Deusto

Unidad 2:

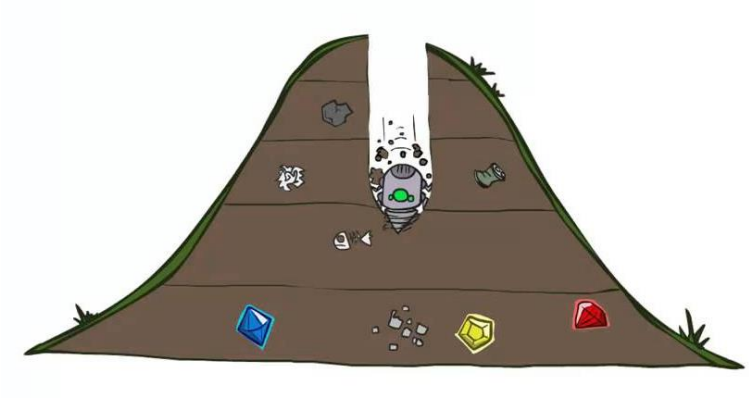
Resolución de problemas mediante búsquedas y heurísticas

Deusto

2.2 Algoritmos no informados

Introducción

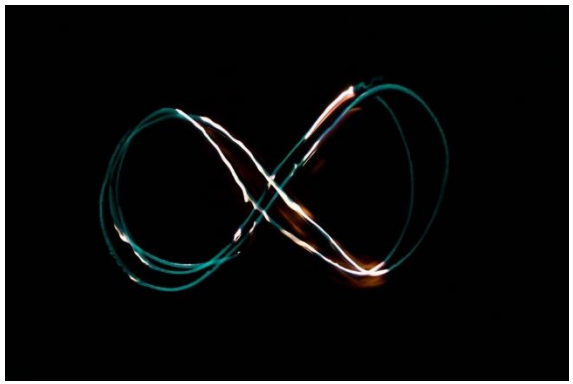
- Búsqueda no informada: no guiada, ciega o por fuerza bruta.
- Sin información sobre el dominio del problema, **solo representación**.
- Estos algoritmos intentan **todas las opciones posibles** en cada nodo.



https://www.youtube.com/watch?v=bSv4CWMTeR0

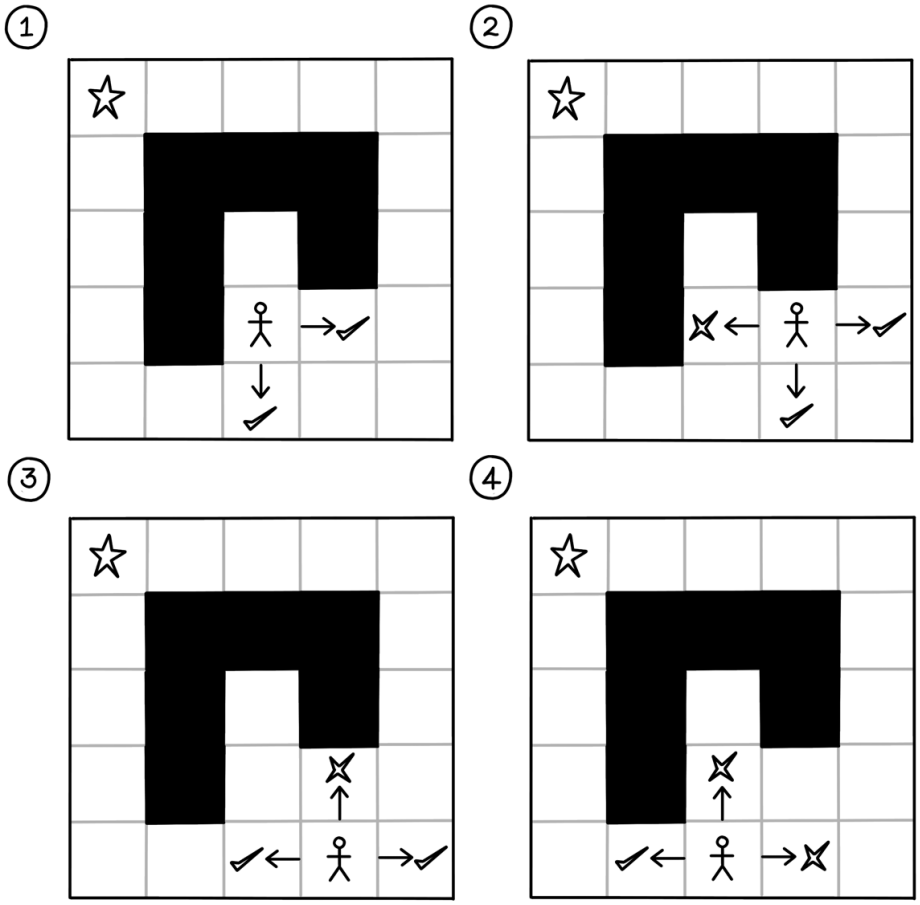
Restricciones: búsqueda de soluciones óptimas (1)

- Para buscar una solución óptima:
  - Imponer restricciones o reglas.
- ¿Objetivo?
  - Evitar caer en bucles: Ciclos en nuestra estructura de datos.



https://nextconf.eu/2018/02/dopamine-loop-dead-long-live-experience-loop/

Restricciones: búsqueda de soluciones óptimas (2)



Hurbans, R. (2020). Grokking Artificial Intelligence Algorithms. Manning Publications.

Entregable 2.1

Representa todos los posibles caminos del laberinto teniendo en cuenta la siguientes restricciones:

- No se puede volver a visitar un nodo que ya haya sido previamente visitado.

- GRUPO A
  - La prioridad de movimiento es norte, sur, oeste y este.
- GRUPO B
  - La prioridad de movimiento es norte, sur, oeste y este.

[Fecha límite: **11/10/2020 12:00**]

Restricciones: búsqueda de soluciones óptimas (3)

- Restricción de prioridad.
  - Influenciará el orden en que son visitados los nodos.
- ¿Representar todas las posibilidades?
  - En la mayoría de los casos no es posible.
- Solución:
  - Generar árboles de forma **iterativa** y **parar** cuando se encuentre una solución.
  - Por **adelantado** es **ineficiente** y computacionalmente complejo.

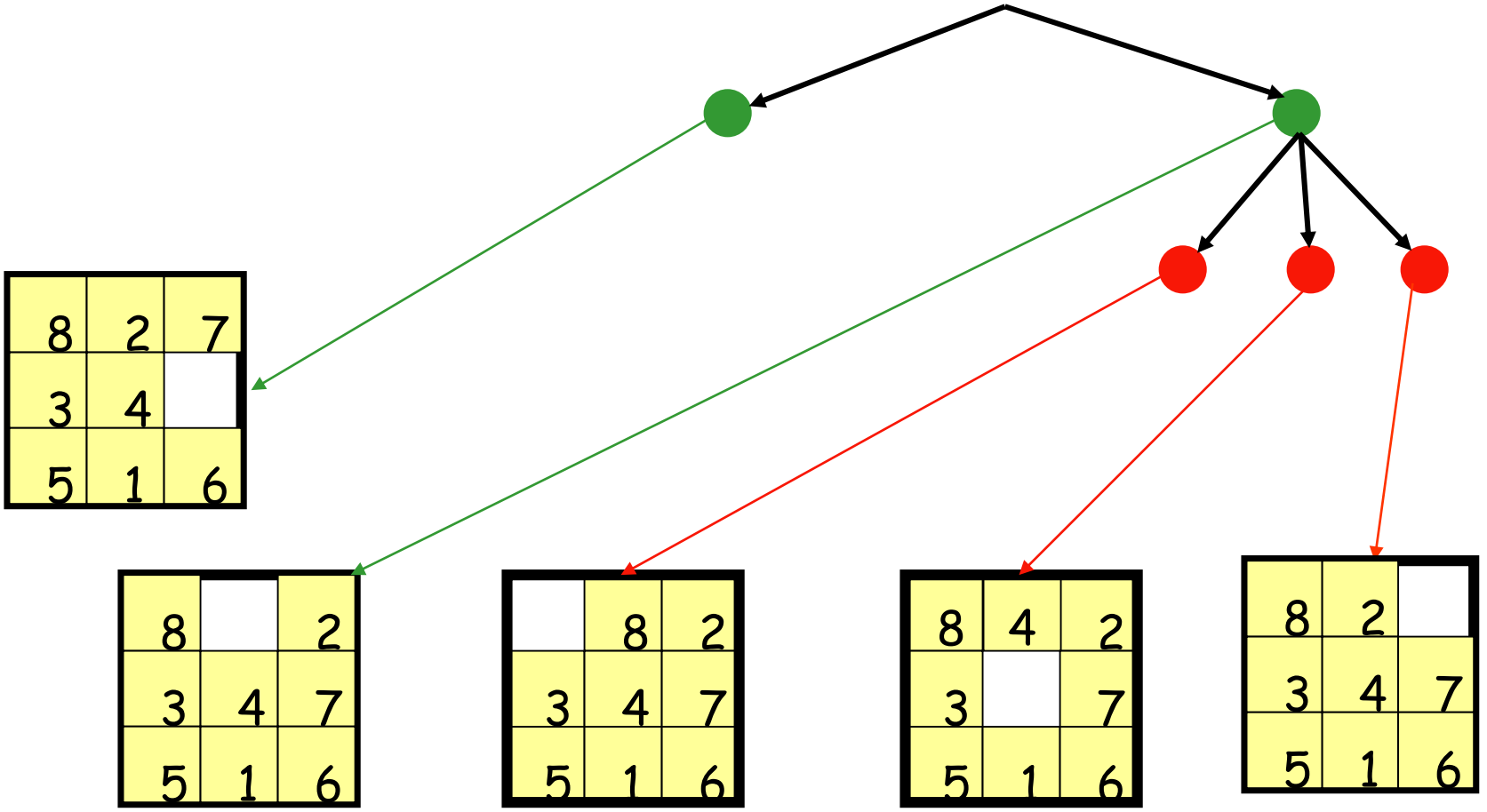
Propiedades de los algoritmos de búsqueda (1)

- **Completitud**: ¿Encuentra siempre una solución si existe?
- **Óptimo**: ¿Encuentra siempre la mejor solución en términos de coste de camino?
- **Complejidad temporal**: ¿Nº de nodos generados?
- **Complejidad espacial**: ¿Nº de nodos en memoria?

Propiedades de los algoritmos de búsqueda (2)

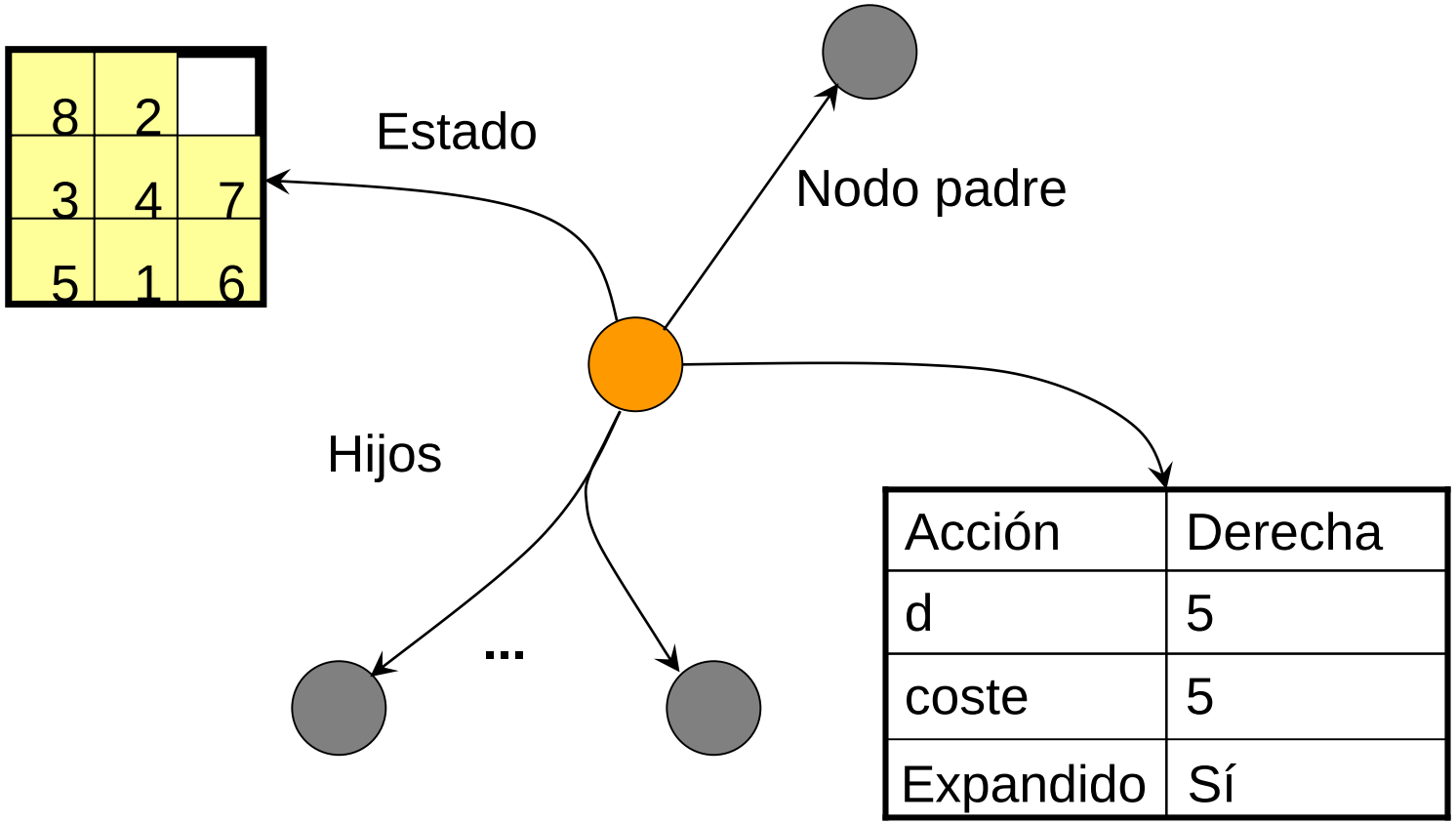
- La complejidad temporal y espacial se mide en términos de:
  - **b**: (*branching factor*) nº máximo de hijos de cada nodo.
  - **d**: (*depth*) profundidad de la solución de menor coste.
  - **m**: (*maximum depth*) máxima profundidad del espacio (puede ser infinito)

Propiedades de los algoritmos de búsqueda (3)



McCoy, K. F. (2010). https://www.eecis.udel.edu/~mccoy/

Propiedades de los algoritmos de búsqueda (4)



McCoy, K. F. (2010). https://www.eecis.udel.edu/~mccoy/

Propiedades de los algoritmos de búsqueda (5)

- **Estado** es una representación de una configuración física. Es la definición de un episodio del problema

- Un **nodo** es una estructura de datos que forma parte de un árbol de búsqueda e incluye **estado**, **nodo padre**, **acción**, **coste** y **profundidad**.

Propiedades de los algoritmos de búsqueda (6)

- **Expansión**:
  - Generar todos los posibles sucesores de un nodo y añadirlos a la frontera.

- **Frontera / Franja [Cola-Lista-Pila]:**
  - Conjunto de nodos que son candidatos a expandir.

- **Estrategia de exploración:**
  - Seleccionar que nodo de la frontera se expande.

Búsqueda básica

1. Make a node with the initial problem state
2. Insert node into the queue data structure
3. WHILE final state not found AND queue is not empty DO
    3.1 Remove first node from the queue
    3.2 IF node contains final state THEN final state found
    3.3 IF node doesn't contain final state THEN
        3.3.1 EXPAND node's state
        3.3.2 Insert successor nodes into queue
4. IF final state found THEN
    4.1 RETURN sequence of actions found
5. ELSE "solution not found"

Función expandir

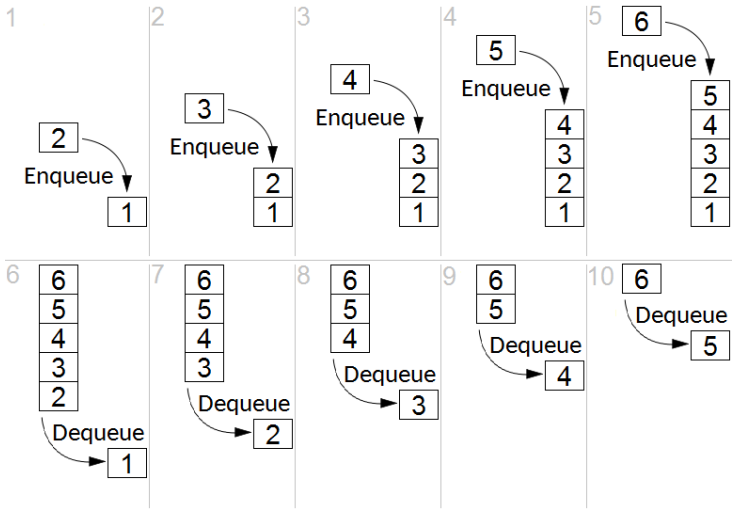
1. Initialize Successors to an empty list.
2. FOR EACH applicable action on state of current node DO
    - 2.1 Make a new successor node and generate successor state
    - 2.2 Successor's state = successor state
    - 2.3 Successor's action = applicable action
    - 2.4 Successor's parent node = current node
    - 2.5 Successor's path-cost = current path cost + action cost
    - 2.6 Successor's depth = current depth + 1
    - 2.7 Add successor node to Successors
3. RETURN Successors

BFS (1)

- *Breadth-First Search*: Búsqueda en anchura.
- Explora todas las opciones en un determinado nivel de profundidad antes de pasar al siguiente.
  - Raíz -> explora cada nodo en ese nivel de profundidad antes de pasar al siguiente.
  - Es necesario visitar todos los nodos hijos de un determinado nivel.
  - Finaliza al llegar al objetivo.

BFS (2)

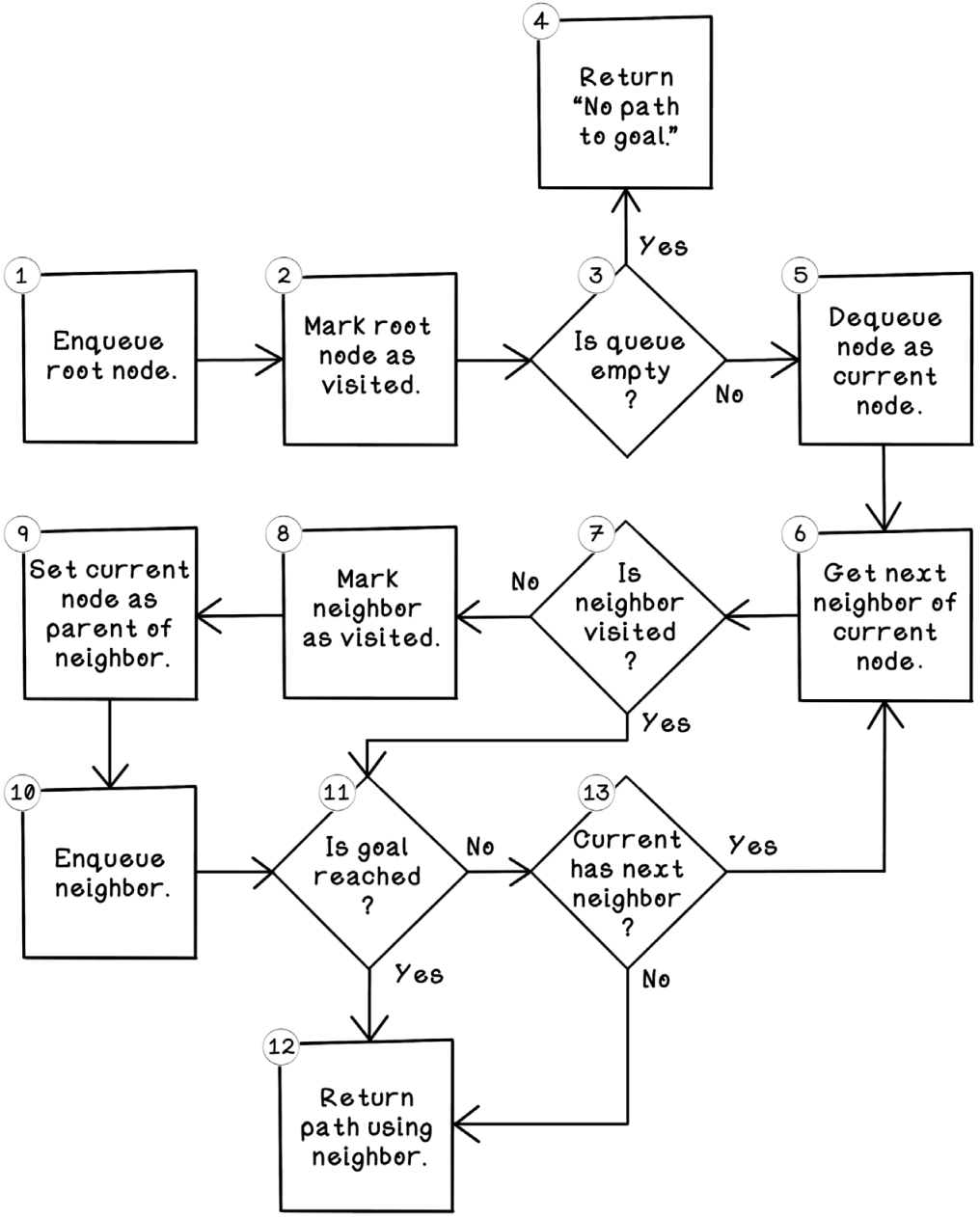
- Para implementarlo se usa una cola tipo FIFO (*first-in first-out*: primero en entrar primero en salir), en el que los nodos de un nivel son procesados, y sus nodos hijos son añadidos a la cola.



https://upload.wikimedia.org/wikipedia/commons/d/d3/Fifo_queue.png

Deusto

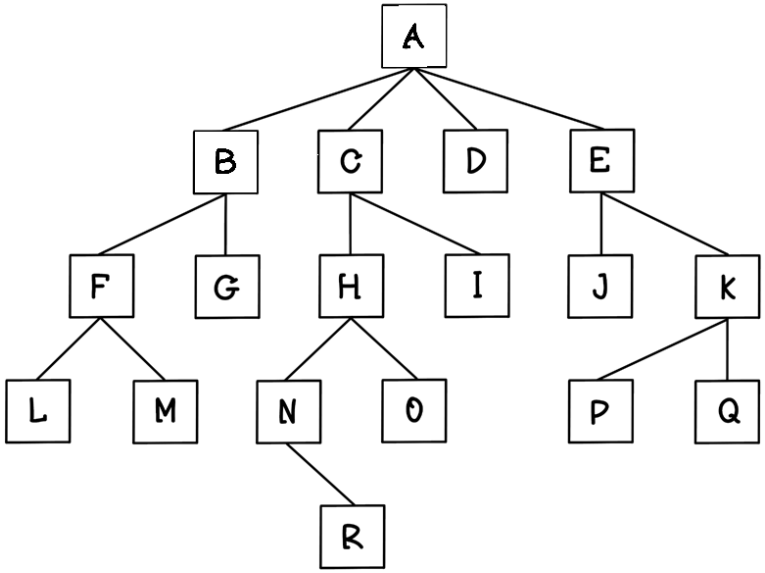
BFS(3)



Hurbans, R. (2020). Grokking Artificial Intelligence Algorithms. Manning Publications.

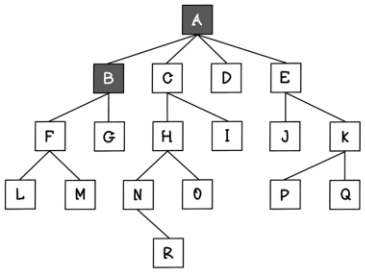
Observación reflexiva #1

¿Cómo se recorrería el siguiente árbol siguiendo el algoritmo BFS?



Observación reflexiva #1

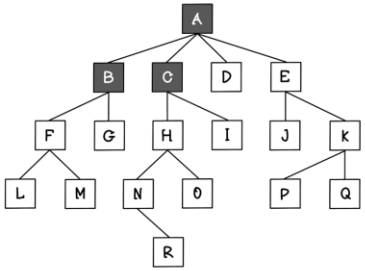
Visit the first child node, B.



Sequence of processing the queue

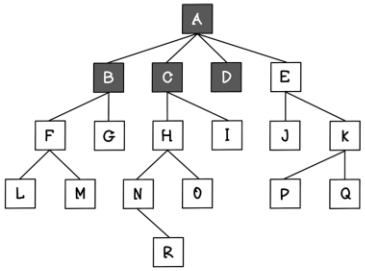
A
B

Visit the next child node, C.



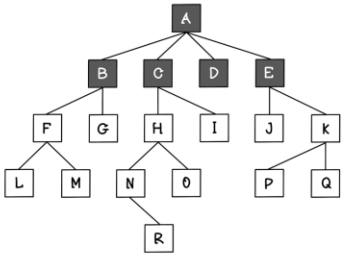
A
B
C

Visit the next child node, D.



A
B
C
D

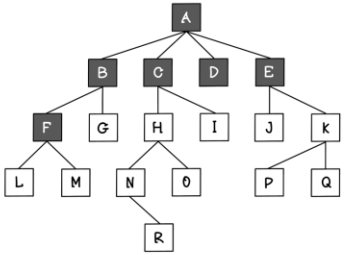
Visit the final node on this depth, E.



Sequence of processing the queue

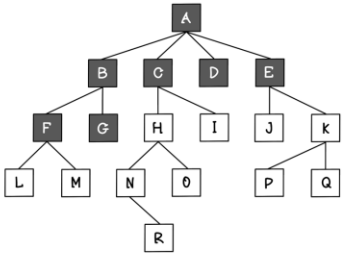
A
B
C
D
E

Visit the first child of the first neighbor of A. This is F, first child of B.



A
B
C
D
E
F

Visit the next child of B. This is G.

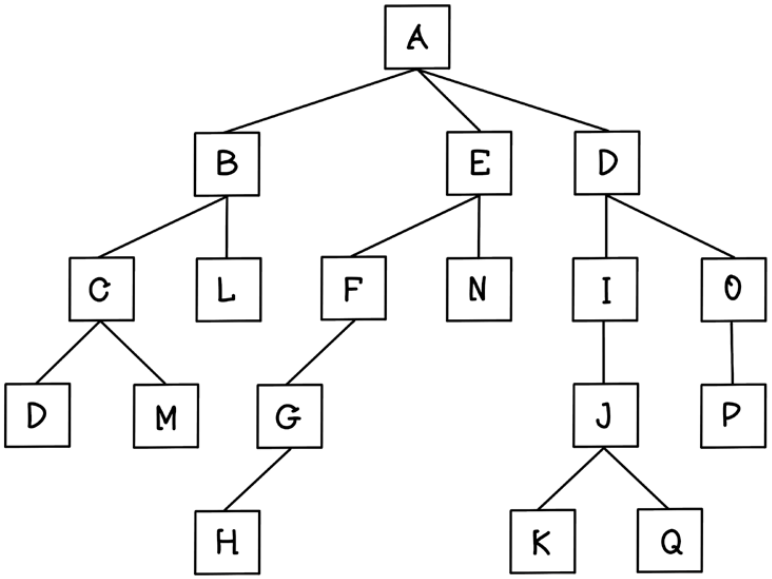


A
B
C
D
E
F
G

Hurbans, R. (2020). Grokking Artificial Intelligence Algorithms. Manning Publications.

Entregable 2.2

¿Cómo se recorrería el siguiente árbol de acuerdo al algoritmo BFS?



[Fecha límite: **11/10/2020 12:00**]

BFS (4)

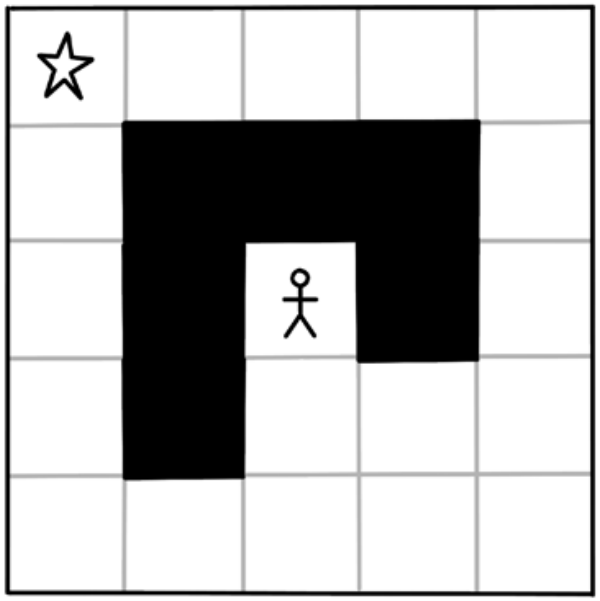
- ¿Cómo se aplicaría el algoritmo BFS en un problema real?
- Ej. Nuestro laberinto
  - Primero, el algoritmo necesita saber la posición inicial.
  - Se evalúan todas las posibles elecciones de movimiento.
  - Se repite la lógica para cada movimiento hasta que el objetivo sea alcanzado.

BFS (5)

- Siguiendo estas directrices, **el algoritmo genera un árbol** con un simple camino al objetivo.
- El número de movimientos necesarios para alcanzar el objetivo se denomina **coste.**
  - Suele ser el mismo que el número de nodos desde la raíz hasta el objetivo.
- El algoritmo visita todos los nodos en cada nivel de profundidad hasta alcanzar el objetivo.
- Después, devuelve el primer camino que lleva al objetivo.
- Puede **no** ser **óptimo**, esa es la razón de que el algoritmo sea **desinformado o ciego**.

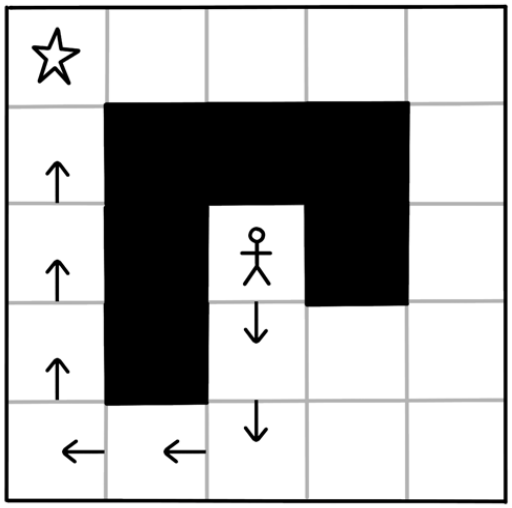
Observación reflexiva #2

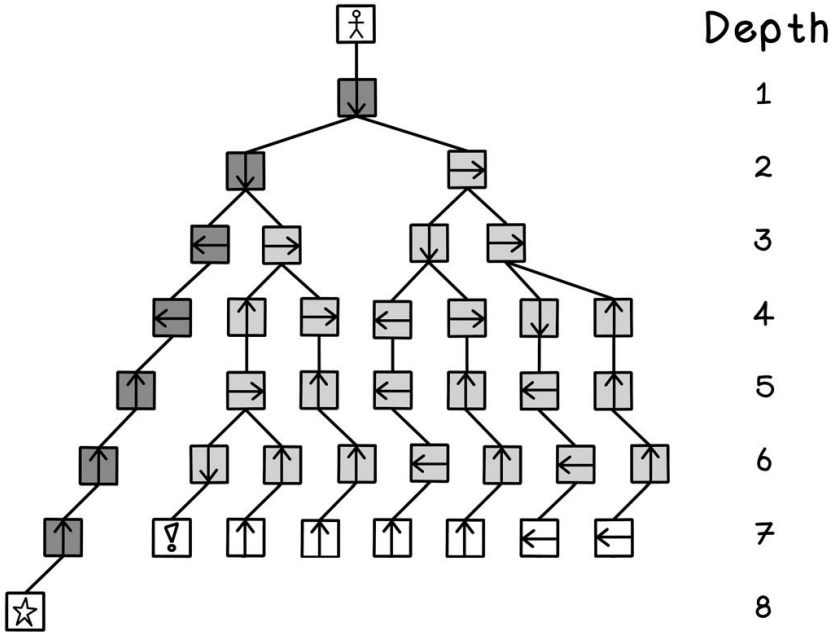
¿Cuál sería el árbol generado a partir del algoritmo BFS en el caso del laberinto?



Observación reflexiva #2

¿Cuál sería el árbol generado a partir del algoritmo BFS en el caso del laberinto?





Hurbans, R. (2020). Grokking Artificial Intelligence Algorithms. Manning Publications.

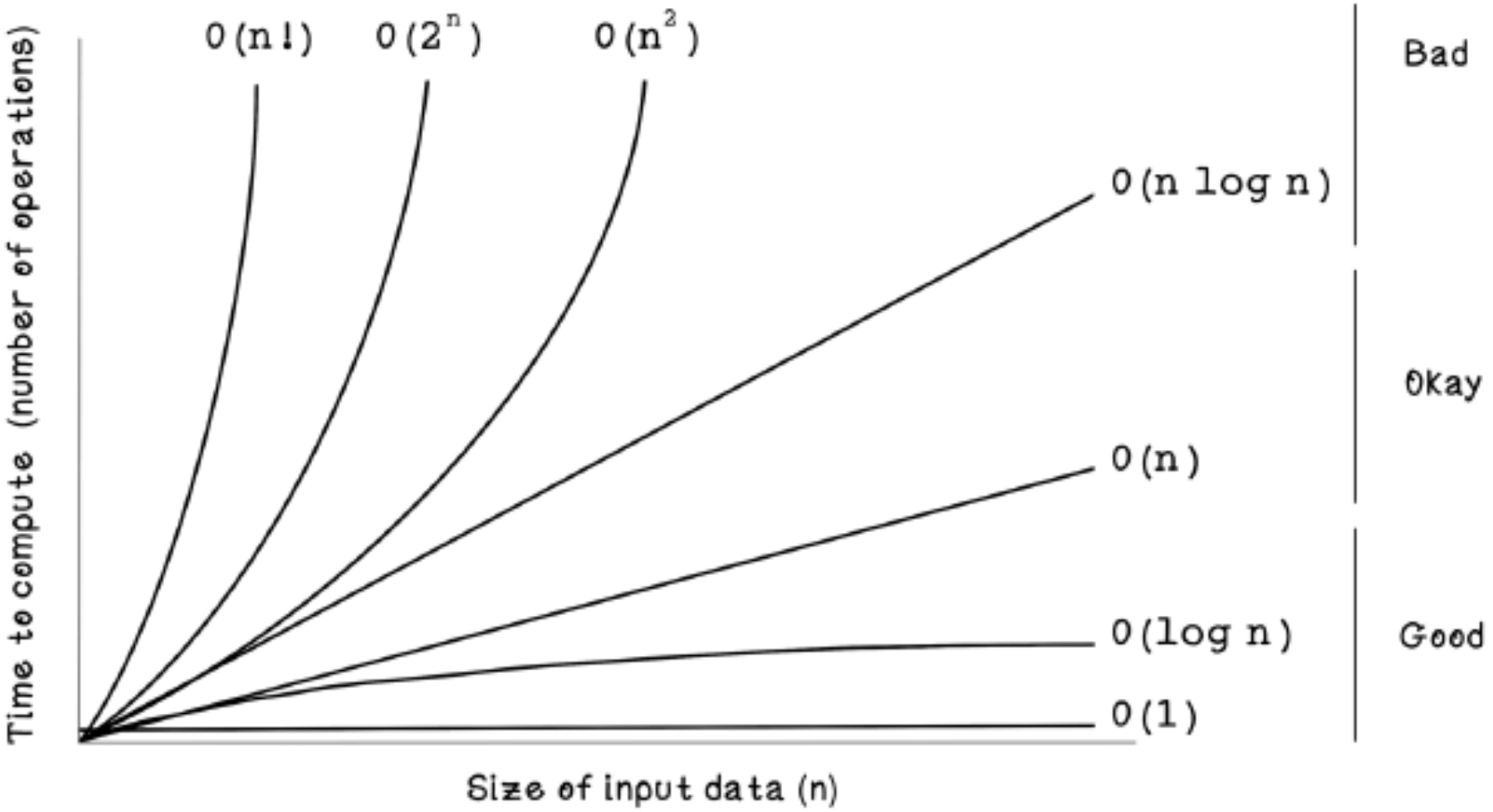
BFS (6)

```
run_bfs(maze, current_point, visited_points):
  let q equal a new queue
  push current_point to q
  mark current_point as visited
  while q is not empty:
    pop q and let current_point equal the returned point
    add available cells north, east, south, and west to a list neighbors
    for each neighbor in neighbors:
      if neighbor is not visited:
        set neighbor parent as current_point
        mark neighbor as visited
        push neighbor to q
        if value at neighbor is the goal:
          return path using neighbor
  return "No path to goal"
```

Hurbans, R. (2020). Grokking Artificial Intelligence Algorithms. Manning Publications.

BFS (7) - Propiedades

- **Completitud**: Siempre que b sea discreto.
- **Óptimo**: Si todas las acciones tienen el mismo coste.
- **Complejidad temporal**: $O(b^d)$
- **Complejidad espacial**: $O(b^d)$ (mantiene cada nodo en memoria)



Hurbans, R. (2020). Grokking Artificial Intelligence Algorithms. Manning Publications.

Observación reflexiva #3

b =10

1000 nodos expandidos por segundo

100 bytes por nodo

Profundidad	Nodos	Tiempo	Memoria
1			
6			
8			

Uniform-cost search: búsqueda de coste uniforme

- BFD funciona bien si todos los caminos tienen el mismo coste.
- ¿Y si no?
  - Se trata de escoger el nodo que tenga menor coste.
  - Cola de prioridad en lugar de FIFO
- En el caso de que todos los nodos tengan el mismo coste:
  - *Uniform-cost search* = BFS

Uniform-cost search - Propiedades

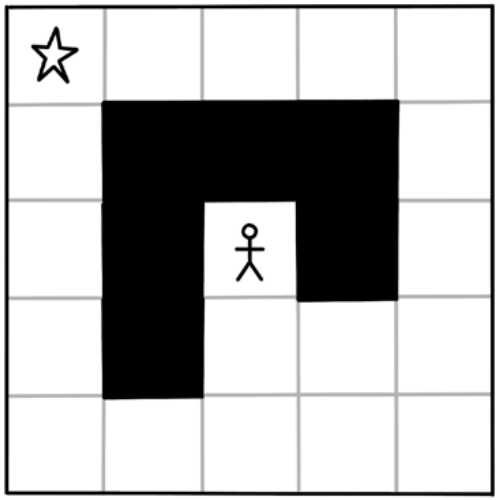
- **Completitud**: Siempre que el coste sea un nº positivo (ε).
- **Óptimo**: Siempre que el coste sea un nº positivo (ε).

C* es el coste de la solución óptima, entonces el nº de nodos generados g <= C* es $O(b^{ceiling\ (c^*/\varepsilon)})$

- **Complejidad temporal**: $O(b^{ceiling\ (c^*/\varepsilon)})$
- **Complejidad espacial**: $O(b^{ceiling\ (c^*/\varepsilon)})$

Observación reflexiva #4

¿Cuál sería el árbol generado a partir del algoritmo *uniform-cost search* en el caso del laberinto? Los movimientos Norte-Sur o Sur-Norte tienen un coste de 5, el resto de movimientos de 1.



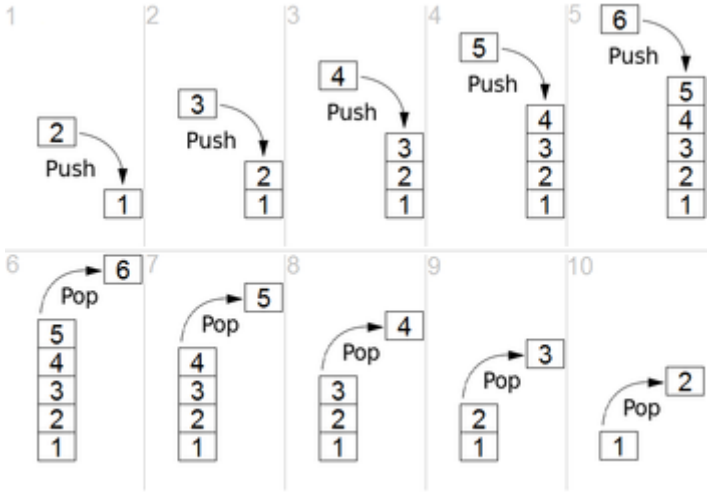
Hurbans, R. (2020). Grokking Artificial Intelligence Algorithms. Manning Publications.

DFS (1)

- *Depth-First Search*: Búsqueda en profundidad.
- Empieza en un nodo y explora todos los caminos del primer hijo.
- Haciendo esto recursivamente alcanza un nodo hoja.
- Repite el proceso para el resto de hijos que ya ha visitado.
- Finaliza al llegar al objetivo.

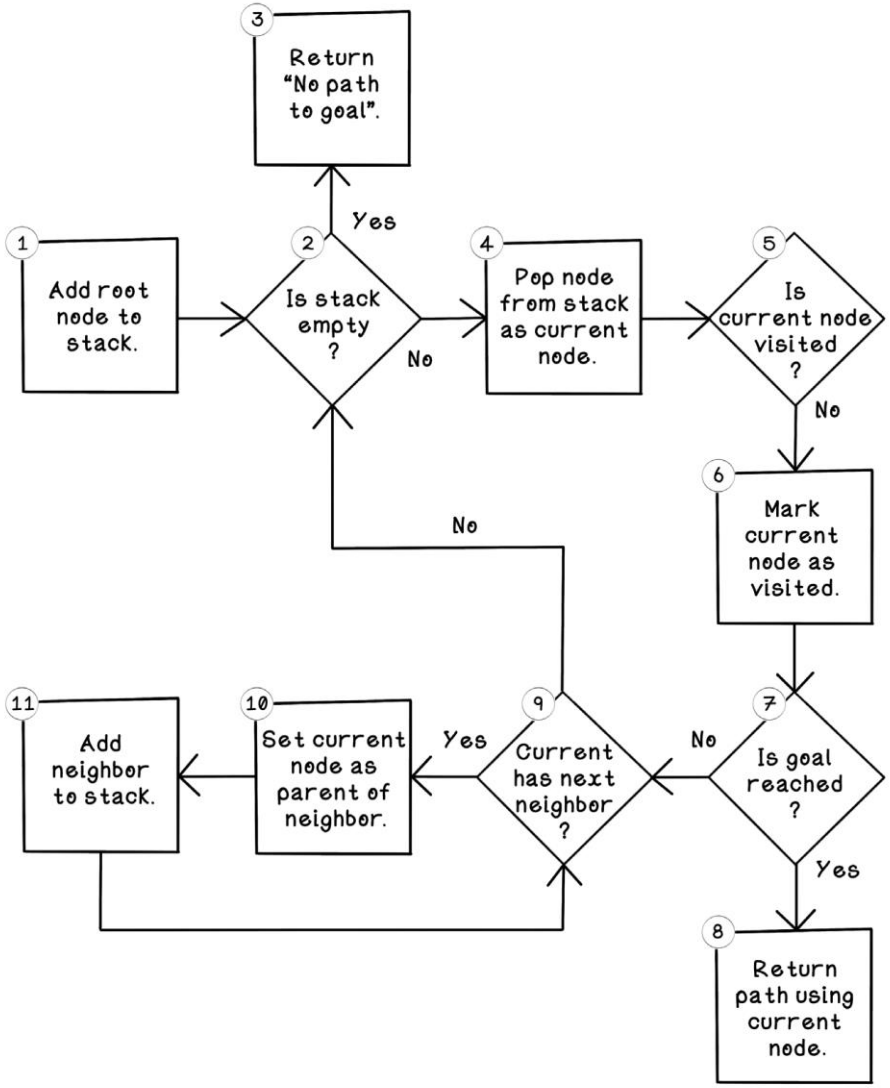
DFS (2)

- Para implementarlo se usa una cola tipo LIFO (*last-in first-out*: último en entrar primero en salir), en el que los **primeros** hijos de cada nodo se van añadiendo (ojo **prioridad**).



https://upload.wikimedia.org/wikipedia/commons/thumb/b/b4/Lifo_stack.png/350px-Lifo_stack.png

DFS (3)



Hurbans, R. (2020). Grokking Artificial Intelligence Algorithms. Manning Publications.

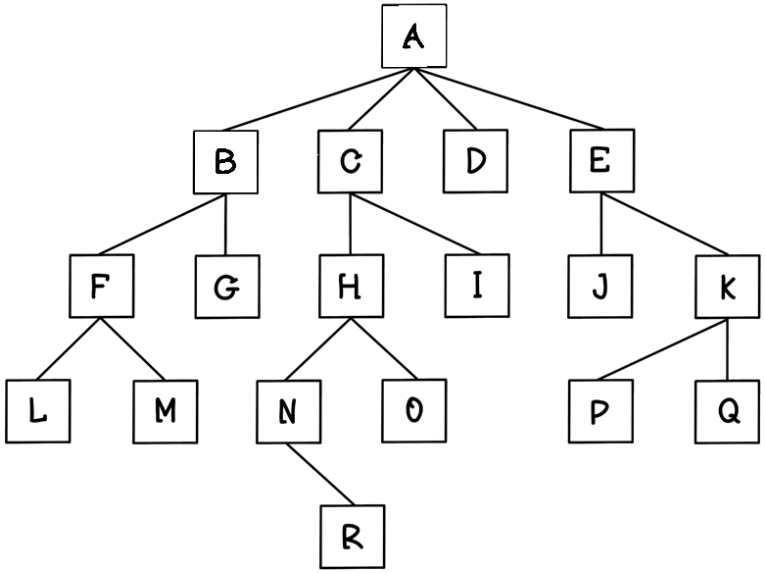
DFS (4)

```
run_dfs(maze, root_point, visited_points):
  let s equal a new stack
  add root_point to s
  while s is not empty
    pop s and let current_point equal the returned point
    if current_point is not visited:
      mark current_point as visited
      if value at current_node is the goal:
        return path using current_point
      else:
        add available cells north, east, south, and west to a list neighbors
        for each neighbor in neighbors:
          set neighbor parent as current_point
          push neighbor to s
  return "No path to goal"
```

Hurbans, R. (2020). Grokking Artificial Intelligence Algorithms. Manning Publications.

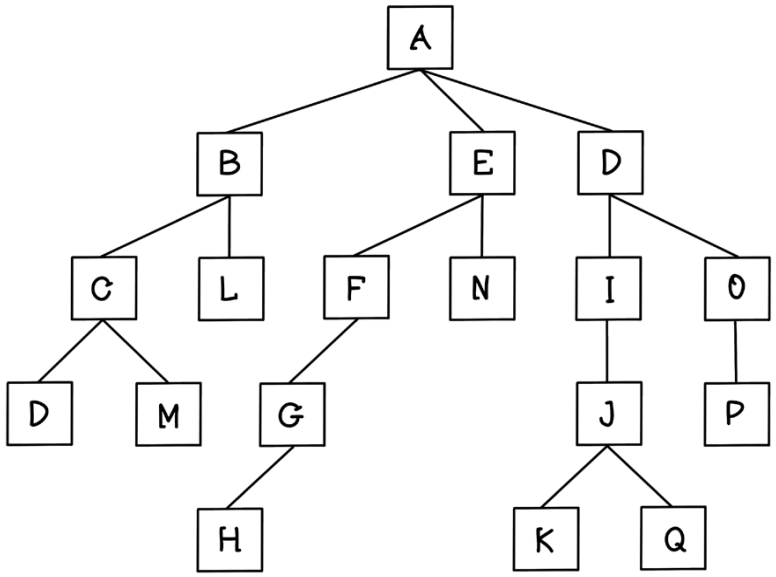
Observación reflexiva #5

¿Cómo se recorrería el siguiente árbol siguiendo el algoritmo DFS?



Entregable 2.3

¿Cómo se recorrería el siguiente árbol de acuerdo al algoritmo DFS?



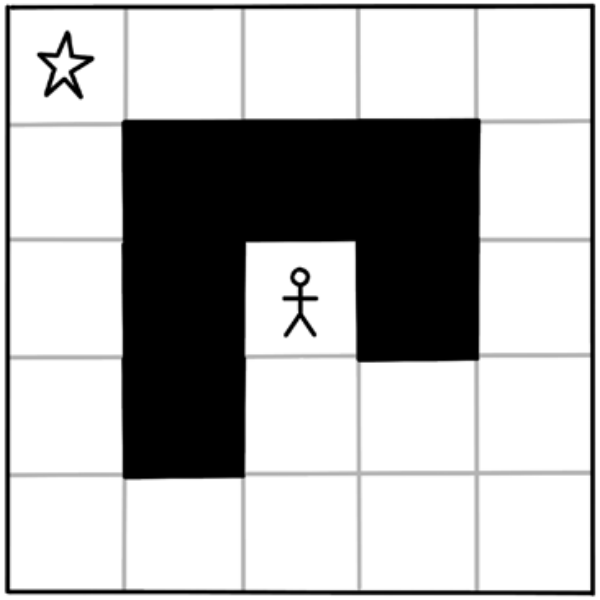
[Fecha límite: **18/10/2020 12:00**]

DFS - Propiedades

- **Completitud**: No es completo en el caso de entorno infinito.
- **Óptimo**: No es óptimo en el caso de entorno infinito.
- **Complejidad temporal**: En el peor caso $O(b^m)$, siendo m la longitud del camino más largo, en el mejor de los casos sería (b-1) d
- **Complejidad espacial**: O(bm)
  - Solo mantiene nodos el camino actual.

Observación reflexiva #6

¿Cuál sería el árbol generado a partir del algoritmo DFS en el caso del laberinto?



Hurbans, R. (2020). Grokking Artificial Intelligence Algorithms. Manning Publications.

DFS limitado

- ¿Y qué pasa si el entorno es continuo?
  - BFS no es tan problemático, pero DFS podría entrar en bucle.
- Solución:
  - Búsqueda en profundidad limitada.
    - Se considera que los nodos por debajo de L no tienen hijos.

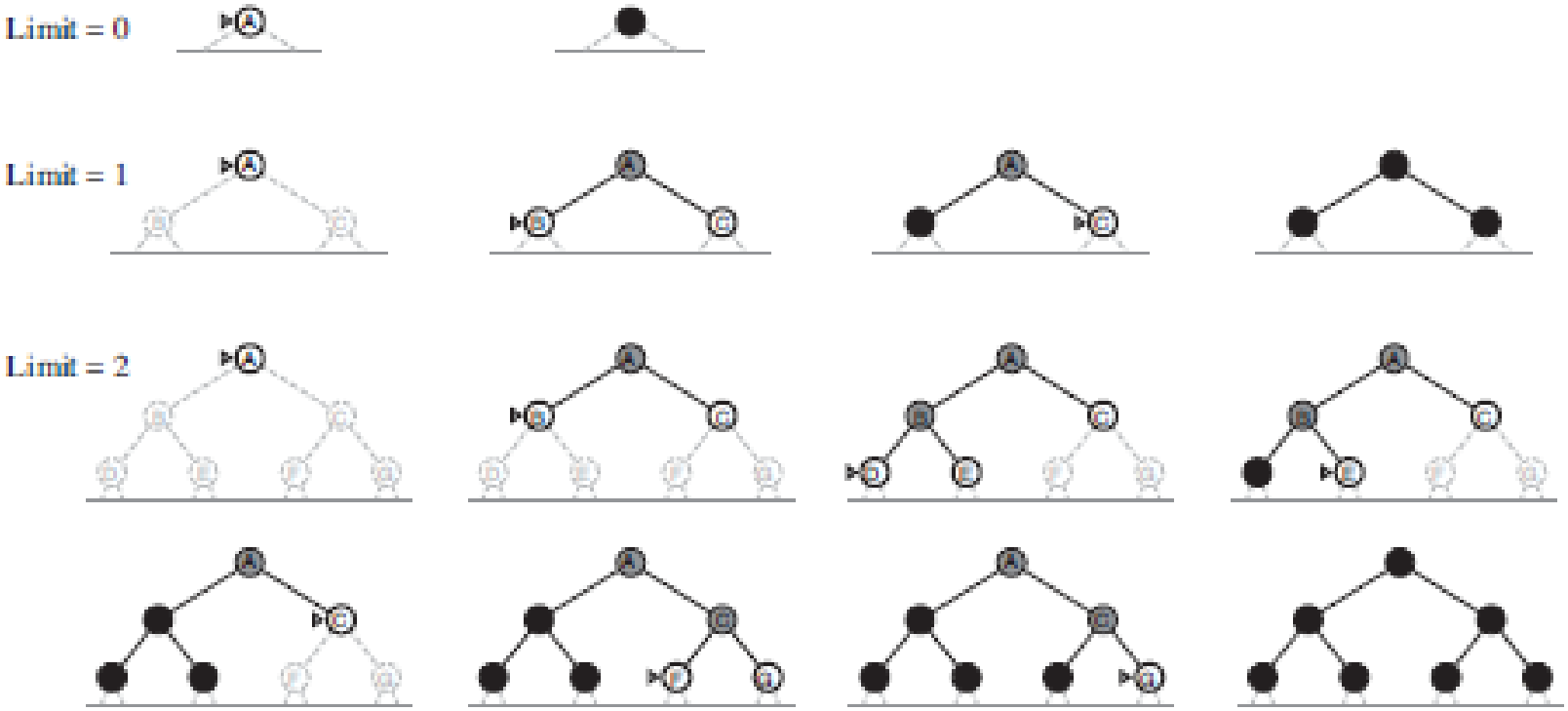
DFS limitado - Propiedades

- **Completitud**: No es completo en el caso de L < d, completo en el caso de L > d.
- **Óptimo**: No es óptimo.
- **Complejidad temporal**: En el peor caso $O(b^L)$, siendo m la longitud del camino más largo.
- **Complejidad espacial**: O(bL)
  - Solo mantiene nodos el camino actual.

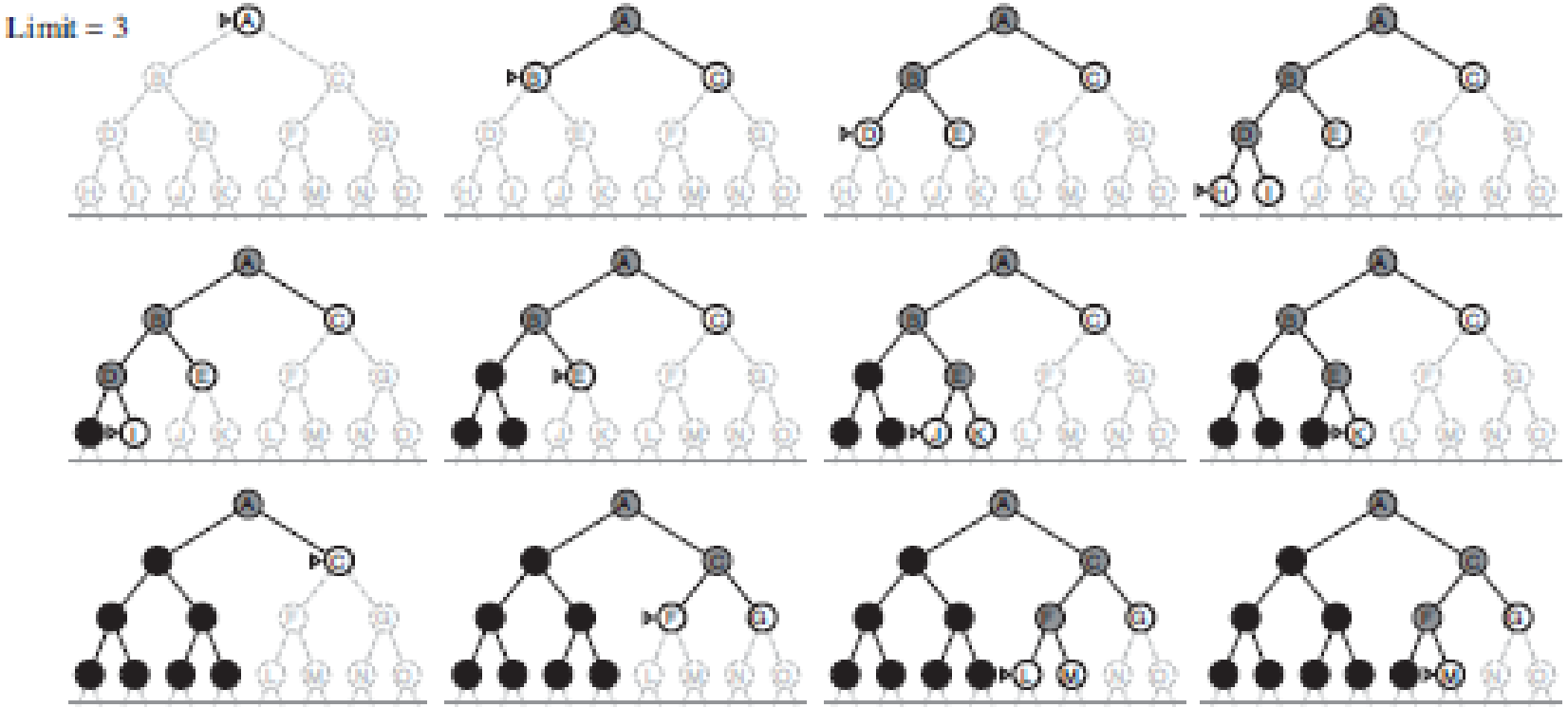
DFS iterativo (1)

- Evolución de DFS limitada.
- Se empieza en L = 0, incrementa el límite de profundidad realizando una DFS limitada para cada límite.
- Para si no encuentra solución o si el limite no se puede incrementar.

DFS iterativo (2)



DFS iterativo (3)



DFS iterativo - Propiedades

- **Completitud**: Siempre que el factor b sea finito
- **Óptimo**: Si todas las acciones tienen el mismo coste.
- **Complejidad temporal**: En el peor caso $O(b^d)$, siendo m la longitud del camino más largo.
- **Complejidad espacial**: O(bd)
  - Solo mantiene nodos el camino actual.

Observación reflexiva #7

Compara el nº de nodos recorridos en la búsqueda en profundidad limitada y búsqueda en profundidad iterativa.

$N_L = b0 + b1 + b2 + \ldots + bd\text{-}2 + bd\text{-}1 + bd$

$N_I = (d+1)b0 + db1 + (d\text{-}1)b2 + \ldots + 3bd\text{-}2 + 2bd\text{-}1 + 1bd$

Imagina que b= 10 y d = 5

Comparación de estrategias no informadas

Criterio	BFS	Coste uniforme	DFS	DFS limitado	DFS Iterativo
Completitud	Sí[1]	Sí[1,2]	No	No	Sí[1]
Tiempo	$O(b^{d+1})$	$O(b^{1+[C^* / \varepsilon]})$	$O(b^m)$	$O(b^l)$	$O(b^d)$
Espacio	$O(b^{d+1})$	$O(b^{1+[C^* / \varepsilon]})$	O(bm)	O(bl)	(Obd)
Óptimo	Sí[3]	Sí	No	No	Sí[3]

1: Completo si b es finito
2: Completo si el coste >= ε para ε positivo
3: Si todos los costes son idénticos.

Conclusión

- Estrategia multipropósito.
- Fortalezas de búsqueda no guiada y fuerza bruta.
- No es necesario disponer de información del espacio o del objetivo para guiarlos.
- Modelos basados en objetivos.

Bibliografía

Esta presentación se basa principalmente en información recogida en las siguientes fuentes:

- Hurbans, R. (2020). *Grokking Artificial Intelligence Algorithms*. Manning Publications.
- Russell, S. & Norvig, P. (2010). *Artificial Intelligence: A modern approach. 3ª Ed*. Prentice-Hall.
- McCoy, K. F. (2010). https://www.eecis.udel.edu/~mccoy/
- Carballedo Morillo R. (2020). *Apuntes de Sistemas Inteligentes*. Universidad de Deusto.