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

IIC2133 — Estructuras de Datos y Algoritmos
2019- 1

Tarea 3

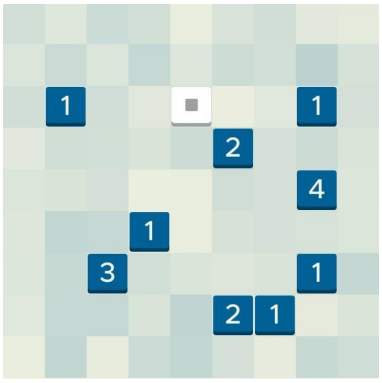
Fecha de entrega código: 9 de junio
Fecha de entrega informe: 11 de junio

Objetivos

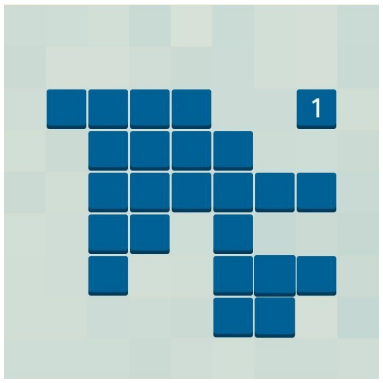
- Modelar un problema de combinatoria de manera de poder resolverlo usando Backtracking
- Analizar optimizaciones al modelo en forma de podas o heurísticas.

Introducción

ZHED es un juego de puzzle por *Ground Control Studios* en donde tienes que armar puentes de manera de llegar al cuadrado objetivo. El **orden** y **dirección** de como extiendes estos puentes son cruciales para poder llegar al final del nivel. Para esta tarea deberás resolver una versión simplificada (pero no tanto) del juego **ZHED**, la cual llamaremos **ZHEDD**. Puedes ver el trailer del juego original para hacerte una idea aquí.



Ejemplo de un problema de ZHED



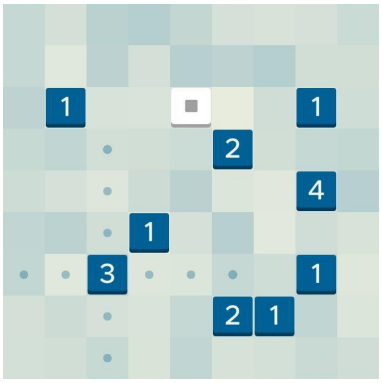
Ejemplo resuelto

Problema

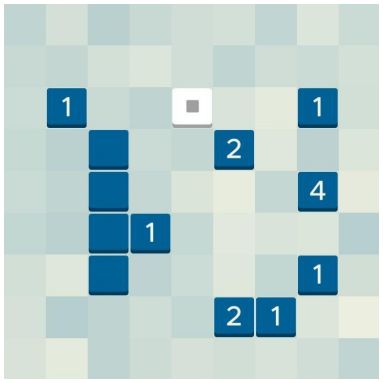
El mundo que constituye un nivel está compuesto por una matriz de $H \times W$, donde inicialmente cada celda puede tener el inicio de un puente de largo $L \in \{1...9\}$ o estar vacía. Los puentes no están extendidos inicialmente, y una vez que se extienden, no pueden volver a contraerse. Para completar el nivel, tienes que extender los puentes en la dirección y orden correctos de manera que uno de ellos se superponga al cuadrado objetivo. La solución del problema se puede representar como la secuencia de pasos para cubrir el objetivo, entiéndase cada paso como la tupla (puente, dirección). La sintaxis especifica se explicará mas adelante.

Extensión de puentes

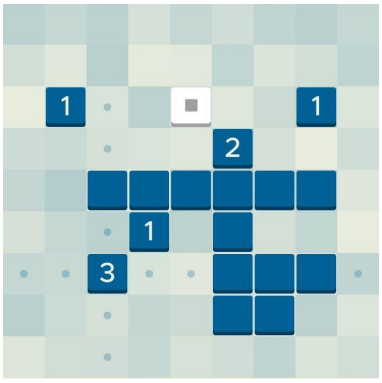
En un paso, un puente de largo L puede ser extendido en alguna las 4 direcciones: `UP, LEFT, DOWN, RIGHT`. Al ser extendido, las L casillas vacías en dicha dirección se transforman en una casilla de tipo puente, además de la casilla inicial del puente. Por lo tanto, si una casilla en dicha dirección ya era de tipo puente, y la extensión de este puente se superpone con dicha casilla y la extensión del puente aumenta en uno. A continuación, un par de ejemplos para ilustrar lo anterior:



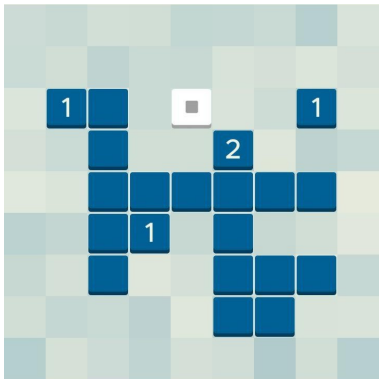
Estado Inicial



Mover hacia arriba (`UP`), sin puentes en el camino



Estado Inicial



Mover hacia arriba (`UP`), pasando por sobre otro puente

Algoritmo

Este es un problema de búsqueda en el espacio de estados, sin embargo el largo de la solución no es un problema, solo se pide resolver el puzzle. Debido a que cada nuevo puente se puede armar en cuatro direcciones diferentes, implica que el factor de ramificación crece de manera exponencial con la cantidad puentes disponibles, por lo que **BFS** no es una solución viable debido a que la complejidad de memoria es lineal con respecto al espacio de estado. Una alternativa que utiliza menos memoria es **DFS**, la cual puede ser implementada mediante **Backtracking**.

En esta tarea deberás escribir un programa en C que reciba como input una representación del problema con puentes no extendidos, y entregue una secuencia de pasos tal que conecte algún puente con el cuadrado objetivo. Se espera que utilices **Backtracking** para resolver este problema, pero eres libre de utilizar otra técnica si lo estimas conveniente (bajo tu propio riesgo).

Interfaz Gráfica

Se ha preparado una interfaz gráfica para que puedas visualizar la ejecución de tu algoritmo. Esta interfaz sirve como ayuda para visualizar el problema pero no tiene influencia en la corrección del programa. Los detalles de los métodos disponibles se dan a continuación:

- `watcher_open(int height, int width)`: Abre una ventana con la matriz de dimensiones height X width, lleno de celdas vacías.
- `watcher_set_cell_goal(int row, int col)`: Dibuja el cuadrado objetivo en la posición indicada.
- `watcher_set_cell_degree(int row, int col, int degree)`: Escribe un número dentro de la celda en la posición indicada.
- `watcher_set_cell_status(int row, int col, bool is_block)`: Cambia el formato de la celda. Si se define is_block como true esa celda pasará a ser de tipo "puente", si se define como false se va a mostrar la celda normal.
- `watcher_refresh()`: Muestra la versión mas actualizada de la ventana (ejecútenla después de cada paso).
- `watcher_snapshot(char* filename)`: Imprime la ventana en formato PNG. Usalo para analizar casos específicos en el informe.
- `watcher_close()`: Cierra la ventana y libera sus recursos.

Ejecución

Tu programa debe ejecutarse de la siguiente forma:

```
./solver <test.txt> <out.txt>
```

Donde `<test.txt>` corresponde al archivo con el problema a resolver, y `<out.txt>` corresponde al archivo donde deberás escribir la solución.

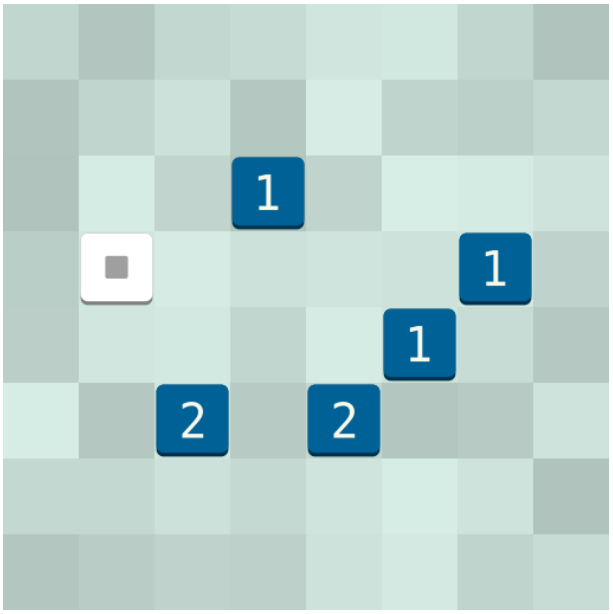
Input

La primera línea del archivo de test contendrá dos números H W, que corresponden al alto y ancho del mundo respectivamente.
La siguiente linea contiene un número N que indica la cantidad de inicios de puentes en el problema.
Las siguientes N líneas contienen la información de cada puente en la forma de 3 números P_{row} P_{col} L que corresponden a la fila, columna y largo del posible puente respectivamente.
La siguiente y última línea contiene 2 números G_{row} G_{col} que indican la fila y columna donde se encuentra la meta.

Este sería un ejemplo de un archivo de input:

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

El cual se vería en el watcher como:



Representación gráfica del ejemplo.

Output

El archivo `[output.txt]` deberá tener un número que indica el número de pasos ejecutados, y luego por cada línea siguiente tendrá tres valores: r c d, que indica que la torre en la fila r, columna c, se extendió en la dirección d, en donde las direcciones son `UP = 0, LEFT = 1, RIGHT = 2, DOWN = 3`.

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

Debe cumplirse que los pasos ejecutados lleven a la solución desde el estado inicial, esto es, que una casilla puente superponga a la casilla de meta.

Evaluación

La nota máxima en esta tarea sera un 10, la cual corresponderá a:

1) 3 puntos por que tu código sea capaz de resolver los niveles del 1 al 10.
2) 3 puntos por que tu código sea capaz de resolver los niveles del 11 al 30
3) 3 puntos por informe, pero sólo si tienes el puntaje de la parte 1)
4) 1 punto base de regalo de los ayudantes.

Tu programa será probado con diferentes niveles de dificultad creciente. Para cada uno de ellos, tu programa deberá ser capaz de entregar el output correcto dentro de 10 segundos. Pasado ese tiempo tu programa será terminado y se te asignarán 0 puntos en ese test.

Informe y Análisis

En este informe deberán hacer un análisis profundo sobre dos podas y dos heurísticas a su elección: como afectan el desempeño del algoritmo, y por qué.

Este análisis deberá estar respaldado por datos y gráficos donde se vea claramente los casos en los que dichas podas y heurísticas son un aporte (o no). Es decir, debes haberlas implementado.

Las métricas que más interesan son el **tiempo** que toma Backtracking en resolver el problema y la cantidad de veces que este hace **UNDO**. Considera también el tamaño del problema N (cantidad de posibles puentes) para cada caso.

Más específicamente, tanto para las podas y heurísticas, deberás:

- Explicar detalladamente en que consisten
- Justificar desde un punto de vista teórico el por qué deberían ser una ayuda.
- Explicar como hiciste para implementarlas de manera eficiente
- Analizar los datos ayudándote de gráficos, comparando las métricas utilizando sólo Backtracking, utilizando podas, utilizando heurísticas y utilizando mezclas de ambas.

- Justificar anomalías dentro de los datos analizando casos representativos.

Nótese que esta opción se provee ya que es posible invertir tiempo implementando podas y heurísticas que no ayudan mucho, por lo que tu programa no tendría puntaje por pasar los test más difíciles. Por esto es que **tu informe sólo será evaluado si tu programa al menos es capaz de resolver los test básicos.**

Entrega

- **Código:** GIT - Repositorio asignado (asegúrate de seguir la estructura inicial de éste).
  - En la raíz del repositorio debe encontrarse el código.
  - Se recogerá el estado en la rama *master*.
  - Se espera que el código compile con `make` y genere un ejecutable de nombre `solver` en esa misma carpeta.
- **Hora Límite:** 1 minuto pasadas las 23:59 del día de la entrega.
- No se permitirán entregas atrasadas.

Bonus: Manejo de memoria perfecto (+5% a la nota del Código)

La nota de tu código aumentará en un 5% si `valgrind` reporta 0 leaks y 0 errores de memoria en tu código. Todo esto considerando que tu programa haga lo que **tiene** que hacer. El bonus solo aplicará en caso que tu nota de código sea sobre 4.

Anexo

Dentro de la carpeta `src/solver` se incluye un pequeño módulo llamado `util`, que contiene funciones que te pueden ser útiles para el manejo de coordenadas y direcciones dentro del tablero.