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

Tarea 1

IIC2133 - Estructuras de Datos y Algoritmos

Primer semestre, 2017

Entrega: Jueves 13 de Abril

Objetivos

- Modelar un problema de planificación con backtracking y llevarlo a la práctica
- Analizar el comportamiento de un problema exponencial
- Analizar el uso de podas y heurísiticas en el algoritmo de backtracking

Introducción

Bien se sabe que en la época colonial el gran duque Fernando I de Médici comenzó la iniciativa de formar colonias italianas en América. Es por esto que en 1608 envió una expedición hacia el norte de Brasil, bajo el mando del capitán inglés Robert Thornton. La razón principal era desarrollar el comercio de madera preciosa desde el Amazonas hacia la italia del Renacimiento, creando una base colonial entre las posesiones españolas y portuguesas en el norte atlántico de Sudamérica.

Lo que no muchos saben es que dentro de esta comitiva se encontraba el innovador arquitecto italiano **O**liverio **C**asimiro **D**ovelinni quien tenía la importantísima misión de llevar los planos del nuevo asentamiento italiano a América. Lamentablemente durante el trayecto a través del Atlántico, la flota italiana se vió envuelta en una tormenta y la nave en la que se encontraba Oliverio se hundió. Afortunadamente lograron rescatar a Oliverio, sin embargo, los planos se estropearon en el accidente.

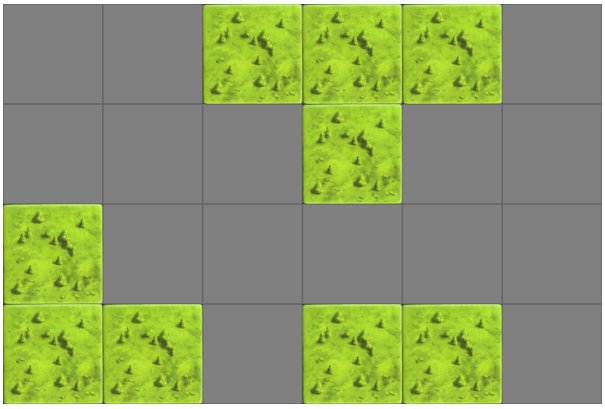
¡Pero no todo está perdido! Oliverio, debido a su particular forma de ser, se acordaba perfectamente del tamaño y forma del terreno que iban a utilizar para la base colonial y, además, recordaba todo lo que se debía construir. Pero, desgraciadamente, no lograba recordar en qué lugares del terreno se debían ubicar las distintas construcciones. Esta es tu labor.

Problema

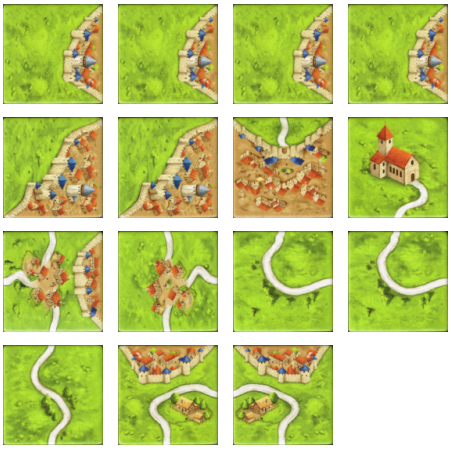
Para ayudar con la organización, el arquitecto se dió el trabajo de dividir el plano en una grilla rectangular, dentro de la cual estarían emplazadas las distintas construcciones del complejo colonial. Estas podían ocupar más de una celda dentro de la cuadrícula, y estaban divididas en 4 tipos:

1. Pueblos
2. Fortalezas
3. Carreteras
4. Iglesias

En definitiva, cada celda puede contener fragmentos de más de alguno de los anteriores. Tu deber es reconstruir el mapa a partir de los fragmentos de celdas proporcionadas por Oliverio. Además, Oliverio recuerda que en el plano hay ciertas celdas que no contenían ningún tipo de construcción (i.e. son sólo pasto).



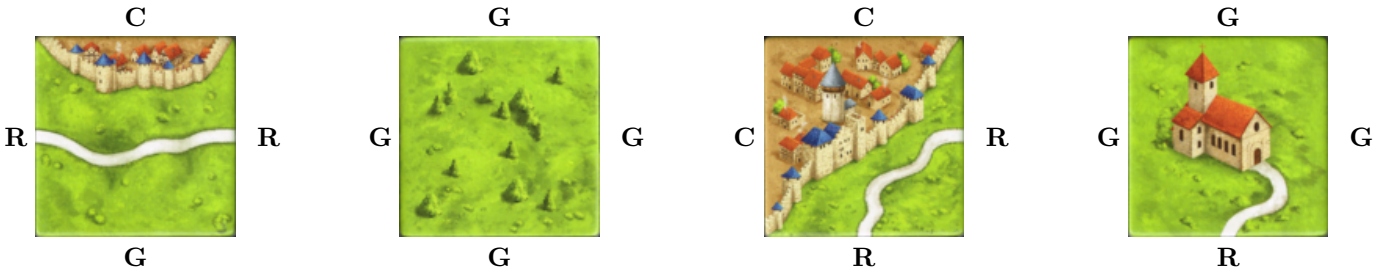
Los sectores que no se construyen son pasto, y donde no se sabe que hay está en gris.



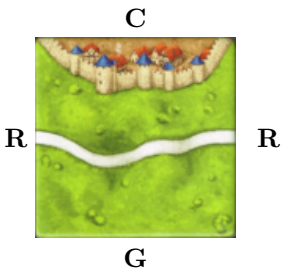
Los fragmentos del mapa que recuerda Oliverio

Posicionando los fragmentos correctamente se puede reconstruir el mapa

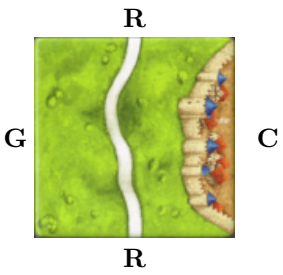
Oliverio le hace notar que solo importa asegurar que tanto los caminos como las fortalezas queden cerradas, ya que las iglesias y los pueblos no ocupan más de una celda. No solo eso, sino que además el problema se reduce a revisar únicamente las fronteras de cada celda y asegurarse que estas coincidan entre celdas colindantes. Siguiendo esta idea, cada celda se puede representar unívocamente por los valores de sus orillas. Estos pueden ser pasto (**G**), carretera (**R**) o fortaleza (**C**), como se ilustra en la siguiente figura:



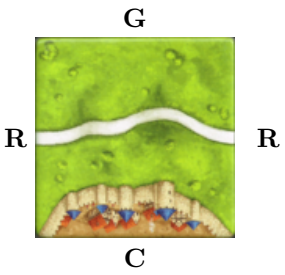
Nótese que las piezas pueden ser **rotadas** para que calcen correctamente en el mapa, como puede verse abajo:



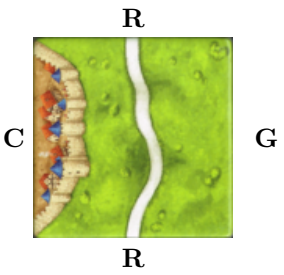
Pieza original x



x rotada **1** vez



x rotada **2** veces

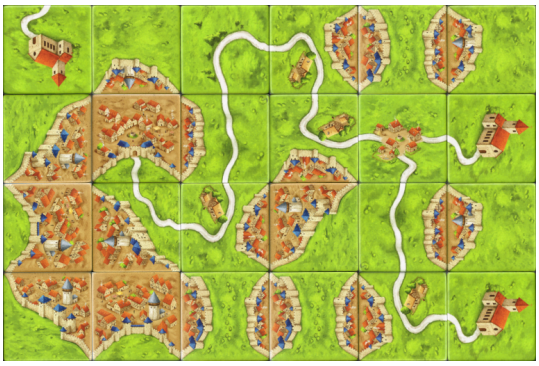


x rotada **3** veces

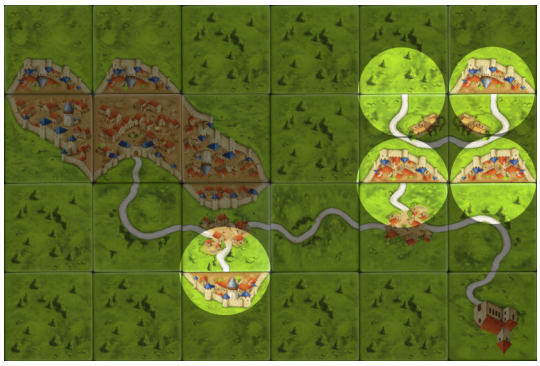
A modo de resumen, para que el mapa haya sido reconstruido correctamente debe cumplirse lo siguiente:

1. Los valores colindantes entre celdas adyacentes deben ser idénticos, ya sean celdas vacías o edificadas.
2. Todas los fragmentos del problema deben estar posicionadas en el mapa.
3. Si una celda no tiene vecino en una dirección, en esa dirección se considera que hay pasto.

A continuación ejemplos de mapas incorrectos:



No es solución, ya que de la iglesia de la celda (0,0) sale un camino que no llega a ningún lado. Por la regla **3**, sabemos que hacia afuera hay pasto, por lo que ese camino no es válido.



No es solución, ya que la regla **2** se rompe en las fronteras marcadas.

Deberás escribir un programa en C que dado los fragmentos del mapa y las celdas no habitadas encuentre una disposición válida de los fragmentos para reconstruir el mapa. Se espera que utilices backtracking para hacer esto.

Análisis

Deberás entregar un informe donde analices el problema. En particular, debes responder lo siguiente:

- (1pto) ¿Por qué este es un problema que debe ser resuelto con backtracking o algo similar?
- (2.5pts) Propón una poda para este problema. ¿Cual es el costo en tiempo de utilizarla? ¿Se puede usar alguna estructura de datos para mejorar su rendimiento? ¿A cuánto se reduce ese costo? Justifica.
- (2.5pts) Propón una heurística para este problema. ¿Cual es el costo en tiempo de utilizarla? ¿Se puede usar alguna estructura de datos para mejorar su rendimiento? ¿A cuánto se reduce ese costo? Justifica.

Puedes leer a que nos referimos con podas y heurísticas en la guía en el sitio web del curso: https://github.com/IIC2133-PUC/2017-1/blob/master/Guias/Backtracking.pdf

Input

Tu programa deberá recibir los siguientes parámetros:

1. La ruta del archivo que contiene los datos recordados por Oliverio
2. La ruta del archivo donde deberás guardar tu respuesta

De la siguiente manera:

```
./solve input.txt solucion.txt
```

El archivo de input sigue el siguiente formato:

Las primera línea contiene la altura **height** en celdas del mapa. La segunda línea contiene el ancho **width** del mapa en celdas. La siguiente línea contiene la cantidad **empty** de celdas inhabitables en el mapa, y las siguientes **empty** líneas contienen un par **row column** que indican la fila y la columna de las distintas celdas sin construcciones. Para evitar redundancia, los fragmentos se especifícan por tipos: la siguiente línea contiene los **k** tipos de fragmentos que hay, seguido de **k** líneas que especifican los fragmentos de la siguiente manera: **up right down left times**, que especifican cual es el valor colindante en cada dirección del fragmento, y la cantidad de veces que aparece ese fragmento en el problema. Para simplificar, los valores de las fronteras serán expresadas como números: $\mathbf{G = 1}$, $\mathbf{R = 2}$, $\mathbf{C = 3}$. Considera que cada fragmento tiene asociado como identificador (id) la posición en la que aparece dentro de esta lista. Un archivo de ejemplo sería:

```
3
5
12
0 0
0 1
0 2
0 3
0 4
1 0
1 4
2 0
2 1
2 2
2 3
2 4
2
1 1 2 1 2
2 1 2 1 1
```

Mapa de 3x5, con 12 celdas deshabitadas. El fragmento de identificador 0 : {`1 1 2 1`} aparece 2 veces en el problema, mientras que el fragmento de identificador 1 : {`2 1 2 1`} aparece 1 sola vez.

Output

En el archivo de output deberás escribir **(width x height) - empty** líneas, y en cada una especificar una posición, el id del fragmento que va en esa posición, y que tan rotado está. Esto debe seguir el siguiente formato:
row column id rotation. Un ejemplo de output para el problema anterior:

```
1 1 0 3
1 2 1 1
1 3 0 1
```

Evaluación

La nota de tu tarea está descompuesta en distintas partes:

- 70 % corresponde a que las respuestas generadas sean correctas. Se evaluarán distintos casos de prueba.
- 30 % corresponde a la nota de tu informe.

Los tests tendrán un tiempo límite de 10 segundos c/u. Si tu programa no ha terminado en ese tiempo será cortado y tendrás 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 `restore` en esa misma carpeta. No modifiques código fuera de la carpeta *src/restore*.

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

Cada regla tiene asociado un puntaje, asegúrate de cumplirlas para no perder puntos.

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

Bonus

Como regla general, un bonus a tu nota se aplica solo si la nota correspondiente es $\geq$ 3,95. A continuación, las distintas formas de aumentar tu nota.

The real deal (+40 % a la nota de *Código*)

Para optar a este bonus, tu solución debe ser lo suficientemente eficiente como para pasar un set de tests de alta dificultad. Se recomienda encarecidamente implementar podas y heurísticas con sus respectivas estructuras de datos para asegurar la alta velocidad que requiere 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, 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.

Anexo: Interfaz gráfica

Se ha preparado para esta tarea una interfaz gráfica en base a GTK+3. Puedes llamarla desde tu programa usando el módulo **watcher.h**. En este archivo podrás leer las funciones que provee y como usarlas.

Para instalar GTK+3 en tu computador sigue la guía en el sitio web del curso: https://github.com/IIC2133-PUC/2017-1/wiki/4.-Setup-GTK-3