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

IIC2133 — Estructuras de Datos y Algoritmos
2019- 1

Tarea 2

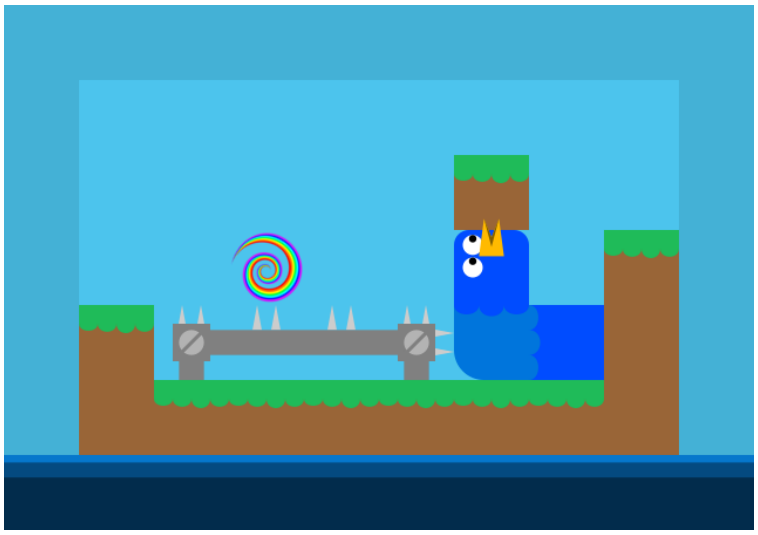
Fecha de entrega código: 19 de mayo
Fecha de entrega informe:

Objetivos

- Implementar un algoritmo de búsqueda en el espacio de estados.
- Crear una implementación propia de una tabla de hash Ad hoc al problema.

Introducción

Snakebird es un juego de puzzle por *Noumenon Games* en que los homónimos pajaros serpiente deben ayudarse mutuamente a comer frutas para luego dirigirse a la meta. Durante su búsqueda por alimento, las serpientes encontrará dificultades y obstáculos que deberá superar para poder llegar al final del nivel. Para esta tarea deberás resolver una versión simplificada del juego **Snakebird**, la cual llamaremos **Snekbirb**. Puedes ver el trailer del juego original para hacerte una idea aquí.



Un ejemplo de un problema de Snekbirb

Snekbirb

Recoger frutas y controlar múltiples Sneks[1] están fuera de lo que nos interesa evaluar en esta tarea, por lo que cada nivel incluye una sola serpiente que debe reptar hacia la meta esquivando los obstáculos.

Mundo y Obstáculos

El mundo que constituye un nivel está compuesto por una matriz de $H \times W$, donde cada celda puede tener un obstáculo o estar vacía.

Existen 2 tipos de celdas obstáculo en este problema:

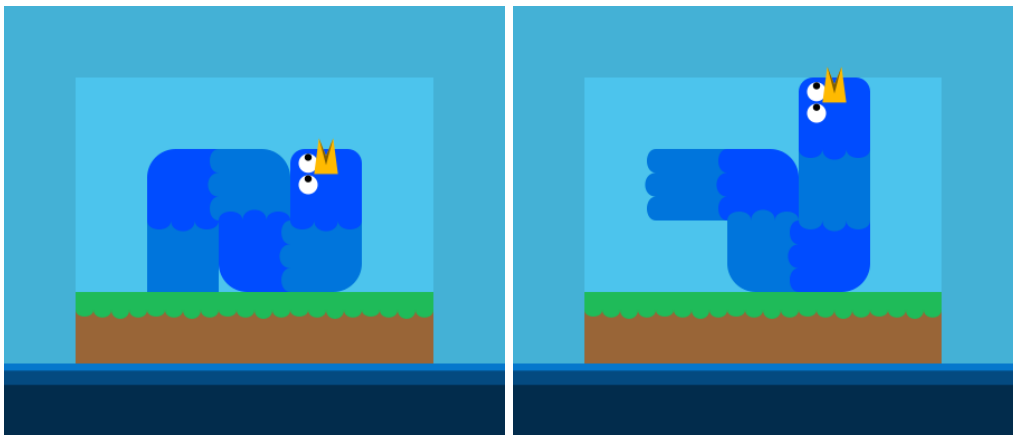
- Espinas: la serpiente muere si alguna parte de su cuerpo está en esta celda.
- Terreno: la serpiente no puede estar en esta celda.

Además, una de las celdas corresponde a la "meta". El puzzle ha sido resuelto cuando la cabeza de la serpiente se encuentra en esta celda.

Serpiente y Movimiento

Una serpiente de largo n está conformada por un bloque de cabeza y $\{s_0, \cdots, s_{n-1}\}$ bloques de cuerpo, donde cada bloque ocupa una celda.

El movimiento de la serpiente consiste en reptar con su cabeza en alguna de las 4 direcciones: `UP`, `DOWN`, `LEFT`, `RIGHT`. El bloque s_0 pasa a estar en la posición donde estaba la cabeza, mientras que el bloque s_i pasa a estar en la posición donde estaba el bloque s_{i-1}.



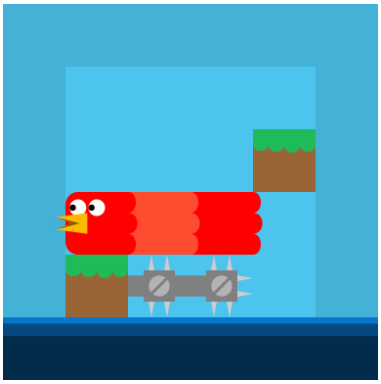
Estado Inicial | Mover hacia arriba (`UP`)

En este mundo existe la gravedad, por lo que la serpiente deberá estar apoyada en una celda de terreno con alguna parte de su cuerpo. De lo contrario caerá hasta quedar apoyada en algún bloque.

[1] Se usará "pajaro", "serpiente" y "snek" indistintamente a lo largo del enunciado para referirse a los *snekbirb*.

Estado inicial. La serpiente está apoyada con su ultimo bloque.

Al moverse a la izquierda (`LEFT`) la serpiente pierde su apoyo



La serpiente cae hasta quedar apoyada en un bloque de terreno

Un movimiento de la serpiente se considera **válido** si luego de llevarlo a cabo la serpiente se encuentra viva y completamente dentro de la matriz de mundo. Aunque un movimiento haga que la serpiente caiga, se sigue considerando como un solo movimiento.

Problema

Deberas escribir un programa en C que dado un mundo y una serpiente encuentre la secuencia de menor cantidad de movimientos válidos que lleva la cabeza de la serpiente a la meta. Para esto se espera que implementes una búsqueda en el espacio de estados con **BFS** y una **Tabla de Hash** para evitar generar estados repetidos.

Se recomienda utilizar la función de hash discutida en el taller de hashing.

Modelación

La modelación del problema no es parte de lo que nos interesa evaluar en esta tarea, por lo que tus ayudantes han preparado una pequeña librería que modela el mapa, los obstáculos, el pájaro y sus movimientos. Esto quiere decir que ya se encuentra programada toda la lógica de los movimientos del snek y su comportamiento con respecto al ambiente.

La modelación del problema consta de 2 estructuras principales: **Snek** y **Board**. Snek controla toda la forma del snekbirb mientras que board controla las interacciones del snekbirb con el mapa. Las funciones que te pueden ser de más utilidad de la modelación están descritas en el archivo **board.h**. Es recomendable leer y entender el ejemplo del código base para empezar la tarea. Cualquier duda se puede hacer en una *ISSUE* en el foro del curso.

Interfaz Gráfica

Se ha preparado una interfaz gráfica para que puedan visualizar como su programa está resolviendo el problema. 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(char* filename)`: abre una ventana con las dimensiones del mapa, rodeado de celdas vacías que representan los límites del mundo. En la fila inferior se dibuja una versión simplificada del agua que rodea la isla representada por el puzzle.

- `watcher_draw_bird(Snek* snek)`: Dibuja el *snek* en ventana, utilizando la modelación dada.
- `watcher_snapshot(char* filename)`: crea una imagen PNG con el contenido actual de la ventana.
- `watcher_close()`: cierra la ventana y libera sus recursos.

Ejecución

Tu programa debe ejecutarse de la siguiente forma:

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

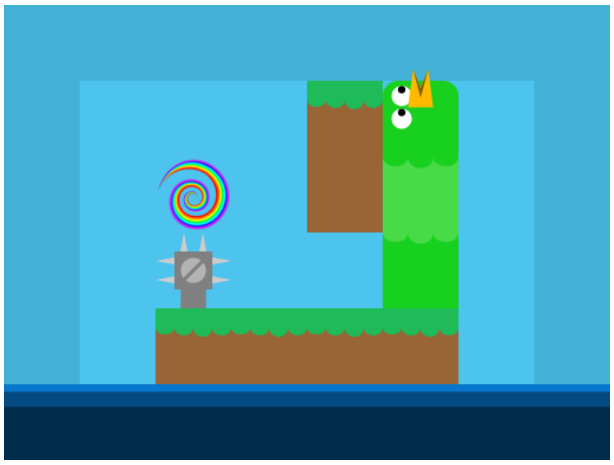
Donde `<test.txt>` corresponde al archivo con el problema a resolver, y `<out.txt>` corresponde al archivo donde deberás escribir la solución.

Input

La primera línea del archivo de test contendrá dos numeros H W, que corresponden al alto y ancho del mundo respectivamente. Las siguientes H líneas contienen W numeros separados por un espacio que representan el contenido de la celda correspondiente. 0 = vacía, 1 = terreno, 2 = espina. La siguiente línea contiene 2 números P_{ROW} P_{COL} que indican las coordenadas del portal de salida (P_{ROW}, P_{COL}). La siguiente línea contiene 3 números S_{ROW} S_{COL} N, los cuales representan la posición de la cabeza de la serpiente (S_{ROW}, S_{COL}) y su largo N. La siguiente línea contiene N números separados por un espacio que representan las direcciones relativas de los bloques de la serpiente respecto al anterior. `0 = UP, 1 = LEFT, 2 = RIGHT, 3 = DOWN, 4 = HEAD` (la cabeza no tiene bloque anterior). Un ejemplo:

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

Esto representa el siguiente puzzle:



Representación gráfica del ejemplo. El portal de la meta se representa como un remolino arcoiris.

Output

El archivo `[output.txt]` deberá tener un número que indica el número de pasos ejecutados y luego las direcciones (0, 1, 2 o 3) en que se mueve la cabeza en cada paso

```
7
2
0
1
1
1
0
1
```

Debe cumplirse que los pasos ejecutados lleven a la solución desde el estado inicial y el número de pasos sea el mínimo.

Evaluación

La nota de tu tarea corresponderá 100% a tu nota de código. Tu programa será probado con diferentes puzzles 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.

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.

Bonus: Tests Lunatic (2 puntos de la nota de código)

Puedes tener puntaje extra en tu nota de código si logras resolver los tests más difíciles de este problema de manera óptima. Para esto se recomienda investigar sobre A*[2] Los tests lunatic se publicarán más adelante.

[2] Asegurate de que tu heurística sea **admisible**. Recuerda que las caidas no aumentan la cuenta de movimientos.