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

IIC2133 — Estructuras de Datos y Algoritmos
2020 - 1

Tarea 3

Fecha de entrega código: 12 de Julio

Objetivos

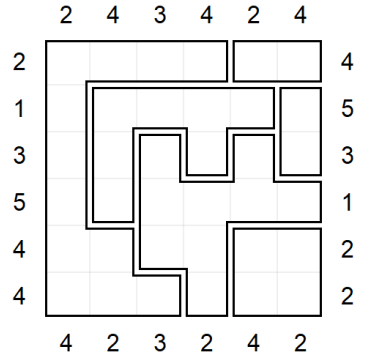
- Modelar un problema para poder resolverlo usando backtracking
- Identificar patrones en un problema de manera de mejorar el algoritmo usando podas y heurísticas
- Utilizar estructuras de datos de uso general para implementar dichas mejoras de manera eficiente

Introducción

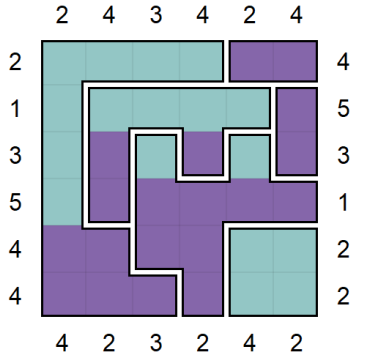
Tras meses a la deriva, el Pod 137 finalmente ha llegado a la Tierra. Desgraciadamente, la situación no podía ser más catastrófica. Los casos de *SpaceFlu* se han salido de control y los robots, inmunes a esta enfermedad, han aprovechado la oportunidad para esclavizar a la humanidad.

La única esperanza en esta guerra es el temible Robo-Bot. Sin embargo, semejante titán requiere como combustible una combinación muy específica de materia clara y oscura. Su manual de instrucciones indica cuánto combustible introducir, sin embargo, la forma de los estanques supondría un gran problema hasta para el mecánico de NP-NASCAR más experimentado.

Luego de que el inventor del Robo-Bot muriera en un duelo a muerte con cuchillos espaciales (en el que se cree pelearon con cuchillos espaciales), el proceso de llenado se volvió conocimiento arcano, y el gran Robo-Bot quedó para siempre desactivado. Tu misión es escribir un programa capaz de decodificar las complejas instrucciones, llenar el estanque y salvar la Tierra!



Estanques con instrucciones de cuanta materia incluir

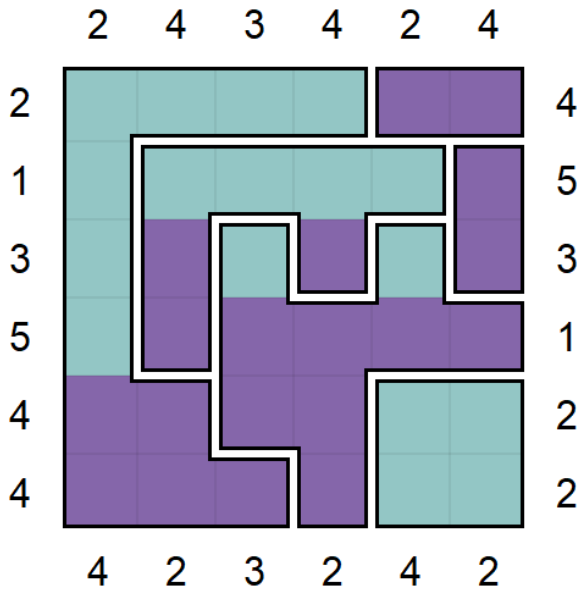


Estanques llenos con materia oscura y materia clara

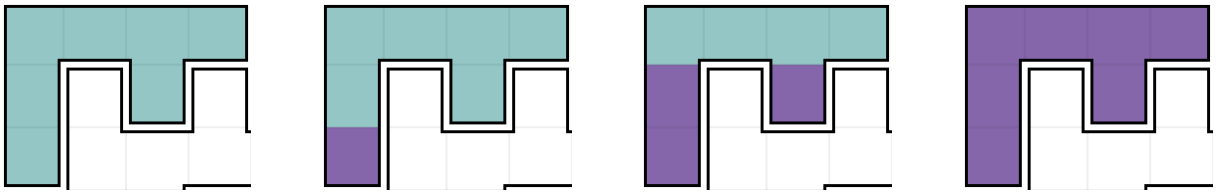
Problema

El problema consiste en una grilla rectangular, donde cada celda forma parte de un **estanque**.

En cada fila o columna debe haber una cantidad exacta de casillas con **materia oscura**, la cual se indica con el número a la izquierda de la fila o el número arriba de la columna. Al lado opuesto, los números indican la cantidad de casillas con **materia clara** que hay en esa fila o columna respectivamente.



Todas las celdas deben tener materia, ya sea clara o oscura. La materia oscura, siendo más pesada, llena el estanque de abajo hacia arriba, mientras que la materia clara lo llena de arriba hacia abajo. En cualquier caso al llenarse un nivel de materia este se debe llenar **en todo su ancho**. Por lo tanto, para un estanque de altura h, identificamos $h + 1$ formas de llenarlo según la altura a la que lo llenamos con materia oscura:



Altura 0 Altura 1 Altura 2 Altura 3

Deberás excribir un programa en **Python3.6+** que sea capaz de encontrar la manera correcta de llenar los estanques con materia clara y oscura de manera que se cumplan todas las restricciones sobre las filas y las columnas. Se espera que utilices un algoritmo de **Backtracking**, y hagas uso de estructuras de datos del lenguaje como **Hash Sets**, **Diccionarios** y **Colas de Prioridad**[1].

Este problema es una reformulación del puzzle Aquarium, el cual puedes encontrar en el siguiente link

[1] https://pypi.org/project/pqdict/

Ejecución

Tu programa deberá responder llamadas de la forma

```
python3.6+ src/solver.py input output
```

desde la carpeta raíz de tu repositorio.

A continuación se detalla el formato del input y output.

Input

Los puzzles están en archivos de texto plano, y siguen el siguiente formato:

```
alto del puzzle
ancho del puzzle
materia oscura en cada columna
materia oscura en cada fila
cantidad de estanques
id estanques fila 0
id estanques fila 1
id estanques fila 2
...
id estanques fila (alto puzzle - 1)
```

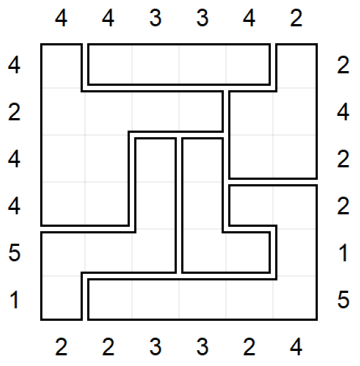
input.txt

Un ejemplo de un archivo de input es el siguiente:

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

(a) `test.txt`

$\Longrightarrow$



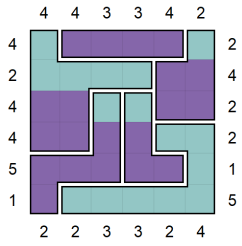
(b) `Puzzle correspondiente`

Los IDs de los estanques siempre irán de 0 a (cantidad de estanques - 1)

Output

Tu programa deberá entregar un archivo de texto plano que indique fila por fila las posiciones donde hay materia clara (`1`) o materia oscura (`0`), separados por espacios.

Un ejemplo de archivo de output es el siguiente:



(a) `Puzzle resuelto`

$\Longrightarrow$

```
1 0 0 0 0 1
1 1 1 1 0 0
0 0 1 1 0 0
0 0 0 0 1 1
0 0 0 0 0 1
0 1 1 1 1 1
```

(b) `output.txt`

Interfaz Gráfica

Para ayudar a debugear, tus ayudantes han preparado una interfaz gráfica en **PyQt5**[2] que permite visualizar el problema. A continuación se detallan las funciones estáticas de la clase `Watcher`:

```
def open(height: int, width: int, max_size: int=640)
```

Abre la ventana para un puzzle de `height x width`, con un tamaño máximo de `max_size` píxeles.

```
def set_cell_tank(row: int, col: int, ID: int)
```

Indica el ID del estanque al que pertenece la celda `(row, col)`

```
def set_cell_matter(row: int, col: int, dark: bool)
```

Indica que en la celda `(row, col)`, la materia es del tipo especificado.

```
def clear_cell(row: int, col: int)
```

Indica que en la celda `(row, col)` no se sabe el tipo de materia que hay.

```
def set_col_number(top: bool, col: int, number: int, color: Tuple[int,int,int] = (0,0,0))
```

Asocia el número `number` a la columna `col`. `top` indica si ha de dibujarse arriba o abajo de esta. Opcionalmente le puedes dar una tupla (R,G,B) si quieres que el número se pinte de algún color específico.

```
def set_row_number(left: bool, row: int, number: int, color: Tuple[int,int,int] = (0,0,0))
```

Asocia el número `number` a la fila `row`. `left` indica si ha de dibujarse a la izquierda o a la derecha de esta. Opcionalmente le puedes dar una tupla (R,G,B) si quieres que el número se pinte de algún color específico.

```
def update()
```

Redibuja el contenido de la ventana para reflejar los cambios hechos con las funciones anteriores.

```
def close()
```

Cierra la ventana.

[2] https://pypi.org/project/PyQt5/

Evaluación

La nota de tu tarea se descompone como se detalla a continuación:

- 50% a que tu programa pueda resolver los tests de dificultad *Easy*.
- 25% a que tu programa pueda resolver los tests de dificultad *Normal*.
- 25% a que tu programa pueda resolver los tests de dificultad *Hard*.

Si el output que entregas no cumple las restricciones del problema, tendrás automáticamente 0 puntos en ese test.

Si tu algoritmo demora más de 10 segundos en un test, será cortado y tendrás 0 puntos en ese test. Tu programa se probará con diversos inputs de tamaños crecientes.

Entrega

Código: GIT - Repositorio asignado. Se entrega a más tardar el día de entrega a las 23:59, hora de Chile continental.

BONUS

Moon Worship (+10 décimas a la nota de tu tarea): La nota de tu tarea aumentará hasta un máximo de 10 décimas según que tantos tests de dificultad *Lunatic* pueda resolver tu programa. Estos tests serán subidos más adelante. Además de podas y/o heurísticas, se recomienda utilizar Cython para hacer más eficiente tu código.

Política de recorrección

Una vez entregadas las notas, todo alumno puede solicitar una recorrección. Si desean solicitar recorrección, deben seguir los pasos que se detallan a continuación:

1. Deben abrir un issue en el repositorio de su tarea solicitando formalmente recorregir su tarea y explicando por qué. Si hacen cambios en su código, deben indicar qué cambiaron y qué commit quieren que se revise. Si estos cambios no cambian la lógica de su algoritmo, no llevan asociado descuento. En caso contrario, hay un descuento aplicado en proporción a la cantidad de cambios que hagan a su algoritmo.
2. Deben enviar el link de dicha issue a caespinoza5@uc.cl con el encabezado [IIC2133] - Recorregir T3 - [Su nombre]. Ejemplo: [IIC2133] - Recorregir T3 - Juan Pérez.