Trabajo Práctico Especial
Estructuras de Datos y Algoritmos
Primer Cuatrimestre 2015

Objetivo

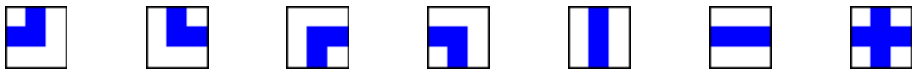
El objetivo del trabajo práctico es desarrollar una aplicación que resuelva niveles de una variante del juego **Pipe Dream**, obteniendo soluciones exactas y aproximadas.

Requerimientos

Se cuenta con una grilla en donde se debe ubicar un conjunto de piezas que representan cañerías de distintas formas. El objetivo es construir la cañería más larga posible que permita transportar el agua desde una celda de origen hacia afuera del tablero. Se tiene una única celda de origen, y además hay "paredes" que son celdas que no pueden ser utilizadas.

La aplicación deberá leer de un archivo la estructura del tablero, junto con el conjunto de piezas disponibles, y resolver el problema de acuerdo al algoritmo elegido por el usuario. La solución se debe mostrar en pantalla una vez encontrada. Es decir, se debe graficar el tablero junto con el camino construido.

Los tipos de piezas disponibles son:



Para todos los casos en los que hay dos extremos, el agua puede ingresar por cualquiera de ellos y salir por el otro. Para la pieza de tipo 7, el agua conserva la dirección con la que ingresa (es decir, se comporta como la superposición de una pieza de tipo 5 con una de tipo 6).

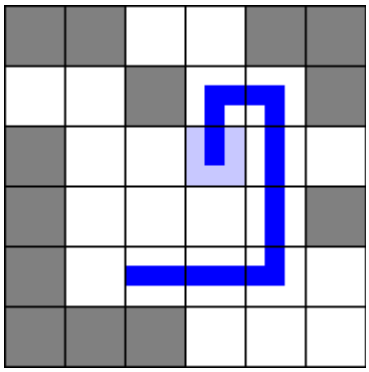
Cualquier combinación de piezas que genere un camino desde el origen hacia afuera de la grilla es una solución al problema. De todas las soluciones posibles, se busca obtener aquella de mayor longitud. Todas las piezas utilizadas aportan 1 a la longitud, salvo la pieza tipo 7 que puede aportar 1 o 2 según una o ambas partes estén siendo utilizadas.

La aplicación debe poder obtener:

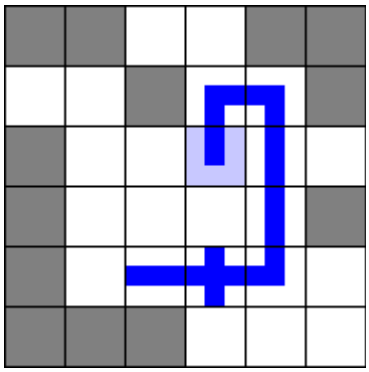
- La solución exacta al problema. Es decir, el camino de mayor longitud posible que comience en la celda de origen y termine en un borde de la grilla. Si existe más de un camino máximo, basta con obtener solamente uno de ellos.

- Una solución aproximada. Es decir, un camino que comience en la celda de origen y termine en un borde de la grilla pero que no necesariamente sea el máximo. Para hacer esto se debe implementar un algoritmo de tipo **hill-climbing** que intente escapar a los máximos locales. En este caso, el usuario deberá indicar el tiempo máximo de ejecución permitido. Pasado este tiempo, se debe retornar la mejor solución encontrada hasta el momento.

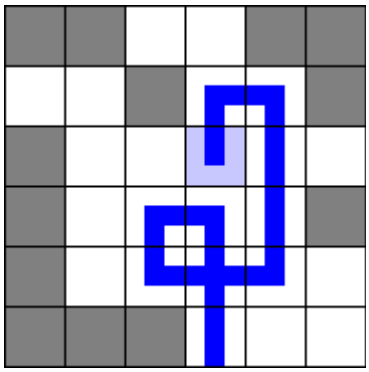
A continuación se muestran algunos tableros de ejemplo:



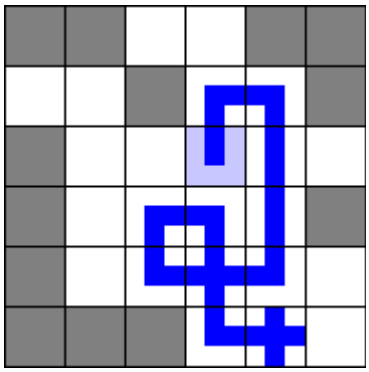
Camino de longitud 8 que no es solución.



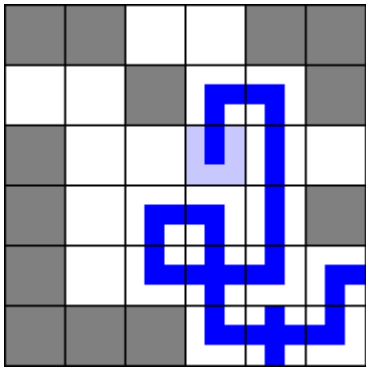
Camino de longitud 8 que no es solución.



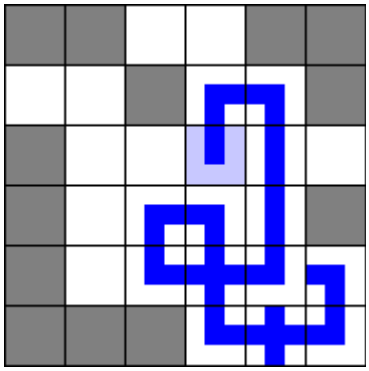
Camino de longitud 12 que es solución.



Camino de longitud 13 que no es solución.



Camino de longitud 15 que es solución.



Camino de longitud 15 que no es solución.

Archivos de entrada

El archivo de entrada describe la estructura del tablero. La primera línea indica las dimensiones de la grilla (dos números enteros separados por coma, indicando primero la cantidad de filas y luego la cantidad de columnas).

Luego a partir de la segunda línea se debe especificar una matriz de caracteres que representa a las celdas de la grilla. Un espacio en blanco indica que no hay nada, un numeral indica una pared. La celda de origen se representa con uno de los siguientes caracteres, según la dirección inicial que tiene el agua que sale de la misma: norte (N), sur (S), este (E), oeste (W). Solamente puede existir una única celda de tipo origen. La cantidad de líneas del archivo utilizadas para describir el tablero es la cantidad de filas de la grilla.

Luego, en las siguientes 7 líneas se debe indicar cuántas piezas de cada tipo hay disponibles para utilizar (un número entero en cada línea). El orden coincide con el tipo de pieza según la tabla de la sección anterior.

El siguiente es un ejemplo de archivo de entrada con el formato indicado:

```
6,6
##  ##
  #  #
#  N
#    #
#
###
1
2
1
3
4
3
1
```

Sintaxis de la línea de comandos

La sintaxis para el uso de la aplicación es la siguiente:

```
# java -jar tpe.jar archivo metodo parametros
```

donde *archivo* es el nombre del archivo de nivel a procesar y *método* es el método a utilizar para encontrar la solución. Hay dos posibles valores:

- **exact**: busca la solución exacta.
- **approx**: busca una solución aproximada. Si se utiliza este método se debe indicar como siguiente parámetro el tiempo (en minutos) que tiene disponible el algoritmo para retornar una solución.

En ambos casos al encontrar la solución se debe imprimir por consola el tiempo que se tardó y la longitud de la solución encontrada. Adicionalmente, se debe graficar en una ventana el tablero con la solución.

En ambos casos se puede agregar opcionalmente el parámetro *progress*, que indica que se desea ver paso a paso el progreso del algoritmo en la búsqueda de la solución. En este caso se debe mostrar la ventana desde el principio, con todas las combinaciones que se intentan en cada paso del algoritmo. En este caso se debe agregar una demora de 100ms entre cada paso para que pueda ser visible la solución parcial.

Ejemplos de uso:

```
# java -jar tpe.jar example.txt exact
```

Encuentra una solución exacta al nivel almacenado en el archivo *example.txt*. Una vez encontrada, la grafica en una ventana e imprime por consola el tiempo total y la longitud de la solución.

```
# java -jar tpe.jar example.txt approx 3
```

Encuentra una solución aproximada al tablero almacenado en el archivo *example.txt* utilizando hill climbing, en un máximo de 3 minutos. Una vez pasados los 3 minutos, imprime por consola el tiempo que demoró en encontrar la solución obtenida (no el tiempo total de ejecución) y la longitud de la misma, y muestra en una ventana el tablero con la solución obtenida.

```
# java -jar tpe.jar example.txt exact progress
```

Encuentra una solución exacta al nivel almacenado en el archivo *example.txt*. Muestra en una ventana el estado del tablero en cada paso del algoritmo. Una vez encontrada la solución final, la grafica e imprime por consola el tiempo total y la longitud de la solución.

Entrega

El trabajo se realizará en grupos de hasta 3 integrantes. La entrega del código fuente se deberá realizar a través del repositorio GIT provisto por la cátedra, antes del **martes 2 de junio a las 21 hs**. Cada grupo deberá enviar un mail a la cátedra confirmando la entrega.

Cada grupo deberá enviar un mail a eda@it.itba.edu.ar indicando los integrantes para poder crear los repositorios.

El código entregado debe contener un buildfile de Ant, cuyo target default genere un archivo jar con la aplicación desarrollada.

El repositorio debe contener el siguiente material:

- Códigos fuentes. Se debe incluir un buildfile de Ant que permita generar el archivo jar de la aplicación.
- Archivos de tableros usados para las pruebas
- Un documento en formato PDF que explique en forma clara:
    1. Algoritmos utilizados
    2. Problemas encontrados durante el desarrollo y decisiones tomadas
    3. Tablas de comparación de tiempos
    4. Conclusiones