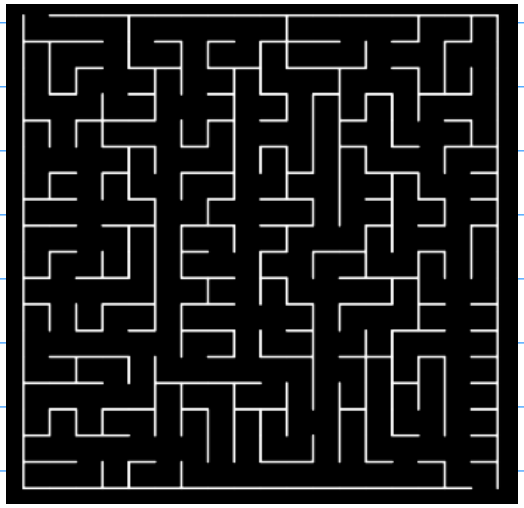
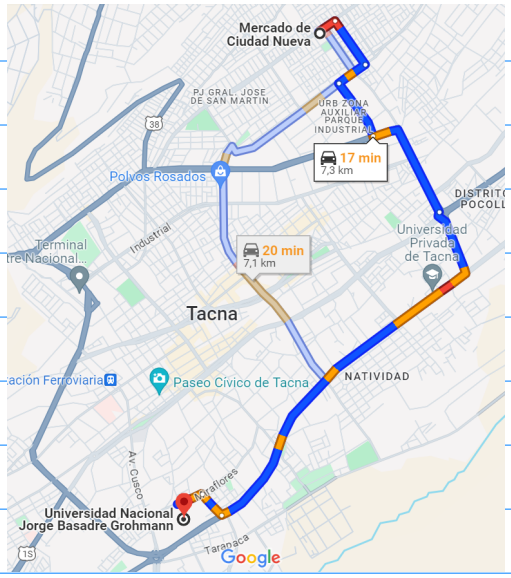
Sistemas Expertos
Docente: Israel Chaparro

Búsqueda I

15-puzle
puede ser resuelto como un problema de búsqueda
ojo: existen puzles sin solución



Laberinto
Objetivo: llegar al final

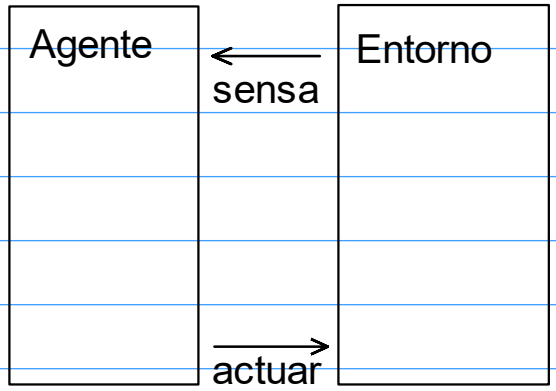


Rutas (Google Maps)
Ir desde A hacia B
Problema REAL

Todos los problemas (3) presentados, son: Problemas de Búsqueda.

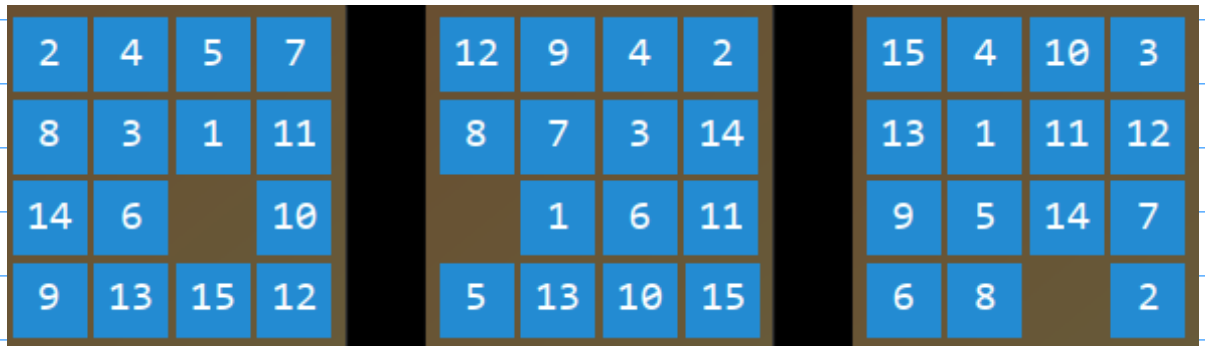
Definición de términos:

- Agente: Entidad que percibe el entorno y actua en su entorno.



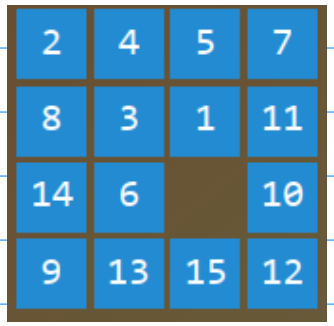
Hay algunas acciones que, dependiendo del entorno, son válidas o no.

- Estado: Alguna configuración del agente y su entorno



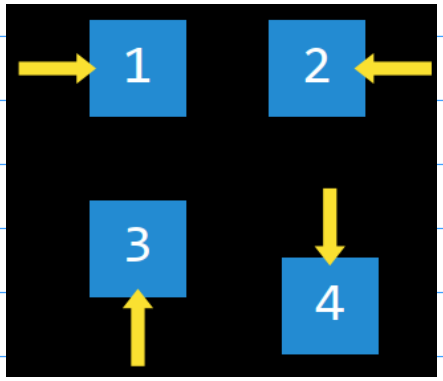
a b c

- Estado inicial: El estado con el que el AGENTE inicia



- Acciones: Elecciones que pueden ser hechas en un estado

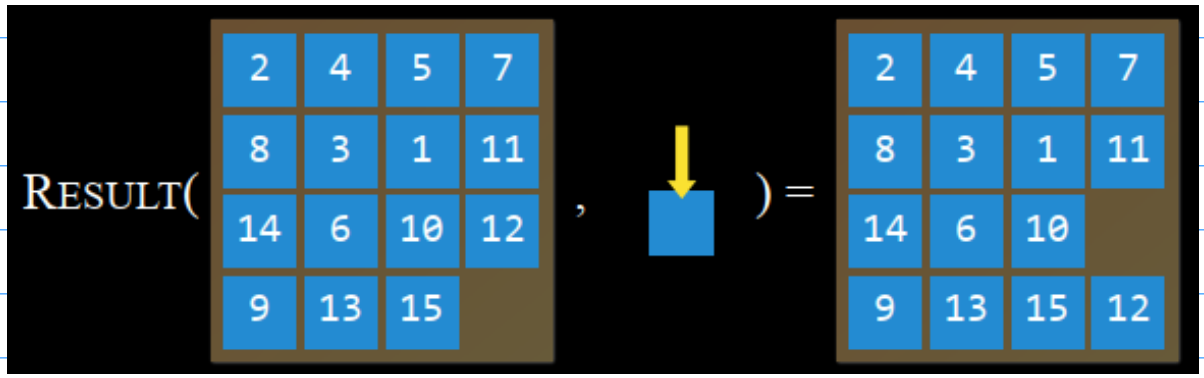
ACTIONS(S) retorne un conjunto de acciones que pueden ser ejecutadas para un estado S



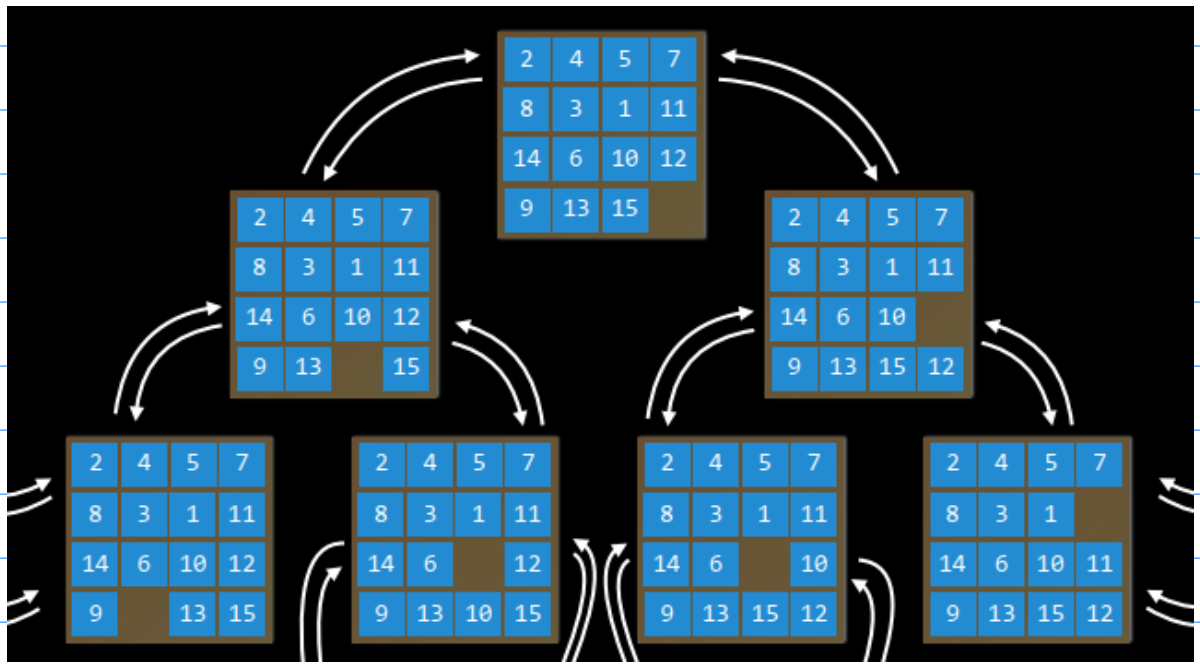
No todas las acciones son posibles, dependen del estado

- Modelo de Transición: es una descripción de que estado resulta de realizar una acción APLICABLE a un estado

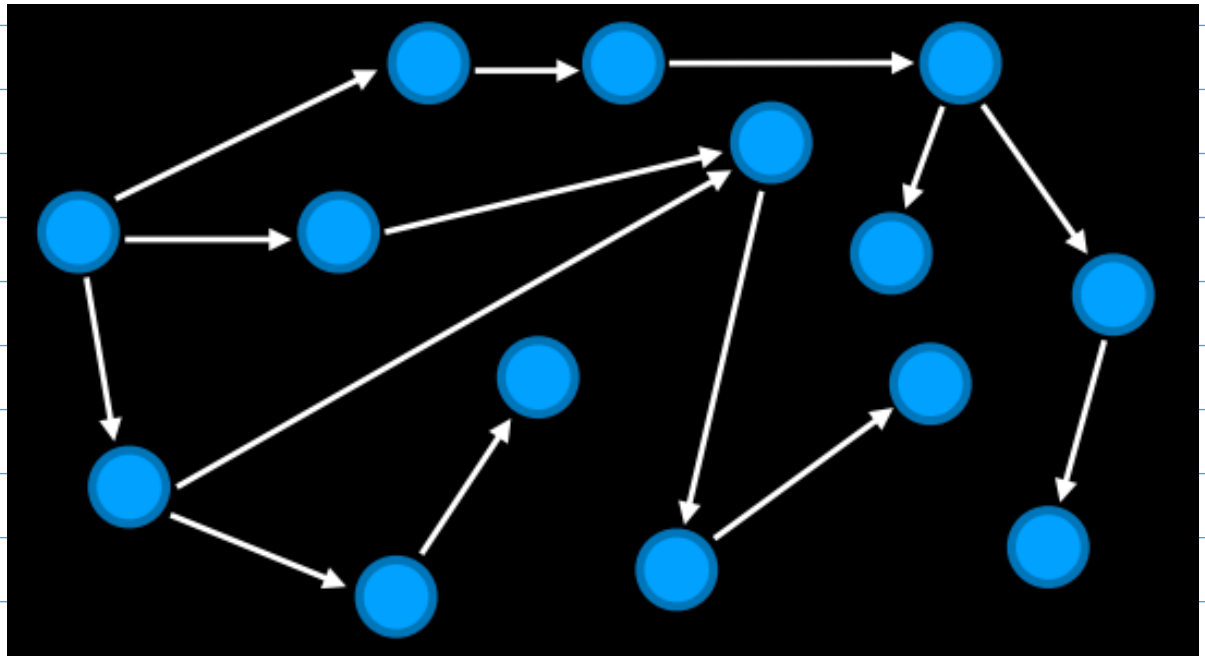
RESULT(S,A): retorna el estado resultante de ejecutar la acción A en el estado S



- Espacio de Estados: El conjunto de todos los estados ALCANZABLES desde el estado inicial y para cualquier secuencia de acciones.



Podemos representar el espacio de estados a través de un grafo.

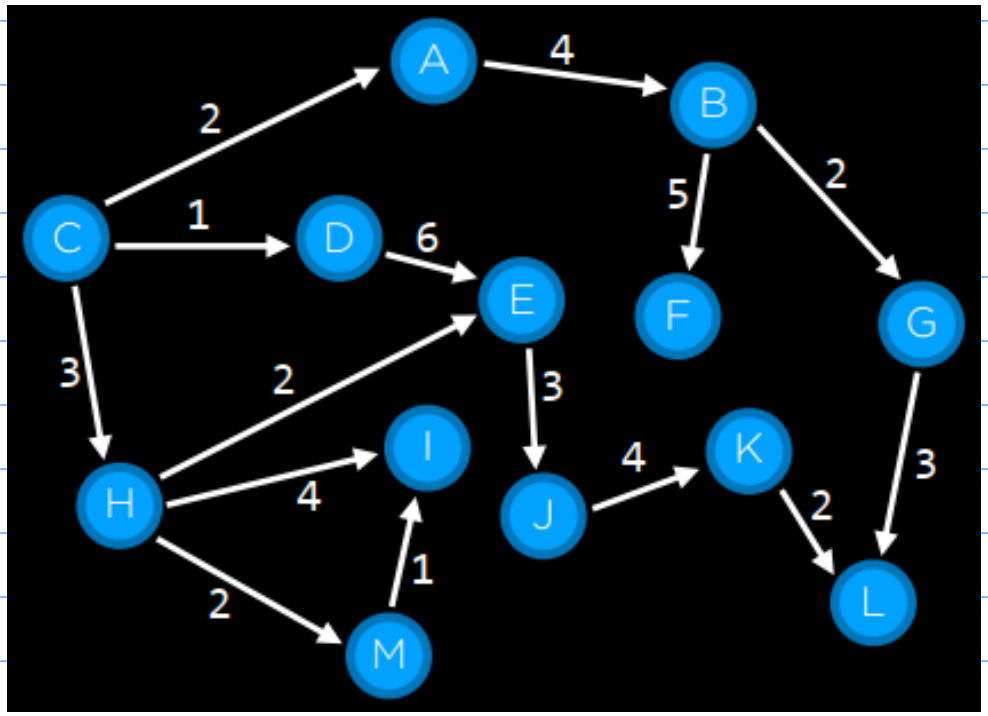


- Test Objetivo: Una forma de determinar si un estado es el estado objetivo.
GOAL(S): Verificar si S es el estado objetivo.

- Costo del Camino: Costo numérico asociado con un camino.

OBJETIVO: Encontrar la solución con costo mínimo (óptima).

¿Cuál es el costo mínimo de llegar de C hacia L?



En algunos problemas existe un costo por camino y en otros se asume que el costo de cada acción es el mismo.

¿Qué consideramos como un problema de búsqueda?

Problemas de Búsqueda:

- Estado inicial
- Acciones
- Modelo de transición
- Prueba objetivo (cuando alcanzamos o no el ESTADO objetivo)
- Función de costo del camino (el costo determinado de un camino)
- Solución: Una secuencia de ACCIONES desde el estado inicial hacia el estado objetivo. (Puede ser cualquiera)

Solución Óptima: Es la solución que tiene el costo de camino más pequeño entre todas las soluciones.

¿Cómo resolvemos el problema?

Representar toda la data a través de una estructura de datos.

NODO

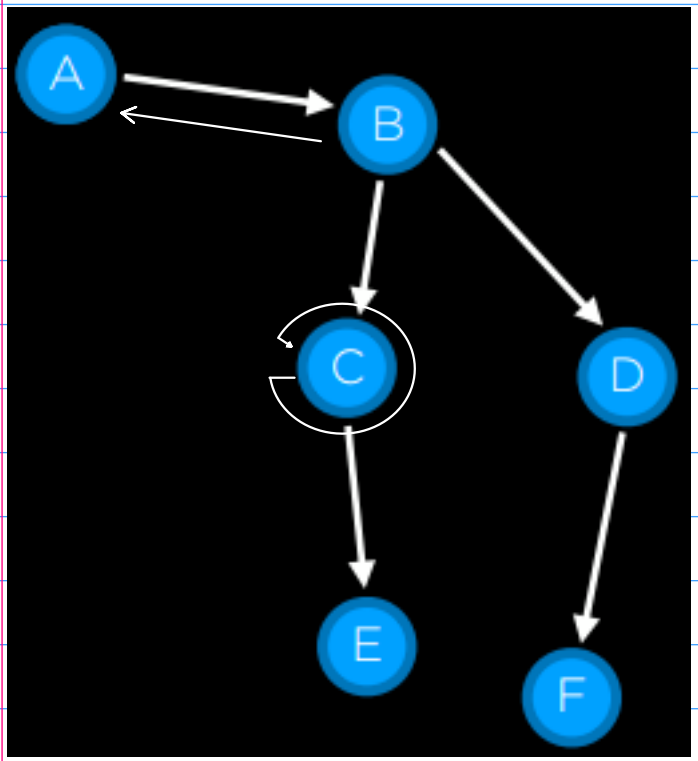
- estado
- padre (nodo que genera este estado)
- acción (la acción aplicada al PADRE para obtener ESTE nodo)
- costo del camino (costo desde el estado inicial hasta ESTE nodo)

¿Cuál es el algoritmo para solucionar el problema?

"Desde un estado (estado inicial), tenemos muchas acciones que realizar y empezamos a explorar los estados resultantes"

```
Inciar con una frontera que contenga el estado inicial.
// la frontera representa todos los nodos (estados) que podemos
// explorar y que aún no hemos explorado
Repetir:
        Si la frontera está vacia, no hay solución.
        Remover un nodo de la frontera.
        Si el nodo contiene el estado objetivo, retornar la solución.
        Expandimos el nodo, añadiendo los nodos resultantes a la frontera
                                    //consideramos las acciones
```

Ejemplo: Encontrar un camino de A a E



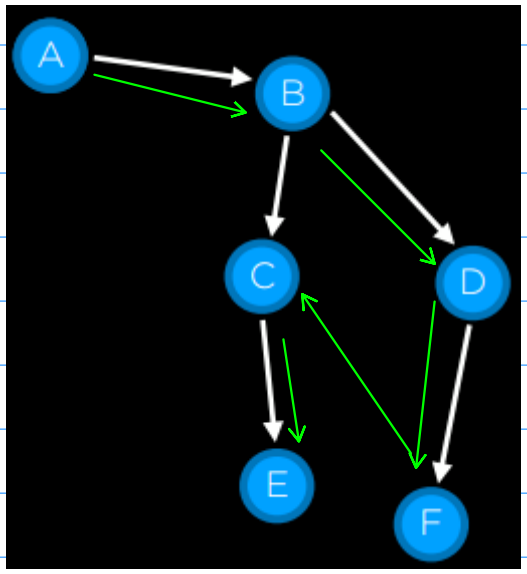
Frontera: [A]
Frontera: [B]
Frontera: [C D]
Frontera: [E D]

Podríamos caer en un bucle

Evitamos caer en un bucle si rastreamos los nodos que ya he visitado para no volver a visitarlos:

```
Inciar con una frontera que contenga el estado inicial.
Iniciar la estructura de datos explorados vacia
Repetir:
        Si la frontera está vacia, no hay solución.
        Remover un nodo de la frontera.
        Si el nodo contiene el estado objetivo, retornar la solución.
        Añadir el nodo a explorados.
        Expandimos el nodo, y añadimos los nodos resultantes a la frontera
                si no están en la frontera o en explorados
```

pila: LIFO (último en entrar es el primero en salir)



Ejemplo: Llegar a de A hacia E

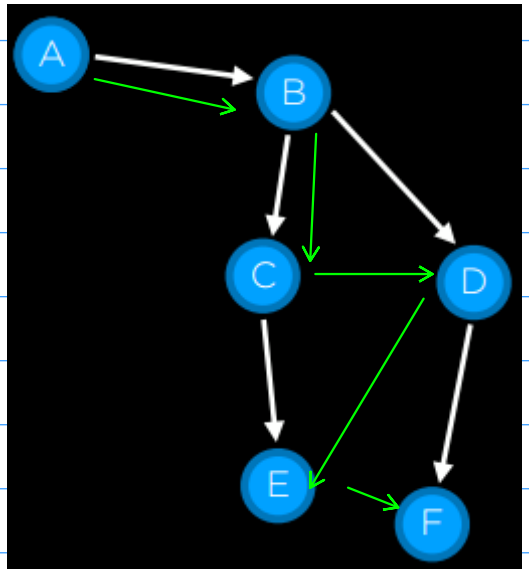
Frontera: [A]

Cuando la estructura de datos frontera es una pila, entonces se realiza una Búsqueda en Profundidad

Depth-First Search (DFS)
Es un algoritmo de búsqueda que siempre expande el nodo más profundo.

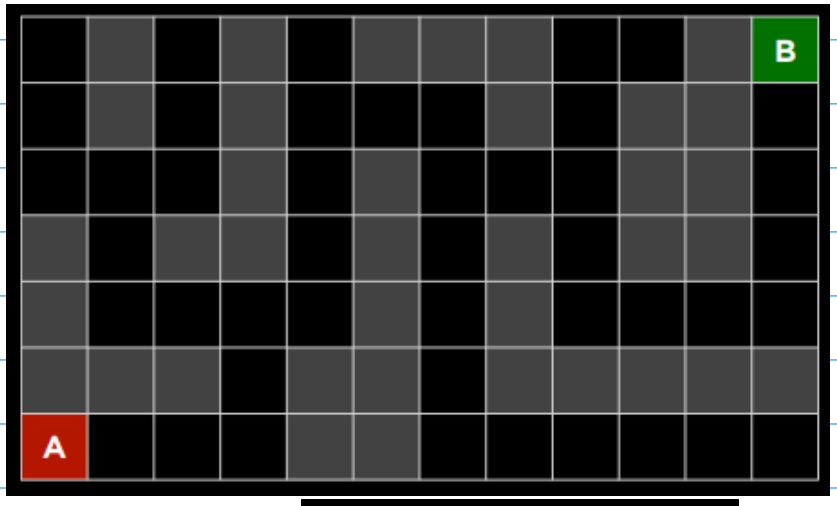
¿Otra opción?
BFS: Breadth-First Search (Búsqueda en anchura)
Es un algoritmo de búsqueda que siempre expande el nodo menos profundo o más superficial.

cola: FIFO (primero en entrar es el primero en salir)



Ejemplo: Llegar a de A hacia E

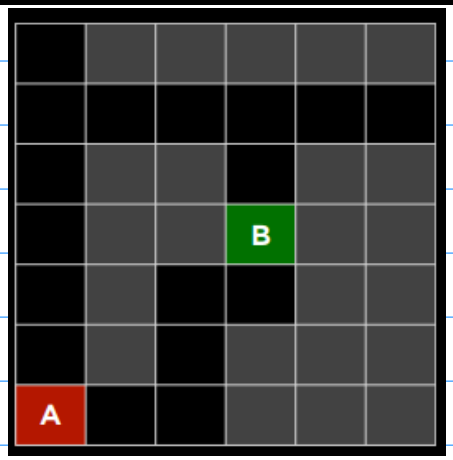
Frontera: [A]



Búsqueda en Profundidad vs Búsqueda en Anchura

(1) DFS vs (4) BFS

¿Encontrará siempre la solución óptima?



(2) DFS: No necesariamente, puede encontrar B por arriba.

(3) BFS: Si, si todos los costos son iguales.

¿Cuál ahorrará más memoria?

Existen 2 tipos de algoritmos de búsqueda:
- Búsqueda no-informada: Estrategia de búsqueda que NO utiliza un conocimiento específico del problema (DFS, BFS).
- Búsqueda informada: Estrategia de búsqueda que utiliza un conocimiento específico del proboema para encontrar una solución MÁS EFICIENTEMENTE.

¿Qué tipo de información nos interesaría?

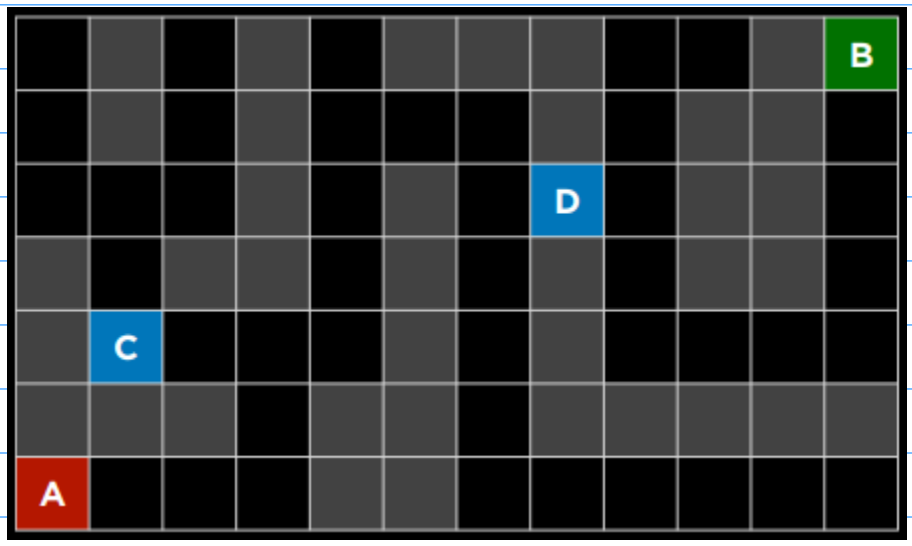
GREEDY BEST-FIRST SEARCH O
BÚSQUEDA CODICIOSA DEL MEJOR-PRIMERO

Algoritmo de búsqueda que EXPANDE el nodo más cercano al objetivo, estimado por una función heurística h(n).

Heurística: Estrategia o método que se utiliza para encontrar soluciones a problemas de una manera más rápida y eficiente, aunque no siempre la solución encontrada será más óptima.

¿Cómo saber cuál está más cerca al objetivo entre C y D?

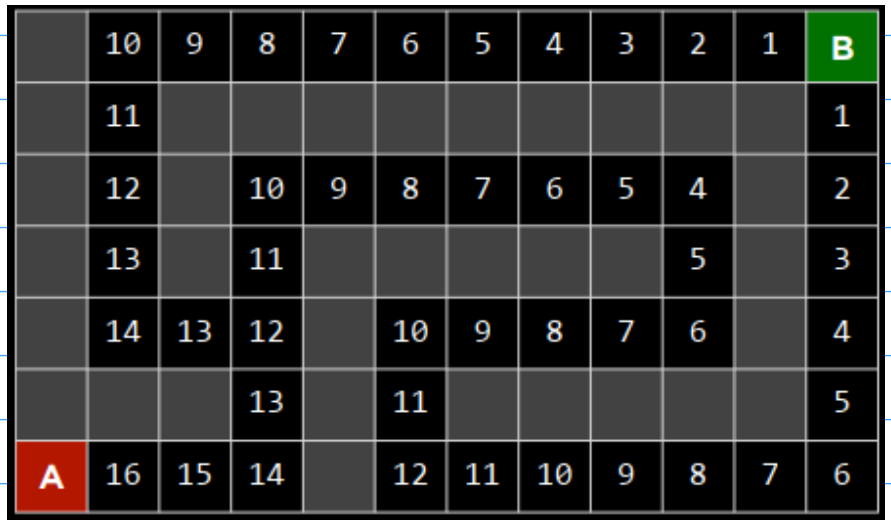
Distancia euclidiana?



Distancia de manhattan: No es una garantía, sino un aproximado, no siempre es acertado.

¿Es mejor que BFS? Probablemnete sí.
En realidad, depende de la heurística y encontrar la heurística CORRECTA puede ser dificil.

¿Siempre encuentra el camino óptimo?

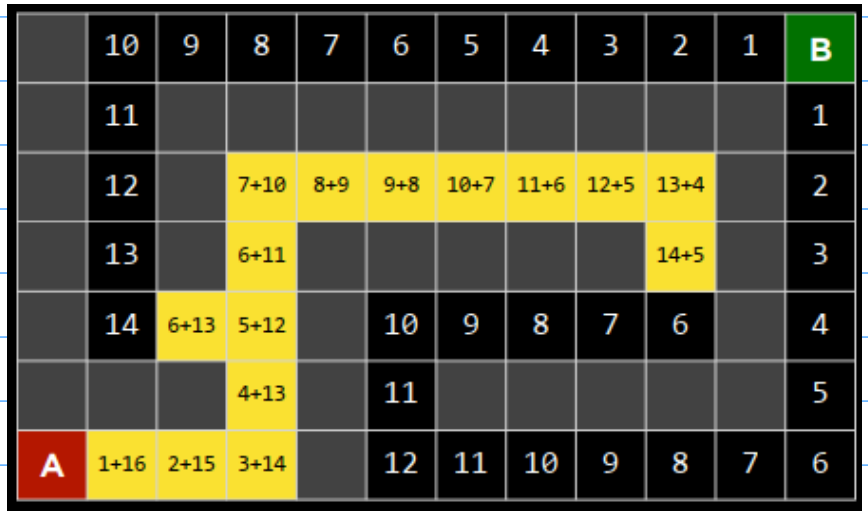


Al ser Greedy (codicioso), siempre toma la mejor decisión LOCALMENTE.
No necesariamente la mejor en el futuro.

¿Cómo podríamos mejorar el algoritmo?

A* SEARCH:
Algoritmo de búsqueda que epande el nodo con el mejor valor g(n)+h(n).

g(n): costo de alcanzar el nodo (¿cuánto me costo llegar hasta aquí?)
h(n): costo estimado al objetivo



Nos entrega la solución óptima si:
- h(n) es admisible (nunca sobreestima el costo verdadero).
- h(n) es consistente (h(n) <= h(n.succesor)+c)