas indican las dimensiones de la grilla, `n` y `m`
```
HEIGHT 4
WIDTH 5
```
La siguiente línea indica qué columnas están activas. El primer elemento indica cuantas son.
```
4 0 1 3 4
```
La siguiente línea indica qué filas están activas. El primer elemento indica cuantas son.
```
2 0 3
```
Las siguientes `n` líneas tienen la información del estado inicial, cada celda separada por un espacio:
```
0 0 0 0 0
0 0 0 0 0
0 6 0 0 0
0 0 0 0 0
```
La siguiente línea especifíca el limite de la cantidad de pasos para este problema. Tu programa debe encontrar una solución de esta cantidad de pasos o menos.
```
LIMIT 5
```
Las siguientes `n` líneas tienen la información del estado final, cada celda separada por un espacio:
```
0 0 0 0 0
0 0 0 6 0
0 0 0 0 0
0 0 0 0 0
```

Output

Tu programa deberá imprimir una secuencia de pasos que lleve al estado deseado, siguiendo un formato similar al descrito anteriormente:

- Si el movimiento fue `shift R j` debes imprimir `R j`
- Si el movimiento fue `shift L j` debes imprimir `L j`
- Si el movimiento fue `shift U i` debes imprimir `U i`
- Si el movimiento fue `shift D i` debes imprimir `D i`

Con el ejemplo anterior, un posible output para tu programa sería

```
D 1
R 3
R 3
U 3
U 3
```

Recuerda imprimir los movimientos en el orden correcto, ya que no son conmutativos. Si la secuencia de pasos es correcta, pero más larga que el límite, se te descontará 0.1 por cada paso adicional, hasta un máximo de 5 pasos.

Solución

Como habrás podido ver, hay muchas formas de llegar a un mismo estado, por lo que tu algoritmo necesitará tener registro de los estados que ya ha visitado. Se recomienda seguir los siguientes pasos para resolver la tarea

1. Implementa un algoritmo de búsqueda. Usa una lista para llevar la cuenta de los estados conocidos.
2. Construye una tabla de hash para llevar la cuenta de los estados conocidos.
3. Diseña una función de hash ad-hoc al problema.
4. Minimiza el recorrido de la funcion de hash.

Análisis

Deberás entregar un informe donde analices tu solución al problema. En particular, se espera que hables de lo siguiente:

- Qué algoritmo de búsqueda usaste y por qué
- Qué tipo de tabla de hash usaste y por qué
- La función de hash que diseñaste. De ella deberás
  - Calcular su recorrido
  - Calcular su complejidad
- Como hiciste para reducir la memoria usada por tu programa

Además, un análisis de desempeño comparando cada una de las distintas formas de registrar estados conocidos.

Evaluación

El 50 % de tu nota proviene del desempeño de tu programa, mientras que el otro 50 % viene del informe.

La nota de código se divide en correctitud (60 %) y uso de memoria (40 %). 25 % del puntaje por uso de memoria será asignado según el nivel de memory leaks que tengas, el resto será relativo al desempeño del resto del curso. Dado que el stack es constante, se considerará solo el uso de heap.

Tu código será evaluado con 4 sets de test de dificultad creciente:

1. Easy: Se puede resolver usando una lista.
2. Normal: Se puede resolver usando una tabla de hash simple.
3. Hard: Necesita una función de hash ad-hoc.
4. Lunatic: Debes revisar bien el problema antes de empezar. Comprime tu programa.

De cada set se probarán 3 test, por 0.5 puntos c/u. Si tu programa no entrega una respuesta en 10 segundos será cortado, obteniendo 0 puntos en ese test.

Entrega

Deberás entregar en el repositorio de Github asignado a tu grupo una carpeta de nombre **Tarea03**. Y dentro de esta carpeta, una carpeta **Programa** y una carpeta **Informe**. Dentro de *Programa* deberás tener un *Makefile* y una carpeta **src** con todo el código de tu tarea. Ésta se compilará con el comando `make` dentro del mismo directorio, y tu programa deberá generarse en un archivo de nombre `solver`.

En la carpeta *Informe* deberás tener un archivo de nombre *Informe.pdf*.

Se recogerá el estado de la rama `master` de tu repositorio, 1 minuto pasadas las 24 horas del día de entrega. Recuerda dejar ahí la versión final de tu tarea.

Bonus

A continuación, formas de aumentar la nota obtenida en tu tarea

Lone Wolf (+40 % a la *Nota Final*)

Se te dará este bonus si haces la tarea de manera individual.

IMPOSIBRU (+15 % a la nota de *Código*)

Tu programa deberá ser capaz de darse cuenta cuando un problema no tiene solución, imprimiendo `IMPOSIBRU` en consola y terminando su ejecución. El límite de tiempo para esto es el mismo que para los otros tests.

Óptimo (+15 % a la nota de *Código*)

Implementa IDDFS para encontrar la secuencia óptima de pasos sin tener que usar memoria adicional.

Ortografía perfecta (+5 % a la nota de *Informe*)

La nota de tu informe aumentará en un 5 % si no tienes faltas de ortografía. Faltas gramaticales te harán perder este bonus.

Buen uso del espacio y del formato (+5 % a la nota de *Informe*)

La nota de tu informe aumentará en un 5 % si tu informe está bien presentado y usa el espacio y formato a favor de entregar la información. A juicio del corrector.