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

Tarea 2

IIC2133 - Estructuras de Datos y Algoritmos

Primer semestre, 2018

Entrega código: 17 de Mayo
Entrega informe: 22 de Mayo

Objetivos

- Modelar un problema de asignación para resolverlo usando Backtracking
- Analizar el impacto en el rendimiento de Backtracking al usar podas y/o heurísticas

Introducción

Distintos grupos rebeldes han ido causando estragos a lo largo de la ciudad, pero nadie ha logrado identificar quienes son realmente, ni si operan en conjunto. Como investigador privado has decidido tomar cartas en el asunto y averiguar quién es quién haciendo una pregunta que no incrimina a nadie: ¿Sabe cuantos de sus vecinos comparten su opinión?

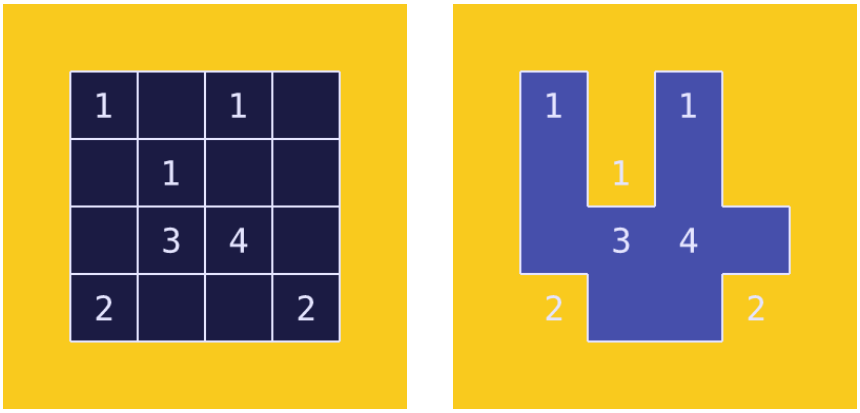
Reportes de tus contactos además te indican lo siguiente:

1. Los rebeldes forman un solo grupo de vecinos.
2. Los leales al imperio forman un solo grupo de vecinos, y rodean a los rebeldes.

Ahora sólo queda ponerse a probar como encajan todas estas pistas, y desenmascarar (o unirte) a la rebelión.

Problema: Rebeldes vs Imperio

El problema trata de una grilla de $h \times w$, donde cada celda representa a un ciudadano. El objetivo es asignar a cada celda un bando, cuidando de seguir las pistas. Algunas celdas indican cuantos de sus vecinos comparten su opinión.



(a) El problema a solucionar (b) Una posible solución

Figura 1: La representación del problema, con los rebeldes en azul y los leales al imperio en amarillo.

Asignación & Restricciones

Este se trata de un problema de satisfacción de restricciones, donde las variables son cada una de las celdas, y el dominio de todas ellas es simplemente si son o no rebeldes. Las restricciones son las siguientes:

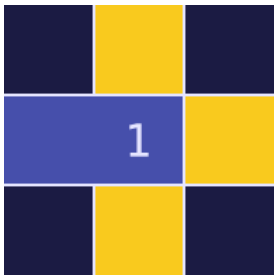
Rodeados

Los rebeldes están completamente rodeados por leales al imperio. Esto se traduce simplemente en que todas las celdas de la periferia deben ser leales al imperio, como puede verse en la Figura 1.a, donde se asignan antes de comenzar.

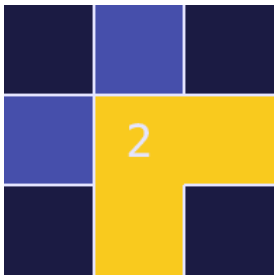
Vecinos

Definimos los **vecinos** de una celda como las celdas adyacentes a ellas.

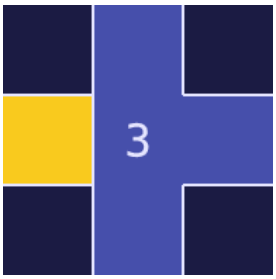
La restricción: Algunas celdas indican exactamente cuantos de sus vecinos piensan lo mismo que ellas, es decir, deben tener asignado el mismo valor que ellas. Llamamos a este número el **grado** de una celda. Las celdas que no tienen grado son libres de esta restricción.



Grado 1



Grado 2



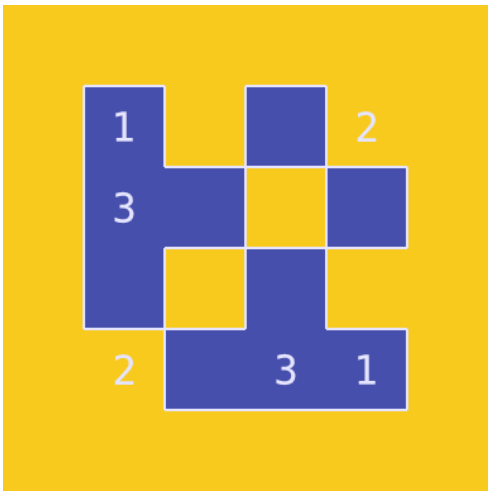
Grado 3

Grado 4

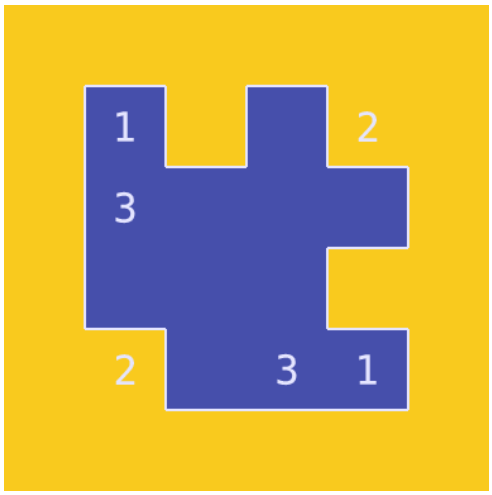
Grupos de Vecinos

Para una celda de cierta opinión, definimos su grupo de vecinos como el grupo de todas las celdas que son **alcanzables** desde esta celda pasando de vecino en vecino sólo por celdas de su misma opinión.

La restricción: El problema resuelto debe constar de sólo dos grupos, uno de rebeldes y otro de leales al imperio.



Una grilla con 7 grupos de vecinos



Una grilla con 2 grupos de vecinos

La solución de un problema sin y con esta restricción

Deberás escribir un programa en C que dada la descripción del problema encuentre la asignación tal que todas las restricciones se cumplen. Se espera que utilices **Backtracking** para resolverlo.

Podas de Backtracking

A simple vista surgen dos podas para este problema. Como mínimo deberás implementar ambas en tu solución.

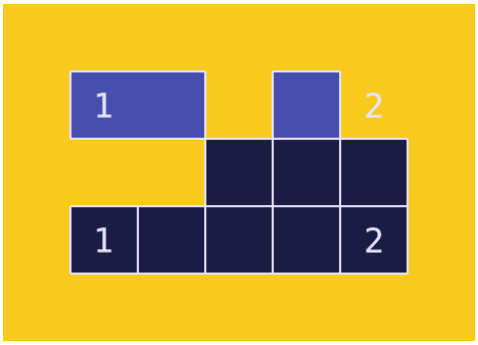
Poda de Vecinos

En ocasiones, ciertas asignaciones nos aseguran que no es posible satisfacer el grado de una celda. No es necesario esperar a que estén asignados la celda y todos sus vecinos para poder desartar esa configuración.

No es posible que los 4 vecinos sean iguales

Poda de Grupos

En ocasiones, ciertas asignaciones nos aseguran de que no es posible formar solo un gran grupo de rebeldes (o de leales al imperio). No es necesario esperar a que estén todas las celdas asignadas para descartar esa configuración.



No es posible unir esos dos grupos de rebeldes si uno esta encerrado

Informe y análisis

Debes hacer un informe que contenga un análisis teórico y empírico de tu solución. En particular, se espera lo siguiente:

- Comparación del desempeño al agregar las podas anteriormente mencionadas a la solución. Utiliza tanto el tiempo de ejecución como la cantidad de devoluciones como métrica de eficiencia. [1]
- Proponer una poda adicional, y justificar su utilidad.
- Proponer como propagar asignaciones, y justificar su utilidad.
- Proponer una heurística, y justificar su utilidad.

Para los últimos 3 puntos, no se espera que implementes eso en tu código, solo que hagas un análisis teórico.

[1] Te recomendamos ser ordenado en tu código o guardar versiones para que comparar sea una tarea simple

Input

Tu programa recibe como único parámetro la ruta del problema a resolver, en formato `.txt`.

La primera línea de estos archivos son dos números, h w correspondientes a las dimensiones de la matriz. Las siguientes h líneas consisten de w números separados por un espacio, los cuales corresponden al grado de cada celda. Las celdas sin grado tendrán un 0.

Por ejemplo, el archivo `.txt` para la configuración de la Fig. 1(a) contendría:

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

Interfaz

La interfaz funciona como una librería externa a tu programa la cual no debes modificar: esta es el módulo `Watcher`, que posee las siguientes funciones:

- `watcher_open(int height, int width)`: Abre una ventana para mostrar una matriz de las dimensiones dadas
- `watcher_set_cell_degree(int row, int col, int degree)`: Escribe un número entre 1 y 4 inclusive dentro de esa celda. Un 0 borra el número escrito.
- `watcher_set_cell_status (int row, int col, bool is_rebel)`: Indica si una celda es o no rebelde
- `watcher_clear_cell (int row, int col)`: Deja la celda en blanco
- `watcher_snapshot(char* filename)`: Imprime la ventana al 100 % de tamaño en formato PDF
- `watcher_close(char* filename)`: Cierra y libera los recursos de la ventana

Output

El output de tu programa serán los valores que estén en la ventana cuando se llame a `watcher_close`.

Evaluación

La nota de tu tarea estará descompuesta en dos partes:

- 70 % corresponde a que tu código pase los tests dentro del tiempo máximo.
- 30 % corresponde a la nota de tu informe.

Se probará tu programa con distintos test de dificultad creciente. Específicamente, dentro de la carpeta `Programa/tests` podrás encontrar 3 categorías:

- `Easy`: Basta con tener la solución básica, sin ninguna mejora.
- `Normal`: Es necesaria la poda de vecinos para poder resolverlos de manera eficiente.
- `Hard`: Son necesarias la poda de vecinos y la de grupos para poder resolverlos en un tiempo razonable.

Tu programa deberá ser capaz de obtener el output los tests dentro de 10 segundos. Pasado ese tiempo el programa será terminado y se asignarán 0 puntos en ese test.

Entrega

Deberás entregar tu tarea en el repositorio que se te será asignado; asegúrate de seguir la estructura inicial de éste.

Se espera que tu código compile con `make` dentro de la carpeta **Programa** y genere un ejecutable de nombre `solver` en esa misma carpeta. **No modifiques código fuera de la carpeta *src/solver*.**

Se espera que dentro de la carpeta **Informe** entregues tu informe en formato *PDF*, con el nombre *Informe.pdf*

Estar reglas ayudan a recolectar y corregir las tareas de forma automática, por lo que su incumplimiento implica un descuento en tu nota.

Se recogerá el estado de la rama `master` de tu repositorio, 1 minuto pasadas las 23:59 horas del día de entrega (tanto para el código como para el informe). Recuerda dejar ahí la versión final de tu tarea. No se permitirá entregar tareas atrasadas.

Bonus

Gotta go fast (+30 décimas a tu nota de código)

Se te asignarán 30 décimas a tu nota de código si este es capaz de resolver los tests de categoría `Lunatic`[2]. Estos son tests superdifíciles, que requerirán que incluyas aún más mejoras a tu solución. Utiliza podas, propagación y heurísticas de manera eficiente para poder resolverlos tan rápido como puedas. El bonus se te asignará según cuantos de estos tests consigas resolver.

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, a criterio del corrector, está bien presentado y usa el espacio y formato a favor de entregar la información.

Manejo de memoria perfecto (+5 % a la nota de *Código*)

Se aplicará este bonus si **valgrind** reporta en tu código 0 leaks y 0 errores de memoria, considerando que tu programa haga lo que tiene que hacer.

[2] Serán subidos en los próximos días