PONTIFICIA UNIVERSIDAD CATÓLICA DE VALPARAÍSO
FACULTAD DE INGENIERÍA
ESCUELA DE INGENIERÍA INFORMÁTICA

Desafio 1 : Light-Up

MARIO JAVIER DOROCHESI OLLINO
17.983.727-7
GONZALO JAVIER TELLO VALENZUELA
18.986.470-1

Profesor: **Ignacio Araya**
Asignatura: **Estructura de Datos Avanzadas Y Algoritmos**

Septiembre, 2020

Índice

1. Solución Propuesta

En la presente sección se procede a explicar la solución propuesta para resolver el puzzle lógico Light-Up.

1.1. Modelado en Grafo

Lo primero que se procede a hacer es modelar el problema como un grafo implícito.

Para ello se define un **estado** como un tablero en un momento cualquiera, donde cada casilla indica o bien un bloque negro, una ampolleta o una casilla vacía.Junto con ello, se define una **acción** como el colocar una ampolleta o bien marcar una casilla con una **X** en una casilla (i,j).

1.2. Pre-Procesamiento del Estado

Lo primero que se procede a hacer una vez recibido el estado es a aplicar ciertas reglas lógicas que nos permiten sin necesidad de explorar el grafo, asegurar que una ampolleta o bien una X va en una determinada casilla. Esta técnica se utiliza con motivo de reducir considerablemente el espacio de búsqueda posterior.

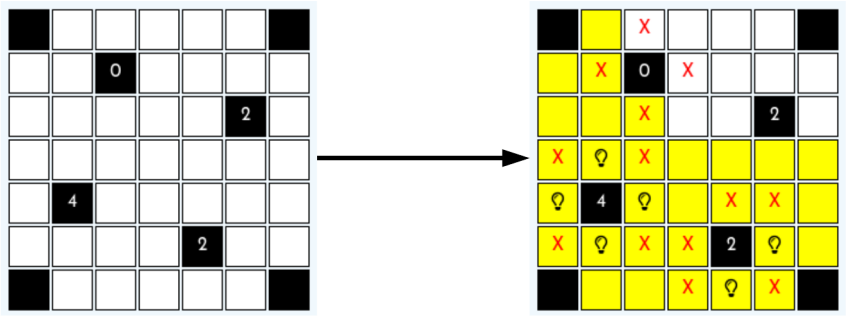

Figura 1: Ejemplo Pre-procesamiento estado

Si bien el ejemplo de la figura anterior es un caso excepcionalmente bueno de pre-procesamiento del estado permite ilustrar claramente la utilidad que tiene esta técnica para reducir el espacio de búsqueda posterior.

1.3. Explorar el Grafo

Una vez teniendo el estado pre-procesado se utiliza como estado de entrada para algunos de los algoritmos de búsqueda implementados durante este desafió, los cuales finalmente entregan un estado final como solución o bien no son capaces de encontrar solución por tiempo limite de ejecución alcanzado.

Los algoritmos de búsqueda implementados son:

- Depth First Search
- Breadth First Search
- Best First Search
- Greedy
- Grasp

1.4. Lógica Principal

A continuación se presenta la lógica a nivel superior del programa en pseudocódigo, para resolver un conjunto de instancias en una carpeta.

```
Por cada instancia en carpeta_instancias:
    estado = cargar_estado(instancia)
    estado = Preprocessor(estado)
    estado_final = BestFirstSearch(estado).resolver()
```

Figura 2: Pseudocódigo lógica principal

1.5. Método de Búsqueda

Como se explico anteriormente, se implementaron en total 5 métodos diferentes de búsqueda, si bien es evidente que cada método tiene un distinto algoritmo a modo de ejemplo se ilustra el funcionamiento del método resolver para **Best First Search**.

```
Funcion Resolver(estado):
    inicializar Pila
    inicializar Visitados
    Pila.agregar(estado)
    Mientras existan elementos en Pila:
        estado = Pila.sacar()
        Si estado.estado_final:
            Retornar estado
        acciones = estado.generar_acciones()

        Para cada accion en acciones:
            estado_s = estado.transicion(accion)
            Si estado_s no se encuentra en Visitados:
                Pila.agregar(estado_s)
                Visitados.agregar(estado_s)
    Retornar Vacio
```

Figura 3: Pseudocódigo búsqueda Best First Search

1.6. Heurística

Finalmente, se presenta en pseudocódigo la heurística utilizada para evaluar estados en los métodos de **Best First Search**, **Greedy** y **Grasp**.

```
Funcion evaluar(estado):
    cantidad_casillas_iluminadas = 0
    suma_bloques_bloqueados = 0
    Para cada casilla en el tablero:
        Si casilla.satisface_restricciones:
            suma_bloques_bloqueados += 1
        Si casilla.esta_iluminada:
            cantidad_casillas_iluminadas +=1
    Retornar suma_bloques_bloqueados * cantidad_casillas_iluminadas
```

Figura 4: Pseudocódigo búsqueda Best First Search